AutoClust: Autonomous Database Partitioning using Data Mining for Relations with One-to-One and One-to-Many Relationships

Technical Report

Le Gruenwald, Sylvain Guinepain, Zhenbo Xing
School of Computer Science
University of Oklahoma

July 2010

1. INTRODUCTION

Databases/files in many applications, such as those that process large volumes of sales transactions, medical records and scientific data, can be very large. The usefulness of these databases/files highly depends on how quickly data can be retrieved. Clever data organization and access is one of the best ways to improve retrieval speed. By improving the data storage, we can reduce the number of disk I/Os and thereby reduce the query response time. In this proposal we combine database/file clustering and parallelism to reduce the cost of I/Os and improve system throughput. We propose an automatic and dynamic database/file clustering algorithm, AutoClust, that uses data mining to produce clusters/partitions containing data frequently accessed together. Clusters consist of smaller records, therefore, fewer pages from secondary memory are accessed to process queries that access only some attributes, instead of the entire record [15]. This leads to better query performance.

AutoClust is an automated and dynamic clustering technique. It is well documented that with the ever-growing size and number of databases to monitor, human attention has become a precious resource [5]. In response to this concern, 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 [2, 19, 6, 20]. For AutoClust to be useful it should be fully automated, which implies that no human intervention is needed during the clustering process. To do this, the algorithm generates attribute clusters automatically based on closed item sets [18] mined from the attributes sets found in the queries running against the database/file. The algorithm is capable of re-clustering the database/file in order to continue achieving good system performance despite changes in the data and/or query sets.

In this technical report, we first present Autoclust with one-to-one relationships among relations and then extend it to include one-to-many relationships among relations. We then present the experimental results comparing AutoClust with the case of no clustering using the TPCH [21] benchmark.

2. RELATED WORK

The clustering problem is a very difficult problem and the number of solutions is equal to the Bell number [5] 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 we wish to cluster. The first well-known attribute clustering technique is credited to [14] 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 by permuting the rows and columns so as to push the numerically larger array elements together. The resulting matrix is in block diagonal form. It is hard however 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) [15] added a second phase to the BEA algorithm in an effort to reduce the subjectivity of the final interpretation. In the phase, the author performs the BEA algorithm against an affinity matrix containing all pairs of attributes in the database. The BEA is then used to rearrange the rows and columns of the 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.

Data mining clustering is another tool to group data items together by finding similarities in the data itself. To accomplish this, data mining clustering algorithms use a similarity measure or distance function to determine the distance/similarity between any two data items [8]. The objective is to create groups or clusters where the elements within each group are alike and elements between groups are dissimilar. Elements in the same cluster are alike and elements in different clusters are not alike.

The three techniques described so far suffer the same problem. These techniques group 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 but rarely be accessed together. The problem with such an approach is that it only helps with record clustering and does not help reducing the number of I/Os for queries accessing only few attributes of a relation. Realizing that the next evolution in clustering algorithm was transaction-based clustering.

A transaction-based vertical partitioning technique, the Optimal Binary Partitioning algorithm (OBP), is proposed in [7] where the attributes of a relation are partitioned according to a set of transactions. This concept derives from the fact that transactions carry more semantic meaning than attributes. The technique creates clusters by splitting up the set of attributes recursively. Each cluster of attributes is separated into two groups with each query: the attributes that are part of the query and those that are not. At each sub level of the tree, a new query is used to split the clusters further. In the end the algorithm produces a tree whose leaves contain the split up clusters of attributes. A cost function is then applied to determine the optimal binary partitioning while merging adjacent leaves of the tree. The disadvantages of this technique are that it may have to examine a large number of possible partitions in order to find the optimal binary partitioning and it produces partitions that contain only two clusters. Other more recent clustering algorithms can cluster attributes in more than 2 clusters. This allows for better performance when the relations have many attributes and there are many queries.

A graph theory approach to the clustering problem was proposed in [12] with a clustering technique based on graph connectivity. 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. Clusters are highly connected sub-graphs, which are defined as subgraphs whose edge connectivity exceeds half of the number of vertices. This technique does not take into account query frequencies and the resulting solution could contain clusters that favour infrequent queries over more frequent ones.

Microsoft's data mining based solution to the automatic clustering problem was presented in [2, 3]. 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 clusters two at a time and evaluates the cost of all possible merged partitions and selects the lowest cost cluster. 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.

A static horizontal partitioning method introduced in Oracle 11gR1 that equi-partitions tables with a parent-child referential relationship between them was presented in [10]. The partitioning key from the parent table is used without duplicating the key columns. This technique does not deal with attribute clustering. In [13] the authors propose two techniques based on workload to segment data in a column store and replicate the segments on disks. It does not deal with clustering of attributes to decide which columns to be placed in the same segment. In [17] a database partitioning technique, called AutoPart, is proposed to perform categorical and vertical partitioning on the database; however it handles neither record clustering nor re-clustering dynamically.

None of the clustering algorithms reviewed above is truly autonomous. 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. None of them is for cluster computing and none integrates attribute clustering with record clustering dynamically and automatically.

There has been work on data placement with data replication on cluster machines to improve system performance [11,1, 23]. Their main focus is to decide which replicas to place on which nodes and how to achieve tradeoffs between data consistency and application performance. None of these works deals with automatic and dynamic clustering of data for efficient data placement despite changes in the system workload, which is the goal of our research.

3. AUTOCLUST WITH ONE-TO-ONE RELATIONSHIPS AMONG RELATIONS

Our objective is to create an automatic and dynamic clustering technique that is triggered automatically when the query response time becomes higher (worse) than a user-determined threshold and that rearrange the data on disk based on attribute and record affinity patterns discovered in the query set. Our technique belongs to the category of mixed partitioning, where vertical partitioning (attribute clustering) is followed by horizontal partitioning (record clustering). Using data mining we will first partitioned the database vertically by identifying attributes that should be clustered together. Then we will partition the different partitions/clusters horizontally by clustering records within each cluster based on their co-access frequencies to produce smaller clusters that better answer the queries considered. Our dynamic automatic clustering solution works as follows. As queries are being processed, the system collects information about each query. When the system detects that the query response time is not as good as what the user expects, it will test the goodness of the record clustering in place. If no bad clustering is detected, then the system goes back to its original state and starts running queries and gathering information again. On the other hand, if a bad record clustering is detected, then the system tests for bad attribute clustering. If only a bad record clustering is detected, the data is rearranged horizontally, i.e. a record clustering is executed. If a bad attribute clustering is detected, then the attribute clustering is performed followed by the record clustering. In this section, we describe the attribute clustering of AutoClust when the relationships among relations are one-to-one.

The attribute clustering part is done in four steps. In Step 1, we build a frequency-weighted attribute usage matrix where each cell (i, j) contains the frequency of query i accessing attribute j. Table 1 shows an example of this matrix.

Table 1: Frequency Weighted Attribute Usage Matrix

Queries	**Attributes**					
	A	**B**	**C**	**D**	**E**	**F**
q_1	10	0	10	10	0	0
q_2	20	20	20	0	20	0
q_3	0	30	0	0	30	0
q_4	0	40	40	0	40	0

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 queries. This information tells us what attributes are often accessed together. 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 item sets, which is described below.

1) Frequent Item Sets [8]: 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 2, we see that {B, C} is accessed by 60% of the queries run. Therefore the item set {B, C} has a support of 60%. The item set {A, C, D} has a support of 10%. If we set the support level threshold at 20%, {B, C} would be a frequent item set but {A, C, D} would not be. A subset of the set of frequent item sets is a set of frequent closed item sets defined as follows.

2) Closed Item sets [18]: A closed item set carries more information because it is a maximal item set contained in the same queries. It meets the following two conditions:

All members of the closed item set X appear in the same queries/transactions. There exists no item set X' such that:

- X' is a proper superset of X and
- Every transaction containing X also contains X'.

Mathematically, the problem is described as follows [9]:

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 queries $o \in O$, 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 [18].

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. This greatly reduces the complexity of the clustering problem. Many algorithms for mining closed item sets exist, such as those in [18] and [22].

In Step 3, the closed item sets mined in Step 2 are augmented and filtered in such a way that the original records can be reconstructed through a natural join after the clustering has taken place. 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 one-to-one relationships, then it suffices to augment the CIS with the primary key attribute(s) of the first relation if necessary. This new set is called the augmented closed item sets (ACIS).

Table 2: Estimated I/O Cost of Candidate Clustering Solutions

Solution	Estimated Individual Query Cost (I/O)				Aggregate Cost
	q_1	q_2	q_3	q_4	
S_1= {{A, F}, {A, B}, {A, C}, {A, D}, {A, E}}	3.857	5.601	3.121	5.601	468.27
S_2 = {{A, F}, {A, B}, {A, C, D}, {A, E}}	2.658	5.779	3.121	5.779	466.95
S_3 = {{A, F}, {A, B, C, E}, {A, D}}	4.403	3.026	3.026	3.026	316.38
S_4 = {{A, F}, {A, B, E}, {A, C}, {A, D}}	3.857	4.406	1.926	4.406	360.79
S_5 = {{A, F}, {A, B, E}, {A, C, D}}	2.657	4.584	1.926	4.584	359.47

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 an ACIS. The solution with the lowest cost 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, given a clustering solution, will therefore be given by the estimated I/O cost returned by a query optimizer like the one available in the SQL Server. The total estimated cost of a query is its estimated cost multiplied by its frequency identified in Step 1. The cost of a clustering solution is the sum of the estimated costs of all queries accessing the attributes in the clustering solution. Table 2 shows an example of the results of Step 4 where the candidate clustering solution **S_3** is chosen as the clustering solution as its aggregate cost is the lowest.

The proposed algorithm currently considers only CIS that contain attributes from the same relation/file and CIS that contain attributes from several relations/files with one-to-one cardinality relationships among them. In the next section we describe how we extend the algorithm to include one-to-many relationships among relations.

4. EXTENTION OF AUTOCLUST TO INCLUDE ONE-TO-MANY RELATIONSHIPS AMONG RELATIONS

In the current AutoClust solution, only one-to-one relationships between relations are considered. For one-to-many relationships, different from one-to-one relationships, the attributes clustered together by AutoClust cannot be stored together in the same cluster on physical storage disk; in other words, the results cannot be stored in one table as the solution derived for one-to-one relationships. To resolve this problem, we modify the four steps in AutoClust which we illustrate below through an example.

Consider an example in which there are two relations with a one-to-many relationship between them, R = {$\underline{A}$, B, C, D} and S = {$\underline{E}$, F, $\underline{\underline{A}}$}. In the relation R, A is the primary key, and in the relation S, E is the primary key with A is the foreign key, which represents the connection/relation to the relation R.

Given the following transaction list and attribute usage matrix:

Table 3: Transaction List and Attribute Usage Matrix for the Example

Transactions	**Attributes in R**				**Frequency**	**Attributes in S**		
	A	**B**	**C**	**D**		**E**	**F**	***A***
T_1	1	0	1	1	10%	0	0	*1*
T_2	1	1	1	0	20%	1	0	*1*
T_3	0	1	0	0	30%	1	0	*0*
T_4	0	1	1	0	40%	1	1	*0*

The four steps of the modified AutoClust are as follows:

Step 1: Build the Frequency-Weighted Usage Matrix

Based on the Usage Matrix above, the Frequency-Weighted Usage Matrix can be built as below:

Table 4: Frequeny-Weighted Usage Matrix for the Example

Transactions	**Attributes in R (%)**					**Attributes in S (%)**		
	A	**B**	**C**	**D**	**One - Many**	**E**	**F**	***A***
T_1	10	0	10	10		0	0	*10*
T_2	20	20	20	0		20	0	*20*
T_3	0	30	0	0		30	0	*0*
T_4	0	40	40	0		40	0	*0*

Step 2: Mining the Closed Item Sets

Table 5: Closed Item Sets and Support for the Example

Closed Item Set	**Support (%)**
{A, B, C}, { E}	20
{A, C}	30
{A, C, D}	10
{B, C}, {E}	60
{B}, {E}	90
{C}	70

Step 3 Filtering the Closed Item Set

Add the primary key A to all clusters which contain attributes from the relation R, and add the primary key E and foreign key A to all clusters which contain attributes from the relation S.

Closed Item Set
{A, B, C}, { E}
{A, C}
{A, C, D}
{B, C}, {E}
{B}, {E}
{C}

{A, B, C}, {E, A}
{A, C}
{A, C, D}
{A, B}, {E, A}

Attribute F in the relation S is not accessed by any transaction, but in the later step, it would be partitioned as a cluster which contains the primary key E, F, and the foreign key A - (E, F, A).

Step 4: Determine the best solution for partitioning

The algorithm retrieves the possible solutions by constructing the Execution Tree. Since the example here has the same results as the one in the paper [Guinepain, 2008], for simplicity, the algorithm and process of constructing the tree are omitted here.

Five Solutions by constructing the execution tree:

S_1: { {E, A, F}, {A, B}, {A, C}, {A, D} }

S_2: { {E, A, F}, {A, B}, {A, C, D} }

S_3: { {E, A, F}, {A, B, C}, {E}, {A, D} }

S_4: { {E, A, F}, {A, B}, {E}, {A, C}, {A, D} }

S_5: { {E, A, F}, {A, B}, {E}, {A, C, D} }

By running the SQL Server Query Optimizer, the cost of each solution will be calculated, and the system will partition the database based on the one with the least cost.

5. SIMULATION EXPERIMENTS COMPARING AUTOCLUST WITH NO-CLUSTERING

We compared AutoClust with the case when no clustering was available using the TPCH benchmark [7]. Experiments showed that AutoClust performs better than no clustering (NoClust), which we describe in detail below.

Experiment Environment Configuration:

SQL Server Query Optimizer was used to simulate the process of queries; in other words, instead of being really executed in SQL Server, the queries are sent to Query Optimizer, and the query optimizer analyzes the queries and returns the results, which includes the I/O cost and CPU cost.

TPCH Benchmark:

The current simulators work with 7 tables and the range of amount of attributes in these tables is from 3 to 9.

Table Name	Num of Attributes	Num of Records	Data Size
CUSTOMER	8	0	0
NATION	4	25	2,575 B
ORDERS	9	1,500,000	143 MB
PART	9	200,000	25 MB
PARTSUPP	5	800,000	100 MB
REGION	3	5	510 B
SUPPLIER	7	10,000	1,172 B

Sample Results & Analysis:

The results below were obtained by running AutoClust and NoClust, respectively, on table **PART** with the same query set (result, I/O Cost returned by SQL Query Optimizer).

Algorithm	**I/O Cost**
AutoClust	0.496582529999999
NoClust	1.2749768

The I/O Cost returned by Optimizer is calculated by adding the total I/O operation costs, in short, the number represents time (*second*). Since the time each I/O operation needs is known, this number is therefore corresponding to the number of I/O operations. The I/O cost is computed as follows:

$$\text{I/O Cost} = 0.003125 + 0.00074074 * (N - 1)$$

N is the num of I/O operations. *0.003125* is the time to finish the first I/O operation, and the time to finish each sequential I/O operation is *0.00074074* second. In the example above, we have:

N = ((I/O Cost – 0.003125) / 0.00074074) + 1

$N_{AutoClust} = 666 + 1 = 667$

$N_{NoClust} = 1717 + 1 = 1718$

For the same table, AutoClust only needs 667 I/O operations while NoClust needs 1718 I/O operations. In this example, AutoClust performs three times better than NoClust.

6. FUTURE RESEARCH

For future research, we will enhance AutoClust so that it will automatically detect bad clustering and perform appropriate re-clustering. We distinguish two different types of bad clustering: bad attribute clustering and bad record clustering. Since attribute clusters contain record clusters, bad record clustering is less severe a problem and it is also easier to fix. We can just re-cluster the records within that particular attribute cluster. On the other hand bad attribute clustering is a severe problem as it requires running AutoClust from the start, which means we need to perform attribute re-clustering and, then inside each newly formed attribute cluster, we need to perform record clustering. So fixing a bad attribute clustering is more difficult and time-consuming than fixing a bad record clustering. Also a bad record clustering can possibly only affect one attribute cluster in which case the rest of the database can remain untouched while we fix that one attribute cluster.

A bad record clustering means that a query has to access data from two or more separate “data clusters” within the same attribute cluster when it could potentially need to access only one. A bad attribute clustering, on the other hand, means that a query has to access data from two or more separate attribute clusters and there is a good chance that within each attribute cluster, it might also access several “data clusters.” So the problem is compounded with bad attribute clustering.

In summary bad attribute clustering has a more dramatic effect on system performance degradation and is also much harder to fix. We can conclude that since record clustering can be fixed by itself and can be fixed easily, bad record clustering should be the first thing we test when we test the system performance in the automated system. Also note the following points: 1) good record clustering implies good attribute clustering; 2) good attribute clustering does not implies good record clustering; 3) bad record clustering does not imply bad attribute clustering; and 4) bad attribute clustering is likely to be compounded with bad record clustering. So, one possible procedure for detecting bad clustering is the following: If bad record clustering (BRC) and if bad attribute clustering (BAC) occur, we will perform attribute clustering and then perform record clustering on individual resulting attribute cluster; otherwise, if we have only bad record clustering, we will perform record clustering on the faulty clusters only. *In our research, we will develop solutions to detect BAC and BRC, when/how often they should be tested, what level of BAC/BRC is necessary in order to trigger a re-clustering, how to switch the clustering configuration from the old one to the new one and how to deal with query execution in the process. We will then also extend AutoClust for cluster computing.*

REFERENCES

[1] Abawajy, J. H., *Placement of File Replicas in Data Grid Environments*, Workshop on Programming Grids and Metasystems, Lecture Notes in Computer Science, Volume 3038, 2004.

[2] Sanjay Agrawal, V. Narasayya, B. Yang, *Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design*, SIGMOD, June 2004.

[3] Sanjay Agrawal, Surajit Chaudhuri, Lubor Kollar, Arun Marathe, Vivek Narasayya, Manoj Syamala, *Database Tuning Advisor for Microsoft SQL Server 2005: Demo*, SIGMOD 2005, June 2005.

[4] Mark Baker, Rajkumar Buyya, *Cluster Computing at a Glance, Chapter 1*, High-Performance Cluster Computing, Architectures and Systems, Vol. 1, Prentice-Hall, 1999.

[5] http://mathforum.org/advanced/robertd/bell.html. Accessed 8/15/2008.

[6] Surajit Chaudhuri and Vivek Narasayya, *Self-Tuning Database Systems: A Decade of Progress*, International Conference Very Large Databases, September 2007.

[7] 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.

[8] Margaret H. Dunham, *Data Mining: Introduction and Advanced Topics*, Prentice Hall, 2003.

[9] 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.

[10] Eadon, G., Chong E., Shankar, S., Raghavan, A., Srinivasan, J., and Das, S., *Supporting table partitioning by reference in Oracle*, ACM SIGMOD international conference on Management of data, 2008.

[11] Stéphane Gançarski, Hubert Naacke, Esther Pacitti and Patrick Valduriez, *Parallel Processing with Autonomous Databases in a Cluster System*, On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE, Lecture Notes in Computer Science, Volume 2519, 2002.

[12] Erez Hartux, and Ron Shamir, *A Clustering Algorithm Based on Graph Connectivity*, Information Processing Letters, Vol. 76, No. 4-6, 2000.

[13] Milena Ivanova, Martin Kernsten, and Niels Nes, *Self-organizing strategies for a column-store database*, ACM International conference on Extending DataBase Technology, 2008.

[14] McCormick, W. T. Schweitzer P. J., and White T. W., *Problem decomposition and data reorganization by a clustering technique*, Operation Research, Vol. 20, No.5, September 1972.

[15] Shamkant Navathe, S. Ceri, G. Widerhold, and J. Dou, *Vertical Partitioning Algorithms for Database Design*, ACM Transactions on Database Systems, Vol. 9, No. 4, December 1984.

[16] Stratos Papadomanolakis and Anastasia Ailamaki, AutoPart: Automating Schema Design for Large Scientific Databases Using Data Partitioning, International Conference on Scientific and Statistical Database Management, June 2004.

[17] Stratos Papadomanolakis, Debabrata Dash, Anastasia Ailamaki, *Efficient Use of the Query Optimizer for Automated Physical Design*, International Conference Very Large Databases, September 2007.

[18] 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.

[19] International Workshop on Self-Adaptive and Autonomic Computing Systems, DEXA 2006.

[20] 3rd International Workshop on Self-Managing Database Systems, ICDE 2008.

[21] http://www.tpc.org.

[22] Mohammed J. Zaki and C. Hsiao, *CHARM: An Efficient Algorithm for Closed Itemset Mining*, SIAM International Conference on Data Mining, April 2002.

[23] Yulai Yuan, Yongwei Wu, Guangwen Yang, and Feng Yu, *Dynamic Data Replication based on Local Optimization Principle in Data Grid*, International Conference on Grid and Cooperative Computing, 2007.