Automatic Database Clustering Using Data Mining

Sylvain Guinepain and Le Gruenwald[1]
School of Computer Science
The University of Oklahoma
Norman, OK 73019, USA
{Sylvain.Guinepain, ggruenwald}@ou.edu

Abstract

Because of data proliferation, efficient access methods and data storage techniques have become increasingly critical to maintain an acceptable query response time. One way to improve query response time is to reduce the number of disk I/Os by partitioning the database vertically (attribute clustering) and/or horizontally (record clustering). A clustering is optimized for a given set of queries. However in dynamic systems the queries change with time, the clustering in place becomes obsolete, and the database needs to be re-clustered dynamically. In this paper we discuss an efficient algorithm[1] for attribute clustering that dynamically and automatically generate attribute clusters based on closed item sets mined from the attributes sets found in the queries running against the database.

1. Introduction

Databases, especially data warehouses and temporal databases, can become quite large. The usefulness and usability of these databases highly depend on how quickly data can be retrieved. Consequently, data has to be organized in such a way that it can be retrieved efficiently. One big concern when using such databases is the number of I/Os required in response to a query. There are four common ways to reduce this cost: indexing, buffering, clustering and parallelism. While either technique can be implemented by itself, it is obvious that clustering is an indispensable complement to the others. For instance it is easy to see that a buffering would be inefficient if the underlying data was not clustered in which case the buffer manager would spend time swapping data pages in and out of main memory to satisfy the queries. The same argument can be made for indexing and parallelism. It is for that reason that our research focuses on data clustering on disk. Clustering can take place along two dimensions: records clustering and attribute clustering. In this paper our focus is on attribute clustering. In attribute clustering, attributes of a relation are divided into groups based on their affinity. Clusters consist of smaller records, therefore, fewer pages from secondary memory are accessed to process transactions that retrieve or update only some attributes from the relation, instead of the entire record (Navathe, 1984). This leads to better query performance.

Another issue faced by today's computing world is that the ever-growing size and number of databases to monitor is becoming overwhelming and human attention has become a precious resource (Bernstein, 1998). To solve this problem the computing world is relying more and more on automated self-managing systems capable of making intelligent decision on their own. The area of autonomic computing has been getting a lot of attention (Agrawal, 2004, SAACS, 2004; AMS, 2003, Chaudhuri, 1998). Our goal in this research is to automate the clustering process. In this paper we describe an efficient algorithm for attribute clustering that automatically generates attribute clusters based on closed item sets mined from the attributes sets found in the queries running against the database.

The remainder of this paper is organized as follows: in Section 2 we review the relevant literature in the areas of traditional attribute clustering, data mining clustering and autonomic computing. In Section 3 we describe our autonomic attribute clustering algorithm. Finally we give our conclusions in Section 4.

2. Literature Review

The first well-known attribute clustering technique is credited to (McCormick, 1972) with his Bond Energy Algorithm (BEA). The purpose of the BEA is to identify and display natural variable groups and clusters that occur in complex data arrays. This task is accomplished by permuting the rows and columns of an input data array in such a way as to push the numerically larger array elements together. Informative permutations of the input data array can be found via a measure of clumpiness

[1] This work was partially supported (while serving at) National Science Foundation

(ME) of an array, which assumes large values when associated with the more informative row and column permutations. Maximizing the ME by row and column permutations serves to create strong 'bond energies' by driving the larger array elements together. The main problem with the BEA algorithm is that when the resulting matrix is in block diagonal form it is hard to determine how many clusters there are and what attributes they contain. The interpretation is subjective and therefore requires human input and cannot be considered reliable.

Navathe's Vertical Partitioning (NVP) (Navathe, 1984) improves upon BEA by providing a two-phase algorithm to determine the vertical partitions. NVP starts by building an AA (attributes affinity) matrix containing all pairs of attributes in the database. The BEA is then used to rearrange the rows and columns of the AA matrix such that the value of the global affinity function is maximized. The rearranged matrix is called the clustered affinity (CA) matrix, which then becomes an input to the second phase of the technique called the Binary Vertical Partitioning (BVP) algorithm. BVP recursively partitions the CA matrix into two halves in order to minimize the number of transactions that access attributes in both the halves. This technique has two drawbacks: 1) the objective function to maximize in phase 2 is subjective and alternative functions could produce different results., and 2)the solution only contains two clusters of attributes.

In the Optimal Binary Partitioning algorithm (OBP) (Chu, 1993) the authors introduced a transaction-based vertical partitioning technique in which the attributes of a relation are partitioned according to a set of transactions. Since transactions carry more semantic meaning than attributes, this approach allows the optimization of the partition based on a selected set of important transactions. An optimal binary partitioning algorithm, OBP, based on the branch and bound method, is presented with the worst case complexity of $O(2^n)$ where n is the number of transactions. Starting from the access pattern matrix (attribute usage matrix), the following concepts are used by the OBP algorithm. A self-contained fragment Ti is the set of attributes that transaction i accesses. The union of such self-contained fragments is called a contained fragment. A binary cut that partitions the attributes into two sets in which at least one of them is a contained fragment is also called a reasonable cut. The OBP algorithm uses a branch and bound algorithm to derive all the reasonable cuts. A cost function is then applied to determine the optimal binary partitioning. OBP suffers of two problems: the number of possible partitions to examine in order to find the optimal binary partitioning could be high and the resulting partitioning contains only two clusters.

(Hartuv, 2000) proposed a clustering technique based on graph connectivity that aims at partitioning gene expression data in the field of bio-informatics. The similarity data is used to form a similarity graph in which vertices correspond to elements and edges connect elements with similarity values above some threshold. In that graph, clusters are highly connected sub-graphs, defined as sub-graphs whose edge connectivity exceeds half of the number of vertices. The main disadvantage is that the query frequencies are not taken into account. Any two attributes queried together more often than a threshold are linked in the graph. This implies that the graph's edges are un-weighted and all similarity links are considered equal. This seldom is the case in database applications and would result in an inaccurate solution.

(Agrawal, 2004) is Microsoft's solution to the automatic clustering problem. It relies on data mining techniques. Using the attribute affinity matrix, the algorithm mines the frequent item sets of attributes and retains the top *k* ordered by confidence level. Each attribute-set forms a binary partition: attributes in the sets and attributes not in the set. The algorithm then determines which such binary partition is optimal for each individual query. The cost is obtained by creating two sub-tables corresponding to the two clusters and running the query through the query optimizer to obtain its cost. Then a merging step combines the resulting binary clusterings two at a time and evaluates the cost of all possible merged partitions. In the end the merged partition with the best cost is selected for the table. The authors clearly state that their goal is "to optimize performance of a database for a given workload". This means that this clustering is static and, given its ties to other database objects such as indices, it is not possible to convert it into a dynamic solution.

Any review of clustering techniques would not be complete without mentioning data mining clustering. Data mining clustering groups data items together by finding similarities in the data itself. To accomplish this data mining clustering algorithm use a similarity measure or distance function to determine the distance/similarity between any two data items (Dunham, 2003). Elements in the same cluster are alike and elements in different clusters are not alike. Data mining clustering is not a viable solution to solve the automatic attribute clustering problem because it groups data items based on similarities found in the actual data, not based on attribute usage by queries. Thus two data items could be stored together because they were found to be similar with respect to a distance function but rarely be accessed together.

As we have just seen none of the attribute-clustering algorithm reviewed in the literature is autonomic. Some

require manual and subjective interpretation of the results; some only produce two clusters of attributes; others use subjective parameters that result in sub-optimal solutions if their values are not chosen carefully. In the next section we describe our proposed attribute-clustering algorithm. It is autonomic, can produce any number of clusters and requires no parameters.

3. Proposed Attribute Clustering

The attribute clustering is done in four steps, which are briefly described in this section. The detailed and complete algorithm can be found in [Guinepain, 2006]. In Step 1, we build a frequency-weighted attribute usage matrix. In Step 2, we mine the closed item sets (CIS) of attributes. A closed item set is a maximal item set contained in the same transactions. This information tells us what attributes have high affinity, i.e., are often accessed together. In Step 3, The closed item sets mined in Step 2 are modified in such a way that the original tuples can be reconstructed through a natural join after the partitioning has taken place. We restrict the set of CIS considered to a) CIS containing attributes from the same relation and b) CIS containing attributes from several relations as long as the cardinality between all the relations containing these attributes is 1 to 1. Every CIS that does not contain the primary key (PK) of the relation will be augmented with the primary key attributes. Note that if the CIS contains attributes from multiple relations with 1-to-1 relationships, then it suffices to augment the Frequent CIS (FCIS) with the primary key attribute(s) of the first relation if necessary.

In Step 4, we use a branch and bound type algorithm that examines all clustering solutions of attributes such that a solution contains at least one cluster that is a modified closed item set (MCIS). The solution with the lowest is the one selected as our next vertical clustering of attributes. The query response time is not an accurate measure of the cost of a query since it varies with the system load, buffering, and indexing. The cost of running a query will therefore be given by the estimated number of logical reads returned by the query optimizer.

3.1 Step 1: Build the Frequency-Weighted Attribute Usage Matrix

When a query is run against the system, the following information concerning that query is logged: the SQL query, the time the query was executed, the attributes list associated with the query, the query result set size in terms of the number of tuples returned and the number of disk blocks accessed to answer the query. The set of queries considered by our technique will be referred to as $Q = \{ q_i, 1 \le i \le p$, where p is the total number of queries.

An example of a log entry is shown here which would produce the first row of the affinity matrix in Table 1.

SQL QUERY: SELECT a, c, d FROM T1 WHERE a BETWEEN 1 AND 10 AND d=c
ATTRIBUTES: T1.a, T1.c, T1.d
BLOCKS ACCESSED: 4

Using the logged information, this step derives the attributes usage matrix similar to the one in Table 1. The attribute usage matrix contains a row for each query considered by our technique and a column for each attribute in the database. If a query requests a particular attribute, the intersection of the query row and the attribute column will contain a "1", and "0" otherwise. The cost of building and updating the first matrix is linear in terms of the number of entries in the log.

Queries	*Attributes*						Record set size	Query frequency (%)
	a	b	c	d	e	f		
q_1	1	0	1	1	0	0	15000	10
q_2	1	1	1	0	1	0	36200	20
q_3	0	1	0	0	1	0	32000	30
q_4	0	1	1	0	1	0	34300	40

Table 1: Example of an Attributes Usage Matrix

The attributes in this example are defined in Table 2:

Attribute Id	Attribute Name	Attribute Database Table	Data Type	Size (in bits)
1	A (PK)	Table1	Short	16
2	B	Table1	Short	16
3	C	Table1	Long	32
4	D	Table1	Byte	8
5	E	Table1	Short	16
6	F	Table1	Text	8

Table 2: Example of an Attributes Information Table

Queries	*Attributes*					
	A	B	C	D	E	F
q_1	15000	0	15000	15000	0	0
q_2	36200	36200	36200	0	36200	0
q_3	0	32000	0	0	32000	0
q_4	0	34300	34300	0	34300	0

Table 3: Result Set Size-Weighted Attribute Usage Matrix

Using the attribute usage matrix we can then build a result set size-weighted attribute matrix (Table 3) by multiplying each row by the size of its result set. This matrix is only an intermediate matrix towards building the frequency weighted attribute usage matrix (Table 4).

Next, we multiply each row by its corresponding query frequency to obtain the frequency weighted attribute usage matrix (Table 4):

Queries	*Attributes*					
	a	b	c	d	e	f
q_1	150000	0	150000	150000	0	0
q_2	724000	724000	724000	0	724000	0
q_3	0	960000	0	0	960000	0
q_4	0	1372000	1372000	0	1372000	0

Table 4: Frequency Weighted Attribute Usage Matrix

The interpretation of the first row of this matrix (query q_1) is as follows: on average 150,000 tuples retrieved from the database out of every 3,206,000 (3,206,000 = 150,000 + 724,000 + 960,000 + 1,372,000) contain exactly the attributes a, c, and d.

3.2 Step 2: Mining the Closed Item sets

Attributes that are frequently queried together should be stored together. If we consider database attributes as items and queries as transactions, the problem of identifying attributes frequently queried together is similar to the data mining association rules problem of finding frequent (also called large) item sets, which is described below.

3.2.1 Frequent Item Sets (Dunham, 2003)

A frequent item set is an item set which is present in a number of transactions greater than a support threshold, s. For example, from Table 4, we see that {b, c} is present in 2.096,000 (= 724,000 + 1,372,000) tuples out of every 3,206,000. Therefore the item set {b, c} has a support of 65.38% (2,096,000 / 3,206,000). The item set {a, c, d} is present in only 150,000 out of 3,206,000 tuples retrieved, giving it a support of 4.68%. If we set the support threshold at 20%, {b, c} would be a frequent (or large) item set but {a, c, d} would not be. A subset of the set of frequent item sets is the set of frequent closed item sets defined as follows.

3.2.2 Closed Item sets (Pasquier, 1999)

A closed itemset X is a set that meets the following two conditions:

1) All members of X appear in the same transactions.
2) There exists no item set X' such that:
   2.1) X' is a proper superset of X and
   2.2) Every transaction containing X also contains X'.

In other words, a closed item set is a maximal item set contained in the same transactions.

Mathematically, the problem is described as follows (Durand, 2002):

Let D = (O, I, R) be a data mining context, O a set of transactions, I a set of items, and R a binary relation between transactions and items. For $O \subseteq O$ and $I \subseteq I$, we define: $f(O) = \{ i \in I \mid \forall o \in O, (o,i) \in R \}$ and $g(I) = \{ o \in O \mid \forall i \in I, (o,i) \in R \}$. $f(O)$ associates with O, items common to all transactions $o \in O$, and $g(I)$ associates with I, transactions related to all items $i \in I$. The operators $h = f \circ g$ and $h' = g \circ f$ are the Galois closure operators (Pasquier, 1999). Let X be an item set, $X \in I$. X is a closed item set iff $h(X) = X$. In other words, a closed item set is a maximal set of items shared by a set of transactions. Closed item sets capture all similarities among a set of transactions. For example, from Table 1 we get $g(\{b, c\}) = \{q_2, q_4\}$ and $f(g(\{b, c\}) = f(\{q_2, q_4\}) = \{b, c, e\}$. Since $f(g(\{b, c\}) \neq \{b, c\}$, $\{b, c\}$ is not a closed item set. However, $g(\{b, c, e\}) = \{q_2, q_4\}$ and $f(g(\{b, c, e\}) = f(\{q_2, q_4\}) = \{b, c, e\}$. Since $f(g(\{b, c, e\}) = \{b, c, e\}$, $\{b, c, e\}$ is a closed item set.

Because {b, c, e} is a closed item set and {b, c} is not, by definition, this means that when attributes 'b' and 'c' are queried, attribute 'e' is always queried along with them. This implies that it suffices to consider the closed item sets as clusters of attributes and there is no need to consider all their subsets that are frequent item sets. Similarly to ordinary item sets, if a closed item sets has a support greater than a predefined threshold, the closed item set is said to be frequent. The advantage of using the set of frequent closed item sets is that it is a subset of the set of frequent item sets. Our first experiments have shown that there are indeed much fewer frequent closed item sets than there are frequent item sets. Many algorithms for mining closed item sets exist, (Pasquier, 1999) and CHARM (Zaki, 2002).

Using the data in Table 4, the list of all items I = {a, b, c, d, e, f} and the closed item sets (CIS) and their respective support are:

CIS = {({a, b, c, e}, 22.6%), ({a, c}, 27.3%), ({a, c, d}, 4.7%), ({b, c, e}, 65.4%), ({b, e}, 95.3%), ({c}, 70.1%)}.

3.3 Step 3 Filtering the closed item sets

Since all of our attribute in this example come from the same table it suffices to augment, if needed, each CIS with the primary key of the relation, i.e. attribute a.

MCIS = {{a, b, c, e}, {a, c}, {a, c, d}, {a, b, e}, {a, c}}.

3.4 Step 4: Determine the Best Clustering of Attributes Based on Closed Item Sets

We wish to partition the database vertically by clustering some attributes together. A clustering solution is therefore a partition of the set of attributes I = {a, b, c, d, e, f}. For example {{a, c, e}, {a, b, d}, {a, f}} is a clustering solution containing 3 clusters. Each cluster

containing a copy of the relation's primary for join purposes.

The partitioning problem is a very difficult problem and the number of solutions is equal to the Bell number (Bell Number Reference, 2004) that follows the following recurrence relation: $b_{n+1} = \sum_{k=0}^{n} b_k \binom{n}{k}$, where n is the total number of attributes in I (n = || I ||). In our case, however, the search space is greatly reduced since we only consider the clustering solution that contains at least one cluster that is a closed item set. The attributes not present in any set in FCIS will each be clustered in its own blocks along with a copy of the primary key.

3.4.1 The Vertical Partitioning Algorithm

A candidate clustering solution (CCS) is any complete partition of the set of items I. A complete partition of the set of items is a partition that contains every item in I. In other words the union of all sets contained in the partition is equal to I. For a candidate clustering solution to be valid it must contain at least one cluster that is a modified closed item set.

The algorithm essentially starts with an empty solution and adds clusters of attributes that are MCIS to the solution until the solution forms a complete partition of the set of attributes. When the solution is complete its cost is measured by running all the queries in Q through the query optimizer which estimates the number of logical reads. Note that the queries are not actually run. All possible combinations of MCIS as clusters will be considered.

Our preliminary test showed that the time required to mine closed item sets is so small that there is no need to set up frequency threshold. As a result the quality of the solution produced is the best possible. We ran simulations with a simpler model that did not require primary key duplication and where the cost was an estimated number of disk blocks accessed and found out that our technique returned a solution 57% better than OBP on average while running 500 faster on the TPC-R benchmark data (TPC, 2005). We also noted that using a simpler model, our algorithm systematically returned the same solution as a brute force algorithm that found the optimum solution.

4 Conclusion

We have just presented an autonomic attribute clustering algorithm that is based on data mining techniques. The idea is to form clusters of attributes that correspond to closed item sets of attributes found in the queries. Preliminary tests results indicate that this algorithm returns an excellent quality solution in record time.

5 References

(Agrawal, 2004) Sanjay Agrawal, V. Narasayya, B. Yang, "Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design", SIGMOD, June 2004.

(AMS, 2003) 5th Annual International Workshop on Active Middleware Services. June 2003.

(Bell Number Reference, 2005) http://mathforum.org/advanced/robertd/bell.html. Accessed 10/14/2005.

(Bernstein, 1998) Phil Bernstein, M. Brodie, S. Ceri, D. DeWitt, M. Frankiln, H. Garcia-Molina, J. Gray, J. Held, J. Hellerstein, H. V. Jagadish, M. Lesk, D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker, and J. Ullman, "The Asilomar Report on Database Research", ACM SIGMOD Record,,Vol. 27 , Issue 4 , Dec. 1998.

(Chaudhuri, 1998) Surajit Chaudhuri and V. Narasayya, 'AutoAdmin "What-if" Index Analysis Utility", SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 1998.

(Chu, 1993) Wesley W. Chu and I. Ieong, "A Transaction-Based Approach to Vertical Partitioning for Relational Database Systems", IEEE Transactions on Software Engineering, Vol. 19, No. 8, August 1993.

(Dunham, 2003) Margaret H. Dunham, "Data Mining: Introduction and Advanced Topics", Prentice Hall, 2003.

(Durand, 2002) Nicolas Durand and B. Cremilleux, "Extraction of a Subset of Concepts from Frequent Closed Itemset Lattice: A New Approach of Meaningful Clusters Discovery" International Workshop on Advances in Formal Concept Analysis for Knowledge Discovery in Databases, July 2002.

(Guinepain, 2006) Sylvain Guinepain and L. Gruenwald, "An Efficient Algorithm for Automatic Attribute Clustering using Data Mining," Technical Report, The University of Oklahoma, School of Computer Science, May 2006.

(Hartuv, 2000) Erez Hartux, and Ron Shamir, "A Clustering Algorithm Based on Graph Connectivity", Information Processing Letters, Vol. 76, No. 4-6, 2000.

(McCormick, 1972) McCormick, W. T. Schweitzer P. J., and White T. W., "Problem decomposition and data reorganization by a clustering technique", Oper. Res. 20, 5, September 1972.

(Navathe, 1984) Shamkant Navathe, S. Ceri, G. Wierhold, and J. Dou, "Vertical Partitioning Algorithms for Database Design", ACM Transactions on Database Systems, Vol. 9, No. 4, December 1984.

(Pasquier, 1999) Nicolas Pasquier, Y. Bastidem, R. Taouil, and L. Lakhal, "Efficient Mining of Association Rules Using Closed Itemset Lattices", Information Systems, Vol. 24, No. 1, 1999.

(SAACS, 2004) 2nd International Workshop on Self-Adaptive and Autonomic Computing Systems. DEXA 2004.

(TPC, 2005) www.tpc.org. Accessed 10/14/2004.

(Zaki, 2002) Mohammed J. Zaki and C. Hsiao, "CHARM: An Efficient Algorithm for Closed Itemset Mining", SIAM International Conference on Data Mining, April 2002.