Parallel Eclat with Large Data Base Parallel Algorithm and Improve its Effectiveness

Ms. Shruti Ingle[1], Mr. Abhay Kothari[2]
AITR, Indore Department of CSE

Abstract

To better utilize the aggregate computing resources of parallel machines, a localized algorithm based on parallelization of Eclat was proposed and exhibited excellent scalability. It makes use of a vertical data layout by transforming the horizontal database transactions into vertical lists of item sets. By name, the list of an item set is a sorted list of ID's for all transactions that contain the item set. Frequent k -item sets are organized into disjoint equivalence classes by common (k 1)-prefixes, so that candidate (k+1)-item sets can be generated by joining pairs of frequent k-item sets from the same classes. The support of a candidate item set can then be computed simply by intersecting the -lists of the two component subsets. Task parallelism is employed by dividing the mining tasks for different classes of item sets among the available processes. The equivalence classes of all frequent 2-itemsets are assigned to processes and the associated lists are distributed accordingly. Each process then mines frequent item sets generated from its assigned equivalence classes independently, by scanning and intersecting the local lists. The steps for the parallel Eclat algorithm are presented below for Distributed-memory multiprocessors divide the database evenly into horizontal partitions among all processes.

I. INTRODUCTION

To count supports of candidates, we need to go through transactions in the transaction database and check if transactions contain candidates. Since the transaction database is usually very large, it is not always possible to store them into main memory.

Furthermore, to check if a transaction containing an item set is also a non-trivial task. So an important consideration in frequent item set mining algorithms is the representation of the transaction database to facilitate the process of counting support. There are two layouts that algorithms usually employ to represent transaction databases: horizontal and vertical layout. In the horizontal layout, each transaction $T_i$is represented asT_i (tid,I) wheretid is the Transaction identifier and is an item set containing items occurring in the transaction. The initial transaction consists of all transactions T_i.

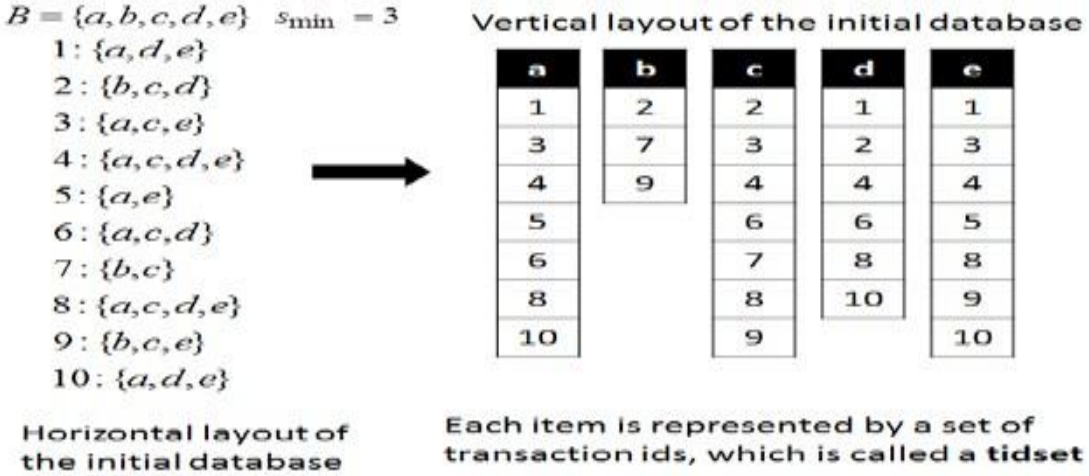

Figure 1. Horizontal and Vertical Layout

In the vertical layout, each item i_K in the item base B is represented as i_k: { i_k, $t(i_k)$} and the initial transaction database consists of all items in the item base.

For both layouts, it is possible to use the bit format to encode and also a combination of both layouts can be used[9].

To count the support of an item set X using the horizontal layout, one has to go through all transactions and check if transactions contain the itemset. Therefore, both the number of transactions in the transaction database and the size of transactions account for the consuming time of the support counting step.

When using the vertical layout, in the transaction database the support of an item set is the size of its data set. To count the support of an item set X, firstly its tidset will be generated by intersecting the tidsets of any two item sets

$$Y, Z \text{ such that } Y \cup Z = X$$

$$\text{So, } t(X) = t(Y) \cap t(Z), \text{where } Y \cup Z = X$$

This could be easily deduced from the definition of tidset. Then the support of X is the size of its tidset. In algorithms that employ the vertical layout, 2 – itemsets are generated from the initial transaction database and thennext generated by K-itemsets are from (k-1) – itemsetsAs the size of item sets increases, the size of their transition sets will decrease, using the vertical layout, counting support is usually faster and using less memory than counting support when using the horizontal layout [8].

II. RELATED WORK

Finding frequent item sets or patterns has a strong and long-standing tradition in data mining. It is a fundamental part of many data mining applications including market basket analysis, web link analysis, genome analysis and molecular fragment mining.

Since its introduction by Agrawal et al[1], it has received a great deal of attention and various efficient and sophisticated algorithms have been proposed to do frequent itemset mining. Among the best-known algorithms are Apriori, Eclat and FP-Growth.

The Apriorialgorithm[2]uses a breadth-first search and the downward closure property, in which any superset of an infrequent itemset is infrequent, to prune the search tree. Apriori usually adopts a horizontal layout to represent the transaction database and the frequency of an itemset is computed by counting its occurrence in each transaction.

FP-Growth [3] employs a divideand-conquer strategy and a FP-tree data structure to achieve a condensed representation of the transaction database. It is currently one of the fastest algorithms for frequent pattern mining.

Eclat[4] takes a depth-first search and adopts a vertical layout to represent databases, in which each item is represented by a set of transaction IDs (called a tidset) whose transactions contain the item. Tidset of an itemset is generated by intersecting tidsets of its items. Because of the depth-first search, it is difficult to utilize the downward closure property like in Apriori. However, using tidsets has an advantage that there is no need for counting support, the support of an itemset is the size of the tidset representing it. The main operation of Eclat is intersecting tidsets, thus the size of tidsets is one of main factors affecting the running time and memory usage of Eclat. The bigger tidsets are, the more time and memory are needed.

Zaki and Gouda [5] proposed a new verticaldata representation, called Diffset, and introduced dEclat, an Eclatbased algorithm using diffset. Instead of using tidsets, they use the difference of tidsets (called diffsets). Using diffsets has reduced drastically the set size representing itemsets and thus operations on sets are much faster. dEclat had been shown to achieve significant improvements in performance as well as memory usage over Eclat, especially on dense databases[5]. However, when the dataset is sparse, diffset loses its advantage over tidset. Therefore, Zaki and Gouda suggested using tidset format at the start for sparse databases and then switching to different set format later when a switching condition is met.

III. PROPOSE WORK PARALLEL APPROACH FOR ECLAT

There is a master node, which acts as a coordinator; it is in charge of assigning equivalence classes in the initial equivalence class to slave nodes. When a node has finish an equivalence class from the initial equivalence class, it will ask master node for another one, if there is no more equivalence class in the initial equivalence class, the node will become idle and it is called a free node. The master will keep a list of free nodes. When the master node cannot return an equivalence class in the initial equivalence class to a

slave node; the slave node will be added into the list of free nodes.

When a slave node has to process an equivalence, whose estimation load is over some threshold, it is called an overload node. The overload node will ask the master node for a free node, the master will then inform both the overload node and the free node about their partner. The overload node and free node then communicate to transfer job from the overload node to the free node.

The pseudo code for the scheme is as below: **Slave node:**

The first parallel approach for Eclat was proposed by Zaki et al [11].The basic idea is that the search tree could be divided into sub trees of equivalence classes. And since generating item sets in sub trees of equivalence classes is independent from each other, we could do frequent itemsets mining in subtrees of equivalence classes in parallel.So, the straightforward approach to parallelize parallel Eclat, is to consider equal equivalence class as a job.And we can distribute jobs to different nodes and nodes could work on jobs without any synchronization.

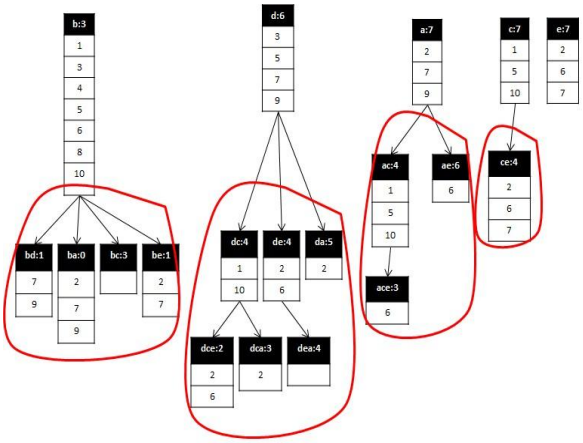

Figure 2: Parallel Approach

This seems a perfect parallel scheme. The workloads here are not equal and we may encounter the problem of load unbalancing. Experiments showed that for some datasets, classes are extremely different. In these cases, even if we add more nodes to our system, we cannot improve the overall performance.

IV. RESULT

A. Comparison Between Éclat and Parallel Éclat Algorithm with Number of Support and Its Execution Time

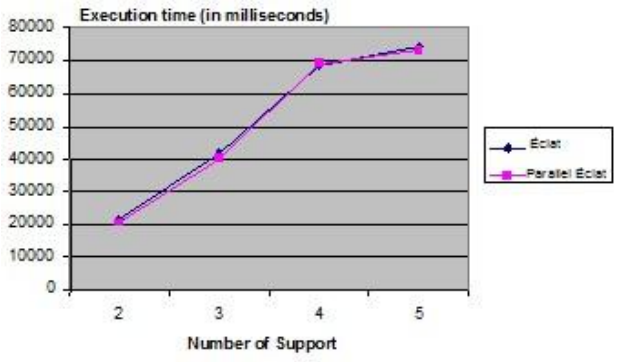

Figure 3: Numbers of Support and Its Execution Time

B. Comparison between Eclat and Parallel Eclat Algorithm with Number of Confidence and its Execution Time

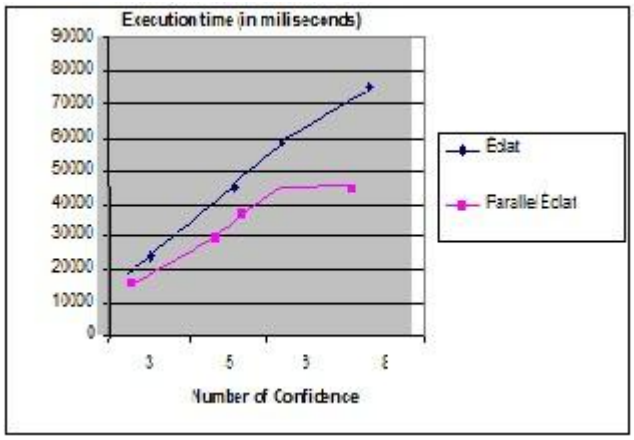

Figure 4 Numbers of Confidence and Its Execution Time

We also proposed a new parallel approach for Éclat algorithm. This new approach could address the problem of load unbalancing in the existing approach. Consequently, Éclat using this new parallel approach could exploit the power of clusters or distributed systems with many nodes. Experiments show that Éclat using our proposed parallel approach was not suffered from load unbalancing problem and the approach had also increased the scalability of Eclat when running in a parallel environment. In the case when datasets did not cause load unbalancing problem for Éclat using the existing parallel approach, Éclat using our proposed

approach was just a little slower than Eclat using the existing parallel approach, while in the case when Éclat using the existing parallel approach suffered from load unbalancing problem, Éclat using our proposed parallel approach was significantly faster than Éclat using the existing parallel approach.

V. CONCLUSION AND FUTURE WORK

In this thesis we studied quite thoroughly Parallel Éclat algorithm and introduced a novel data format for it, combination set and different set, which uses both set and different set formats to represent a database in vertical layout. This new format can fully take advantages of both set and different set formats and eliminates the need for switching from set to different Data set. Experiments showed that it reduced memory usage of Eclat. It also speeded up Éclat though not significantly. We showed the benefit of sorting different sets of data in descending order and sets in ascending order according to size, which significantly reduced the memory usage and increased parallel Éclat by several orders of magnitude. Log ECLAT algorithm combines ECLAT algorithm and method that uses special candidates to find frequent patterns from data base. In this way, Log ECLAT algorithm cans extract necessary information from the database with fewer times of creating a new database. Log ECLAT algorithm also performs well with the change of number of transactions and support thresholds in experiment. So Log ECLAT can find frequent patterns efficiently from a large database.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A.N. Swami, "Mining association rules between sets of items in large databases," in ACM SIGMOD International Conference on Management of Data, Washington, 1993.

[2] R. Agrawal, and R. Srikant, "Fast algorithms for mining association rules," in 20th International Conference on Very Large Data Bases, Washington, 1994.

[3] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in ACM SIGMOD International Conference on Management of Data, Texas, 2000.

[4] M.J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "New algorithms for fast discovery of association rules," in Third International Conference on Knowledge Discovery and Data Mining, 1997.

[5] a. K. G. M.J. Zaki, "Fast vertical mining using diffsets," in The nineth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003.

[6] Paul W. Purdom, Dirk Van Gucht, and Dennis P. Groth, "Average case performance of the apriori algorithm," vol. 33, p. 1223–1260, 2004.

[7] S. Orlando, P. Palmerini, R. Perego, and F. Silvestri, "Adaptive and resource-aware mining of frequent sets," in Proceedings of the 2002 IEEE International Conference on Data Mining, 2002.

[8] P. Shenoy, J.R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah, "Turbo-charging vertical mining of large databases," in ACM SIGMOD International Conference on Management of Data, 2000.

[9] B. Goethals, "Survey on frequent pattern mining," 2002.

[10] Yan Zhang, Fan Zhang, Jason Bakos, "Frequent Itemset Mining on LargeScale Shared Memory Machines," 2011.

[11] Mohammed JaveedZaki, SrinivasanParthasarathy, and Wei Li, "A Localized Algorithm for Parallel AssociationMining," in 9th Annual ACM Symposium on Parallel Algorithms and Architectures, 1997.

[12] "Frequent Repository," Itemset [Online]. Mining Available: Dataset http://fimi.ua.ac.be/data/.

[13] Zaki, M. J., Parthsarathy, S., Ogihara, M., and Li, W. New Algorithms for Fast Discovery of Association Rules. KDD, 283-286. 1997. Agarwal, R., Aggarwal, C., and Prasad, V.V.V. 2001.

[14] Goulbourne, G., Coenen, F., and Leng, P. H. Computing association rule using partial totals. In Proceedings of the 5th European Conference on Principles and Practice of Knowledge Discovery in Databases, 54-66. 2001.

[15] Pei, J., Han, J., Nishio, S., Tang, S., and Yang, D. H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases. Proc.2001 Int.Conf.on Data Mining. 2001.