T-Trees, Vertical Partitioning and Distributed Association Rule Mining

Frans Coenen, Paul Leng and Shakil Ahmed
Department of Computer Science, The University of Liverpool
Liverpool, L69 3BX, UK
{frans,phl,shakil}@csc.liv.ac.uk

Abstract

In this paper we consider a technique (DATA-VP) for distributed (and parallel) Association Rule Mining that makes use of a vertical partitioning technique to distribute the input data amongst processors. The proposed vertical partitioning is facilitated by a novel compressed set enumeration tree data structure (the T-tree), and an associated mining algorithm (Apriori-T), that allows for computationally effective distributed/parallel ARM when compared with existing approaches.

1 Introduction

An approach to distributed/parallel Association Rule Mining (ARM), DATA-VP, that makes use of a vertical partitioning approach to distributing the input data is described. Using this approach each partition can be mined in isolation while at the same time taking into account the possibility of the existence of large itemsets dispersed across two or more partitions. To facilitate the partitioning a compressed set enumeration tree data structure (the T-tree), developed by the authors, is used together with an associated ARM algorithm — Apriori-T. The approach described offers significant advantages with respect to computational efficiency when compared to alternative mechanisms for (a) dividing the input data between processors and/or (b) achieving distributed/parallel ARM.

2 The T-tree and the Apriori-T algorithm

A significant consideration, with respect to ARM, is the nature of the data structures used to store itemsets as the algorithm progresses. The authors have developed a number of compressed set enumeration tree structures to support ARM including the T-tree (Total Support Tree) [7, 6], and a number of serial ARM algorithms to be used in association with these structures. Of particular relevance, with respect to this paper, is the Apriori-T algorithm.

The T-tree differs from more standard set enumeration trees in that the nodes at the same level in any sub-branch are organised into 1-D arrays so that array indexes represent column numbers. For this purpose it is more convenient to build a "reverse" version of the tree (see Figure 1(a)), which permits direct indexing with attribute/column numbers. The T-tree offers two initial advantages over standard set enumeration trees: (1) fast traversal of the tree using indexing mechanisms, and (2) reduced storage, in that itemset labels are not required to be explicitly stored, and thus no sibling references (pointers) are required. The implementation of this structure is illustrated in Figure 1(b) where nodes in the T-tree are objects comprising a support value and a reference to an array of child T-tree nodes.

The Apriori-T algorithm combines the classic Apriori ARM algorithm [2] with the T-tree data structure[1]. As each level is processed, candidates are added as a new level of the T-tree, their support is counted, and those that do not reach the required support threshold pruned. When the algorithm terminates, the T-tree contains only large itemsets. At each level, new candidate itemsets of size K are generated from identified large K-1 itemsets, using the *downward closure property of itemsets*, which in turn may necessitate the inspection of neighbouring branches in the T-tree to determine if a particular K-1 subset is supported. We refer to this process as *X-checking*. Note that X-checking adds a computational overhead; offset against the additional effort required to establish whether a candidate K itemset, all of whose K-1 itemsets may not necessarily be supported, is or is not a large itemset.

The number of candidate nodes generated during the construction of a T-tree, and consequently the computational effort required, is very much dependent on the distribution of columns within the input data. Best results are produced by reordering the dataset, according to the support counts for the 1-itemsets, so that the most frequent 1-itemsets occur first ([5]).

[1]Elsewhere [5] the authors have described an algorithm, Apriori-TFP, which uses both the T-tree structure and another set-enumeration tree, the P-tree. In this paper we examine methods using only the T-tree structure.

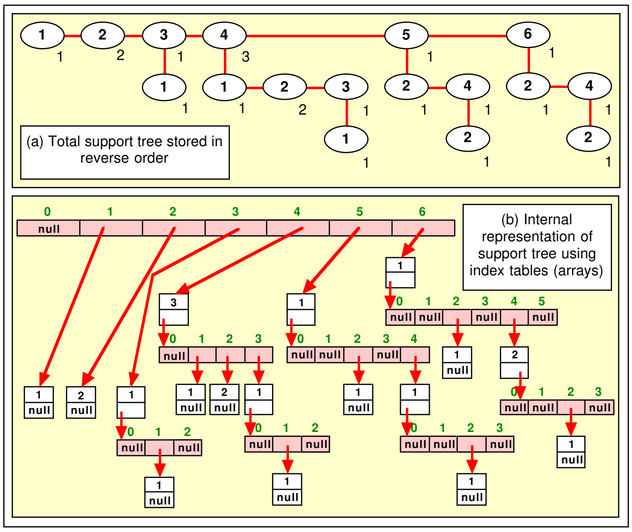

Figure 1. The T-tree for the data set $\{\{1,3,4\},\{2,4,5\},\{2,4,6\}\}$. Note that, for ease of processing, items/attributes are enumerated commencing with 1

3 The distributed Apriori-T Algorithm with Vertical Partitioning (DATA-VP)

The DATA-VP algorithm commences by distributing the input dataset over the available number of processes using a vertical partitioning strategy. Initially the set of single attributes (columns) is split equally between the available processes so that an $allocationItemSet$ (a sequence of single attributes) is defined for each process in terms of a $startColNum$ and $endColNum$:

$$allocationItemSet = \{n | startColNum < n \leq endColNum\}$$

Each process will have its own $allocationItemSet$ which is then used to determine the subset of the input dataset to be considered by the process. Using its $allocationItemSet$ each process will proceed as follows:

1. Remove all records in the input dataset that do not intersect with the $allocationItemSet$.
2. From the remaining records remove those attributes whose column number is greater than $endColNum$. We cannot remove those attributes whose identifiers are less than $startColNum$ because these may represent the "leading sub-string" of large itemset to be included in the sub T-tree counted by the process.

The input dataset distribution procedure, given an $allocationItemSet$, can be summarised as follows:

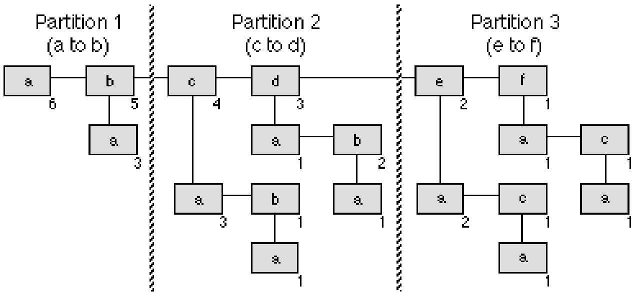

Figure 2. Distributed T-tree representing the vertical partitioning presented in the example

$$\begin{array}{l} \forall records \in input\ data \\ \quad if\ (record \cap allocationItemSet \equiv true) \\ \quad\quad record = \{n | n \in record\ n \leq endColNum\} \\ \quad else\ delete\ record \end{array}$$

For example — given the data set $\{\{a,c,f\},\{b\},\{a,c,e\},\{b,d\},\{a,e\},\{a,b,c\},\{d\},\{a,b\},\{c\},\{a,b,d\}\}$, and assuming three processes, the above partitioning process will result in three dataset partitions:

Process 1 (a to b): $\{\{a\},\{b\},\{a\},\{b\},\{a\},\{a,b\},\{\},\{a,b\},\{\},\{a,b\}\}$
Process 2 (c to d): $\{\{a,c\},\{\},\{a,c\},\{b,d\},\{\},\{a,b,c\},\{d\},\{\},\{c\},\{a,b,d\}\}$
Process 3 (e to f): $\{\{a,c,f\},\{\},\{a,c,e\},\{\},\{a,e\},\{\},\{\},\{\},\{\},\{\}\}$

Figure 2 shows the resulting sub T-trees assuming all combinations represented by each partition are supported.

Note that because the input dataset is ordered according to the frequency of 1-itemsets the size of the individual partitioned sets does not necessarily increase as the $endColNum$ approaches N (the number of columns in the input dataset); in the later partitions, the lower frequency leads to more records being eliminated. Thus the overall result of the vertical partitioning is that the overall size of the dataset (applicable to the process in question) is reduced. Once partitioning is complete each partition can be mined, using the Apriori-T algorithm, in isolation.

The DATA-VP algorithm can thus be summarised as follows:

1. Start all processes, Master plus a number of Workers.
2. Master determines the division of $allocationItemSet$ according to the total number of available processes and transmits this information to the Workers.
3. Each process then generates a T-tree for its allocated partition (a sub tree of the final T-tree).
4. On completion each process transmits its partition of the T-tree to all other processes which are then merged

into a single T-tree (so that each process has a copy of the final T-tree ready for the next stage in the ARM process — rule generation).

The process begins with a top-level "tree" comprising only those 1-itemsets included in its $allocationItemSet$. The process will then generate the candidate 2-itemsets that belong in its sub T-tree. These will comprise all the possible pairings between each element in the $allocationItemSet$ and the lexicographically preceding attribute; of those elements (see Figure 2). The support values for the candidate 2-itemsets are then determined and the sets pruned to leave only large 2-itemsets. Candidate sets for the third level are then generated. Again, no attributes from succeeding $allocationItemSet$ are considered, but the possible candidates will, in general, have subsets which are contained in preceding $allocationItemSet$ and which, therefore, are being counted by some other process. To avoid the overhead involved in X-checking, which in this case would require message-passing between the processes concerned, X-checking does not take place. Instead, the process will generate its candidates assuming, where necessary, that any subsets outside its local T-tree are large.

4 Related Work

Generally speaking two types of parallel/distributed ARM algorithms can be identified [4]:

Data Distribution (e.g. Agrawal and Schafer's "count distribution algorithm" [1]): The data is apportioned amongst the processes, typically by "horizontally" segmenting the dataset into sets of records. Each process then mines its allocated segment (exchanging information on-route as necessary).

Task Distribution (e.g. Agrawal is Schafer's "data distribution algorithm" [1]): Each process has access to the entire dataset but is responsible for some subset of the set of candidate itemsets generated at each level.

To evaluate the DATA-VP algorithm (see below) we will compare its performance with a data parallel Apriori-T algorithm (DATA-DD), and an Apriori-T version of the task parallel approach (DATA-TD).

5 Evaluation

The implmentation of the algorithms was carried oput using Java Spaces [3]. The evaluation presented here uses five processes and the data set `T20.I10.D500K.N500` (generated using the IBM Quest generator used in Agrawal and Srikant [2]), although the research team have obtained similar results using other data sets. In all case the dataset has been preprocessed so that it is ordered.

	Support %			
	2.0	1.5	1.0	0.5
DATA-DD	3472	4621	8855	36119
DATA-TD	695	924	1771	7225
DATA-VP	11	21	60	228

Table 1. Average Total size (Kbytes) of messages sent and read/taken per process

5.1 Number of T-tree Messages

The most significant overhead of any distributed/parallel ARM algorithm is the number of messages sent and received between processes. For DATA-DD and DATA-TD processes are required to exchange information as each level of the T-tree is constructed; the number of levels will equal the size of the largest supported set. For DATA-VP the number of messages sent is independent of the number of levels in the T-tree; communication takes place only at the end of the tree construction. DATA-VP therefore has a clear advantage in terms of the number of messages sent.

5.2 Amount of Data Sent and Received

Table 1 shows the average amount of data sent and received by each process for each of the Apriori-T algorithms under consideration assuming five processes. Note that:

- With respect to DATA-DD, for each generated T-tree level, un-pruned levels of the T-tree are passed from one process to another (and then pruned).
- In the case of DATA-TD pruned sections of levels in the T-tree are passed from one process to another.
- DATA-VP passes entire pruned sub T-trees (pruned T-tree branches, not entire levels). Consequently the amount of data passed between processes when using DATA-VP is significantly less than that associated with the other approaches.
- In the case of DATA-DD adding more processes increases the amount of communication, because all processes send data to all others. In the case of DATA-TD and DATA-VP, however, each process sends only the set of candidates it is counting or has counted, which becomes proportionately smaller as the number of processes is increased; i.e. for these methods the messaging overhead remains approximately constant.

5.3 Number of Updates

The number of support value updates/incrementations per process is a good indication of the amount of work done by each process. Table 2 gives the number of updates for

	Support %			
	2.0	1.5	1.0	0.5
DATA-VP, -DD, -TD	20	22	30	57
Apriori-T (with X-check)	99	111	148	285
Apriori-T (No X-check)	99	111	148	286

Table 2. Average number of updates ($\times 10^6$) to generate a final T-tree per process

	Support %			
	2.0	1.5	1.0	0.5
DATA-DD	13	16	25	99
Apriori-T (No X-check)	15	19	31	95
DATA-TD	9	9	16	66
DATA-VP	3	4	10	31

Table 3. Average execution time (seconds) per process

each of the algorithms under consideration for a range of support thresholds and using five processes. Note that:

- Table 2 includes, for comparison, values for the serial form of the Apriori-T algorithm with and without X-checking.
- DATA-DD, DATA-TD and DATA-VP all have the same average number of updates (as would be expected).

5.4 Execution Time

The overall execution time for each algorithm is arguably the most significant performance parameter. A set of times (seconds) is presented in Table 3. The table includes execution times using the Apriori-T serial algorithm (without X-checking as this provides a slightly better result for this data set).

In terms of execution time the Task Distribution algorithms, DATA-TD, and (especially) the vertical partitioning algorithm DATA-VP, perform much better than the data distribution algorithm (DATA-DD) because of the messaging overhead. Note also that, for DATA-DD, as further processes are added, the increasing overhead of messaging more than outweighs any gain from using additional processes, so that distribution/parallelisation becomes counter productive. DATA-TD shows some gain from the addition of further processes, however DATA-VP gives the best results and the best scaling.

6 Summary and Conclusions

We have described the DATA-VP approach to distributed/parallel ARM founded on the T-tree data structure and the associated Apriori-T algorithm. We have evaluated DATA-VP against established data and task distribution approaches. The principal advantages offered by DATA-VP are: (1) minimal amount of message passing compared to DATA-DD and DATA-TD, (2) minimal message size, especially with respect to DATA-DD but also when compared to DATA-TD, and (3) enhanced efficiency as the number of processes increases, unlike DATA-DD.

Our experimental evaluation of DATA-VP clearly demonstrates that the approach performs much better than those methods that use data and task distributed approaches. This is largely due to the T-tree data structure which: (a) facilitates vertical distribution of the input dataset, and (b) readily lends itself to distribution/parallelisation.

More generally we have demonstrated that both the T-tree data structure and the Apriori-T algorithm are good generic mechanisms that can be used effectively to implement many approaches to distributed/parallel ARM.

References

[1] Agrawal, R, and Shafer, J.C. (1996). Parallel Mining of Association Rules. IEEE Transactions on Knowledge and Data Engineering, Vol 8, No 6, pp962-969.

[2] Agrawal, R. and Srikant, R. (1994). *Fast algorithms for mining association rules*. Proc. 20th VLDB Conference, Morgan Kaufman, pp487-499.

[3] Arnold, K., Freeman, E. and Hupfer, S. (1999). *JavaSpaces: Principles, Patterns and Practice.* Addison Wesley.

[4] Chattratichat, J., Darlington, J., Ghanem, M., Guo, Y., Hning, H., Khler, M., Sutiwaraphun, J., To H. W., Yang, D. (1997). *Large Scale Data Mining: Challenges and Responses.* Proc. 3rd Int.Conf. on Knowledge Discovery and Data mining (KDD'97), pp143-146.

[5] Coenen, F. and Leng, P. (2001). *Optimising Association Rule Algorithms Using Itemset Ordering.* Research and Development in Intelligent Systems XVIII: Proc ES2001 Conference, eds M Bramer, F Coenen and A Preece, Springer, pp53-66.

[6] Coenen, F., Goulbourne, G. and Leng, P., (2003). *Tree Structures for Mining association Rules.* To appear in the journal of Data Mining and Knowledge Discovery.

[7] Goulbourne, G., Coenen, F. and Leng, P. (2000). *Algorithms for Computing Association Rules Using a Partial-Support Tree.* Journal of Knowledge-Based Systems, Vol (13), pp141-149.