Autonomous Database Partitioning using Data Mining on Single Computers and Cluster Computers

Liangzhe Li
School of Computer Science
University of Oklahoma
Norman, OK 73019, USA
lzli@ou.edu

Le Gruenwald
School of Computer Science
University of Oklahoma
Norman, OK 73019, USA
ggruenwald@ou.edu

Abstract
One of the most important metrics in measuring the performance of a database system is query response time, which is composed of I/O time and CPU time. I/O time is decided by the amount of data read/write from/to disks and how the data is located on disks. CPU time is decided by how the database system performs the query operations. So if we want to reduce the query response time we can reduce either I/O time or CPU time, or both of them. We know retrieving data from disks is much slower than retrieving data from main memory. Hence, one of the common ways to reduce I/O times is clustering data on disks so that queries will access only relevant data. This paper introduces an efficient algorithm, called AutoClust, for automatic database attribute clustering (or also called automatic database vertical partitioning) for single computers as well as cluster computers. It is based on closed item sets mined from queries and their attributes using association rule mining. The paper then presents experimental results comparing the performance of AutoClust with that of a baseline algorithm on both single computers and cluster computers using the TPC-H benchmark running on major commercial database systems. The experiments show that AutoClust has better query costs for both types of computers.

Keywords
Vertical Partitioning, Attribute Clustering, Cluster Computer, Query Optimizer

1. Introduction
In a digital age, many business or scientific decisions cannot be made without an analysis on large amounts of data. Therefore databases are widely used in many application areas. A database application's performance highly depends on how quickly data could be retrieved from the database. When the database size is small, the time spent on reading/writing data from/to disk and operating (selecting, merging, filtering, etc.) data is usually small, and thus its impacts on query response time may not be very critical. However, today with database size getting bigger and bigger like those in bio-science, finance and medicine, if the database is not organized properly, such time overhead could yield unacceptable query response time.

Vertical partitioning and horizontal partitioning are two major techniques which can considerably improve query response time when physical database design is performed [2]. Today, most database systems support horizontal partitioning [24]. Three common horizontal partitioning approaches that are used by most database developers are range partitioning, list partitioning and hash partitioning [2]; but it is rare to find a database system that has a sophisticated algorithm to support vertical partitioning.

As query response time is composed of I/O time and CPU time, reduction on either of them can lead to an improvement of the database application's performance. Without partitioning databases, when the database system processes a query that accesses some attributes in a relation, the whole relation, rather than just those attributes, will be read from secondary memory. If we reorganize database tables in such a way that each table is partitioned vertically into sub-tables and the database system, when executing the query, will access only the relevant sub-table that contains the attributes in the query, then fewer pages from secondary memory will be accessed to process the query [17], which reduces I/O time, and thus can lead to a better query response time.

Cluster computers[5] can be used to deploy database to improve performance [23]. As query sets may change, optimal physical database design and database self tuning attract more and more attention [9][6][12]. Vertical partitioning (or also called attribute clustering) on tables is one way that can help designers achieve the performance goal. A clever algorithm is needed to automatically provide different partitioning solutions which can be implemented on different nodes of the cluster computers so that queries can always be processed on the best nodes.

In this paper we introduce a technique called AutoClust which combines data mining and parallelism together to automatically perform attribute clustering in order to reduce the cost of I/Os on cluster computers when processing queries. A preliminary version of AutoClust without any performance studies was presented in [10].The algorithm uses query frequencies and attributes accessed by queries to generate attribute clusters automatically based on closed item sets [22] mined from the attributes found in the queries processed by the database system. The best attribute clustering solution is the one that has the best cost estimated by the query optimizer. The most important part of this algorithm is that it can be implemented for distributed databases located on a cluster computer where two or more computers are linked together by a network and work together as a single integrated system. AutoClust can generate more than one solution for a table based on a query set. A table may have

IDEAS12 2012, August 8-10, Prague [Czech Republic]
Editors: Bipin C. Desai, JaroslavPokorny, Jorge Bernardino

different optimal attribute clustering solutions according to different queries and those different optimal solutions can be implemented on different nodes of a cluster computer so that a query can always be processed on the node with the best solution.

The remainder of this paper is organized as follows. In Section 2, we review the related work on database attributes clustering. In Section 3 we introduce the AutoClust algorithm for a single computer and then extend it to a cluster computer. In Section 4, we present the experimental comparison results between AutoClust and a baseline algorithm for both single computers and cluster computers using the TPC-H benchmark. Finally we give our conclusions and future work in Section 5.

2. Related Work

As discussed in [10], how to minimize disk I/Os is an important topic since the early 1970s. From that time on, algorithms have been developed to reduce I/Os by clustering huge complex data arrays in order to reduce pages reading from secondary memory [8]. Those algorithms can be categorized into three major categories: attributes affinity based, transaction based, and graph theory based vertical partitioning. In attribute affinity based vertical partitioning, a two dimensional matrix, which contains the information of what attribute appears in what query, is built according to a query set and the partitioning algorithm is based on this matrix. In transaction based vertical partitioning, transactions are considered to have more semantic meanings than attributes and a binary partitioning method is used based on a selected set of important transactions. In graph theory based vertical partitioning, a complete graph is used to replace the attributes affinity matrix. Each meaningful partition fragment is generated when the algorithm detects a certain cycle in the graph. Some algorithms combine two different groups together.

The first well-known attribute clustering algorithm was introduced in 1972 with the name of Bond Energy Algorithm (BEA) [16]. This algorithm is an attributes affinity based algorithm. It uses a two dimension array to represent the relationship between two different kinds of variables, row variable and column variable. Each column represents one kind of variable and each row represents the other kind of variable. Each element in the array is represented by a numerical value, which usually is an integer, to show the relationship between the row and column variables corresponding to this element. This algorithm permutates rows and columns of the array in order to group elements with similar values together. At the end of the algorithm, elements with similar values are located in the same block in the array and each block can be considered as a cluster. When doing permutation on the array, the algorithm needs user's subjective judgment to tell the similarity of elements; so this algorithm is hard to implement without human's interpretation. Sometimes blocks may have overlaps and some elements do not belong to any block. It means the clustering result is not always as good as what people expect.

Later after the development of BEA, another new important algorithm emerged, which was called Navathe's Vertical Partitioning (NVP) [17]. NVP is also an affinity matrix based algorithm. This was the first time that a clustering algorithm considers the frequency of queries and reflects the frequency in the attributes affinity matrix on which clustering was performed. NVP is an extension and improvement of BEA. This algorithm repeatedly does binary vertical partitioning (BVP) on a larger fragment, which is gotten from the previous BVP, to form two fragments: one larger fragment and one smaller fragment until no more fragment can be partitioned. An evaluation function, which replaces the human's subjective judgment, is used for automatic fragments selection. NVP even considers the situation when it is implemented on a distributed database on multiple sites. However $O(2^n)$ time complexity where n is the number of times binary partitioning (which is proportional to the number of queries) is repeated makes this algorithm expensive. If fragment overlapping is allowed, the time complexity will be even bigger than that. So this algorithm is only suitable for those tables with a small number of attributes.

Because of the poor time complexity of NVP, a new transaction based vertical partitioning technique, called Optimal Binary Partitioning algorithm (OBP), was proposed [26]. This algorithm constructs a binary search tree using the branch and bound method [11]. Each node of the tree represents a transaction. The left branch of a node represents those attributes being queried by the transaction that are included in a reasonable cut (if a binary cut that partitions the attributes into two sets in which at least one of them is a contained fragment which is the union of a set of attributes that the transaction accesses then this binary cut is a reasonable cut). The right branch of a node represents the remaining attributes. If all attributes of an unassigned transaction are contained in the fragment of the current node, then this transaction needs not be considered as the child of the current node. This algorithm focuses on a set of important transactions rather than attributes themselves. It does reduce time considerably compared to NVP but the run time still grows exponentially when the number of transactions grows. So it is not an ideal technique for heavy transaction systems.

Some algorithms use a graph search technique when doing data clustering. [18] is one of the examples. It is a graph theory based clustering technique. The attributes that are usually queried together are used to form a similarity graph. Vertices of the graph are elements and edges connect elements that have similarity values higher than a predefined threshold. Clusters are the sub-graphs with edge connectivity containing more than half of the number of vertices. When this technique is implemented for database vertical partitioning, a vertex represents an attribute and an edge represents how often the two attributes connected by this edge will appear together in the same transaction. Then the algorithm will traverse the graph and divide the graph into several sub graphs, each of which represents a cluster. This technique considers frequent transactions and infrequent transactions to be the same and this will lead to an inefficient partitioning result. This is because attributes that are usually accessed together in infrequent transactions but are not accessed together in frequent transactions may be put in the same fragment if all transactions are considered to be the same. Along with the increase of processor's speed and the sophistication of software, database systems become cleverer and more powerful than ever before. Researchers then realized that a database system itself can give a lot of help on physical database design to developers. A new idea of using query optimizer of a database system for automated physical design was proposed in [21]. The author introduced a cost estimation technique, which uses the query optimizer of a database system for physical database design.

In [1], a vertical partitioning algorithm that uses the idea of performing clustering based on an attributes affinity matrix from [17] was proposed. This algorithm starts with a vertex V that satisfies the least degree of reflexivity and then finds a vertex

with the max degree of symmetry among V's neighbors. Once such a neighbor is found, both V and its neighbor are put in a subset. The neighbor would become the new V. The process would continue to search neighbors of the most recent V recursively until a cycle is formed or no vertex is left. After that, the fragments will be refined using a hit ratio function. The disadvantage of this technique is similar to the disadvantage in [18]. Infrequent queries are treated the same as frequent queries.

A dynamic vertical partitioning of distributed systems, called DYVEP, was proposed in [24]. DYVEP monitors queries in order to accumulate relevant statistics for the vertical partitioning process. It analyzes the statistics in order to determine if a new partitioning is necessary; if yes, it triggers a vertical partitioning technique (VPT) to generate a new partitioning solution. The VPT could be any existing VPT that can make use of the available statistics. The algorithm then check to see if the new partitioning solution is better that the one in place; if yes, then the system reorganizes the database according to the new partitioning solution. This algorithm depends heavily on the existing VPT used and the set of rules that it develops to decide when to trigger the VPT. The algorithm does not address how it would take advantage of distributed databases that have partial or full replication so that queries can be directed to nodes that yield the best costs to execute them.

Though there are many database vertical partitioning algorithms, the efficient execution of ad-hoc heavy-weight On-Line Analytical Processing (OLAP) queries is still an open problem. Today cluster computers are widely used for solving such a problem. That is why database clustering has gained much interest for various database applications [4]. In the meanwhile, some partitioning algorithms have been developed for distributed databases on cluster computers (e.g. [7] and [1]). Some of them are table level algorithms (e.g. [1]) and some of them are schema level algorithms (e.g. [7]).

The key idea of schema level partitioning is that for a large number of database schemas and applications, transactions only access a small number of related rows which can be potentially spread across a number of tables. A recent typical schema level algorithm is ElasTraS[7], which takes the root table of a tree structure schema as the primary partitioning table and other nodes of the tree as the secondary partitioning table. The primary partitioning relation is partitioned independent of the other tables using its primary key. Because the primary table's key is part of the keys of all the secondary tables, the secondary partitioning tables are partitioned based on the primary table's partition key. Then all partitions will be spread across several Owning Transaction Managers, which own one or more partitions and provide transactional guarantees on them. Analyzing a schema is much more difficult than analyzing a table and this algorithm is generally configured for static partitioning purposes.

Table level distributed database partitioning on cluster computers is easier than schema level partitioning since all single node partitioning algorithms, like those we discussed earlier ([16][17][26][18][1][24]), can be deployed on multiple nodes; however, their disadvantages still exist on each node. Researchers have recognized that attributes vertical partitioning for distributed database on cluster computers needs an algorithm which can generate multiple solutions at a time so that different solutions can be deployed on different nodes for appropriate queries. This means that for a particular query it can always be directed to the node with the best solution for processing. In [10], the preliminary work for such an algorithm was described. In this paper, we provide details on the algorithm and its performance results.

A lot of evaluation work has been done on evaluating the performance of distributed databases on cluster computers. The results show that distributed databases can greatly improve the performance and satisfy business requirements [23]. Because of this, distributed databases have become widely used and important for many applications, which call for more research to find ways to improve their physical database design. Some researchers have proposed schema level tuning [7], table level horizontal partitioning [15], and table level vertical partitioning [17][26][18][1][24]. However, as we have discussed in the previous paragraphs, schema level partitioning is generally for static purposes and all table level partitioning algorithms we have reviewed have different weaknesses. Researchers are still looking for a better vertical partitioning algorithm which can be used together with a horizontal partitioning algorithm. In the next sections we present AutoClust, an automatic attribute clustering algorithm for both single computers and cluster computers, and performance evaluation comparing it with a baseline algorithm. A preliminary version of the algorithm without performance evaluation was presented earlier in [10]. In this paper, we further detail the extension version for cluster computers and provide performance studies for both single computers and cluster computers.

3. AutoClust

3.1. AutoClust on a Single Computer

AutoClust for a single computer was first introduced in [10]. However, In that paper, as AutoClust was still in its early stage of research, no performance studies were conducted. In this section, we summarize the key ideas of AutoClust and present experimental studies comparing the performance of AutoClust with a baseline case where the database table is in its original form without any partitioning. The AutoClust algorithm can be divided into five steps. In Step 1, an attributes usage matrix is built based on a query set showing which attributes are accessed by which queries. In Step 2, the closed item sets (CIS) [22] of attributes are mined in order to identify which attributes are accessed frequently by the same query. An item set is called closed if it has no superset having the same support which is the fraction of transactions in a data set where the item set appears as a subset [22]. For such attribute sets we need to keep them together as an independent cluster as much as possible. In Step 3, augmentations to add the primary key of the original table to each existing closed item set are done to form the augment closed item set (ACIS) which is a combination of CIS and the primary key. Then we will remove duplicate ACIS. In Step 4 an execution tree is generated where each leaf represents a candidate attribute clustering solution. Finally, in Step 5, the solutions are submitted to its query optimizer of the database system that will process the queries for cost estimation and the solution with the best cost estimation is chosen as the final solution. The cost we mentioned here is based on a special unit of workload used by the query optimizer to estimate how much work needs to be done in order to process the query. This query cost includes I/O cost and operator (or CPU) cost. We use "cost" to represent the total of the two types of costs in our examples and experimental results unless we specify the individual type of cost specifically.

Figure 1. Execution tree without partitioning

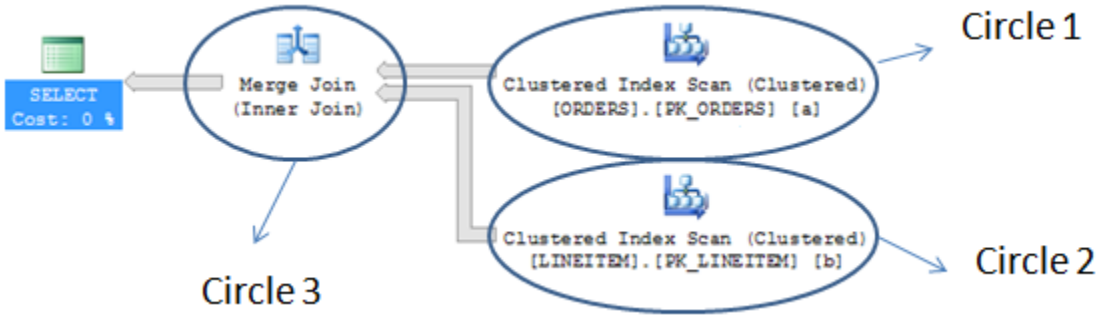
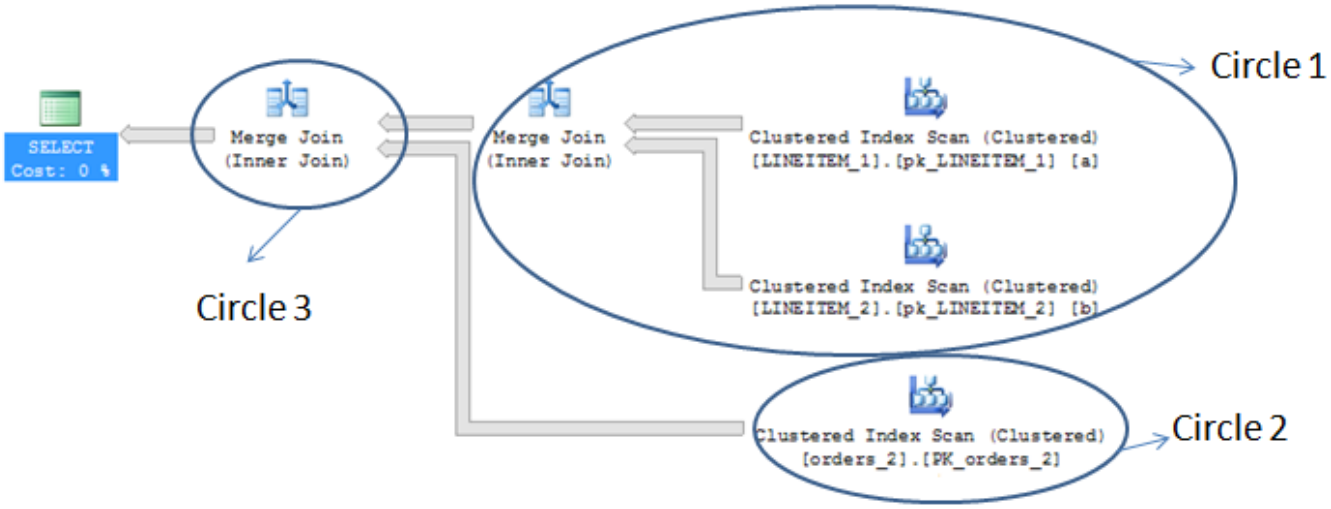


Figure 2. Execution tree with sub-trees (branches) partitioning



The execution plan generated by the query optimizer for a query is usually a tree structure. In a cost based optimization (CBO) system, one or more mechanism for returning data will be given for a query. This mechanism is represented either in a graphic mode or a textual mode. The system will choose the mechanism with the least cost as the final plan used to execute the query [19] [3]. Data access for each table can be regarded as a portion of the whole execution tree. We can partition each table to reduce the cost for each sub-tree so that we reduce the total cost for the entire execution tree. For instance, an execution tree in SQL Server (Figure 1) can be changed to a larger execution tree (Figure 2) when we do partition on its branches. Once we compare the cost table 1 and cost table 2 we can see that the total cost decreases.

Below we provide the key ideas of the five steps of AutoClust for a single computer. More details can be found in [10].

Table 1. Estimate cost of Figure 1

Circle No.	Operation	Est. Cost
Circle 1	Index scan on ORDERS table	18.04
Circle 2	Index scan on LINEITEM table	83.86
Circle 3	Merge join	16.23

Table 2. Estimate cost of Figure 2

Circle No.	Operation	Est. Cost
Circle 1	Get data from LINETEM_1 and LINEITEM_2 tables and then join them together	75.8
Circle 2	Get data from ORDERS_2 table	4.95
Circle 3	Merge join	16.23

Step 1: Build the Frequency-Weighted Attribute Usage Matrix

A frequency-weighted usage matrix is built based on attribute affinity matrix [17]. Each row of the frequency-weighted attribute usage matrix represents what attributes are accessed by the corresponding query and what is the percentage the query takes in the whole query sets.

Step 2: Mining the Closed Item sets

In order to reduce I/O's when a query accesses data, attributes in a database table that are queried together by the same query

should be put in one cluster. In other words, we need to find the attributes set which is the maximal attributes set contained in the same queries. Such attributes set is a closed item set [22].

Step 3: Filtering the Closed Item Sets
Once we have mined all the closed item sets, we augment them by adding the primary key of the database table to each set and remove the duplicate closed item sets. Then we get the augmented closed item sets (ACIS).

Step 4: Generating all possible attribute clustering solutions based on Augmented Closed Item Sets.
In this step an execution tree is constructed. The root of the tree is the item set that contains unused attributes (i.e. those that are not in any ACIS) and the primary key of the database table. The tree is extended by adding possible ACIS until all attributes are included in one leaf. Each leaf of the tree represents a valid candidate attribute clustering solution.

Step 5: Determining the best attribute clustering solution
Each solution generated in Step 4 is implemented on a database system and the aggregate query cost is estimated using the database internal query optimizer. The solution with the least aggregate cost is chosen as the best attribute clustering solution.

3.2. AutoClust on Cluster Computers with Full Replication on Every Computer Node

Computers linked together through a high speed network can work together as a single integrated system in order to improve performance. This system is called a cluster computing system [5]. When a database application is deployed on a cluster computing system, AutoClust can be used to perform vertical partitioning on the database and it can greatly decrease the query response time for the whole system. In [10], the preliminary extension of AutoClust without any performance evaluation was presented for two cases: 1) when the database is replicated on every computer node and 2) when the database is not replicated. Here we provide the detailed extension and performance evaluation for the first case. AutoClust can generate multiple attribute clustering solutions at a time and a different solution may have a different query cost for the same query. We can implement different solutions on different nodes so that a query can always be executed on the node with the best solution for the query. If this node happens to be busy then the query can be executed on the node with the second best solution, and so on. This can greatly benefit the system performance.

Formally, to do vertical partitioning or attribute clustering for a database table on cluster computers that have full data replication, AutoClust performs the steps described below. In order to make these steps more understandable we will use an example to explain each step. The database table and queries of the example come from the TPC-H benchmark [25]. The database table is SUPPLIER as shown in Table 3 and the ten queries out of the twenty-two TPC-H queries as shown in Table 4 access this database table. The queries' frequencies are generated randomly.

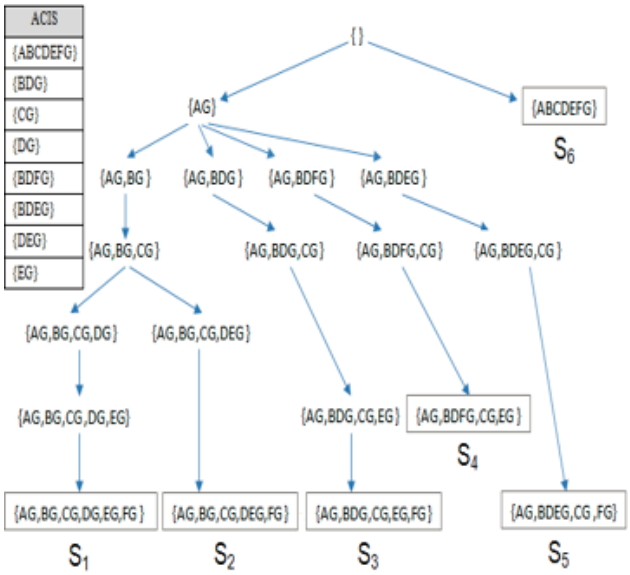
Table 3. Attributes information of the database table SUPPLIER

Attribute Name	Database Table	Data Type
S_ACCTBAL (A)	Supplier	Decimal
S_ADDRESS (B)	Supplier	Varchar
S_COMMENT (C)	Supplier	Varchar
S_NAME (D)	Supplier	Char
S_NATIONKEY (E)	Supplier	Int
S_PHONE(F)	Supplier	Char
S_SUPPKEY(G)	Supplier	Int

Table 4. Attribute usage matrix

Queries	Attributes							Frequency (%)
	A	B	C	D	E	F	G	
q_1	1	1	1	1	1	1	1	14.09
q_2	0	0	0	0	1	0	1	16.96
q_3	0	0	0	0	1	0	1	4.01
q_4	0	0	0	0	1	0	1	1.06
q_5	0	0	0	0	1	0	1	0.12
q_6	0	0	0	0	1	0	1	14.26
q_7	0	1	0	1	0	1	1	1.06
q_8	0	0	1	0	0	0	1	19.26
q_9	0	1	0	1	1	0	1	14.46
q_{10}	0	0	0	1	1	0	0	14.72

Figure 3. Execution tree generated by AutoClust for a single computer on the SUPPLIER table



Step 1: run Steps 1 – 4 of the AutoClust algorithm for a single computer. This step will produce the possible attribute clustering solutions for one computer node. The AutoClust algorithm for a single computer will generate the execution tree to get the candidate attribute clustering solutions S_i's where each S_i is a set of attributes. The execution tree is shown in Figure 3. Once the execution tree is built successfully the cost is produced as shown in Table 5.

Table 5. Estimate costs for candidate attribute clustering solutions

Solution	Estimate Query Cost Produced by Query Optimizer										Aggregate cost
	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}	
S_1	98	7	7	7	7	7	49	32	44	26	33
S_2	90	18	18	18	18	18	49	32	36	18	34
S_3	89	7	7	7	7	7	40	32	35	35	32
S_4	82	7	7	7	7	7	33	32	41	41	33
S_5	82	28	28	28	28	28	41	32	28	28	37
S_6	68	68	68	68	68	68	68	68	68	68	68

Table 6. Estimate costs for possible candidate attribute clustering solutions after ranking

Solution	Estimate Query Cost Produced by the Query Optimizer										Aggregate cost
	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}	
S_3	89	7	7	7	7	7	40	32	35	35	32
S_4	82	7	7	7	7	7	33	32	41	41	33
S_1	98	7	7	7	7	7	49	32	44	26	33
S_2	90	18	18	18	18	18	49	32	36	18	34
S_5	82	28	28	28	28	28	41	32	28	28	37
S_6	68	68	68	68	68	68	68	68	68	68	68

Step 2: rank the solutions in increasing order based on their aggregate costs as the solutions will be chosen later in Step 4 based on their increasing order of aggregate costs for implementation on computer nodes (i.e. the best solution will be chosen first). Applying Step 2 on Table 5, we get Table 6.

Step 3: remove the solutions the costs of which are larger than the cost of NoPartition. When we do attribute clustering on a database we can decrease the I/O cost, but the trade-off is that we might increase the chance of join operations, which leads to an increase in the operator cost. So, this removal is necessary to avoid the case of over-partitioning. For the SUPPLIER table in our example, the estimate query cost is 68 for NoPartition, and as we can see in Table 6, no solution has an aggregate cost larger than 68, so we do not need to remove any solution in this step.

Step 4: from the solutions resulted from Step 3, implement them on the computer nodes according to the increasing order of the solutions' aggregate costs so that the best solution is implemented on the first node, the second best solution on the second node, and so on. If there are still nodes remaining after all the solutions have been implemented, then repeat the loop again, i.e., implementing the best solution on the first remaining node, the second best solution on the second remaining node, and so on. As an example, if our cluster computers have three nodes, node 1, node 2 and node 3, then we will implement the first three solutions highlighted in Table 6 on these three nodes, i.e., S_3 on node 1, S_4 on node 2, and S_1 on node 3.

Step 5: from the results generated in Step 1, construct an attribute clustering solution choice table as shown in Table 7 that shows which solution is the best, second best, and so on for each query. Then from this table and from the solution implementation in Step 4, construct a query routing table as shown in Table 8 that shows which implemented solution (and on which node it is implemented) is the best, second best, and so on for each query so that when a query arrives, it will be directed to the node where its best solution is implemented. If this node is busy, then the query will be routed to its second best node, and so on. For example, when query q_1 arrives, it will be directed to the first node of its first choice, which is node S_4 in Table 8. If this node is busy, then q_1 will be directed to the implemented node of its second choice, which is node S_3.

Table 7. Best attribute clustering choices for each query

Choices	Queries									
	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}
1st	S_6	S_3	S_3	S_3	S_3	S_3	S_4	S_3	S_5	S_2
2nd	S_4	S_4	S_4	S_4	S_4	S_4	S_3	S_4	S_3	S_1
3rd	S_5	S_1	S_1	S_1	S_1	S_1	S_5	S_1	S_2	S_5
4th	S_3	S_2	S_2	S_2	S_2	S_2	S_1	S_2	S_4	S_3
5th	S_2	S_5	S_5	S_5	S_5	S_5	S_2	S_5	S_1	S_4
6th	S_1	S_6	S_6	S_6	S_6	S_6	S_6	S_6	S_6	S_6

The AutoClust algorithm on cluster computers is shown in Figure 4.

Table 8. Query routing table for a 3 nodes cluster computer

Choices	Queries									
	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}
1st	S_4	S_3	S_3	S_3	S_3	S_3	S_4	S_3	S_3	S_1
2nd	S_3	S_4	S_4	S_4	S_4	S_4	S_3	S_4	S_4	S_3
3rd	S_1	S_1	S_1	S_1	S_1	S_1	S_1	S_1	S_1	S_4

Figure 4. The AutoClust algorithm on cluster computers

```
Run Steps 1-4 of AutoClust for one node to generate the set S of all possible clustering solutions;
Sort solutions in S into increasing order of their aggregate query costs;
For each solution Si in S
  If cost of Si> cost of NoPartition
    Then remove Si from S
  End If
End For;
//Implement solutions on nodes
Index of solution to use  i = 1 //start with the first solution in the solution set S;
While number of remaining nodes n > 0
  If Index of solution to use i = 0 //last solution is reached
    Then index of solution to use i =1 //reset i to start from beginning of solution set S again
  End If;
  Implemented Solution IS = solution Si in S;
  Implement IS  on node Ci
  i = i – 1 //go to next solution in set S;
  n = n – 1 //go to next computer node;
End While;
//Construct query routing
For each query Qi
  Sort the solutions in increasing order of their average costs to perform the query Qi and store them in column  Qi  in the  Routing  RT[Qi];
  For each Si in RT[Qi]
    If  Si is not implemented
      Then delete Si from RT[Qi]
    End If
  End For
End For
```

From the procedure of how AutoClust works on cluster computers, we can see that multiple queries can run in parallel on the most efficient nodes. The node with a better estimate cost has a higher priority to be selected to process a query. So the total response time of the query set can be greatly decreased.

4. Experimental Performance Studies

We conduct experiments to compare the performance of AutoClust with that of a baseline algorithm, denoted as "NoPartition" where no database partitioning is performed for the database table, on a single computer as well as on a cluster computer using the popular TPC-H benchmark [25]. Below we describe the experiments and report their results.

4.1. Test results for AutoClust on a single computer

We evaluate AutoClust's performance on a single computer by running the TPC-H benchmark queries [25] on a desktop computer with a processor of Intel Core 2 Quad Q8400, RAM of 3 GB and hard disk of 300 GB. The database system is Oracle 11g Express Edition. The AutoClust algorithm is implemented in Java.

In our experiment we first randomly assign the query frequency to each query, and then select queries corresponding to each database table as the input of the AutoClust algorithm. We take the ORDERS table from the TPC-H benchmark as an example to show the experiment. For the ORDERS table, there are 12 TPC-H queries accessing it, which are Q3, Q4, Q5, Q7, Q8, Q9, Q10, Q12, Q13, Q18, Q21 and Q22. After we run AutoClust using those 12 queries, we got 33 candidate attribute clustering solutions. Then we calculate the aggregate cost for each candidate solution and select the one with the least aggregate cost as the best solution since less cost means less query response time. In our experiment, this is the solution with the aggregate cost of 3003 calculated using the cost of each query listed in the last row of Table 9 (1491*14.27%+1491*16.98% +1491*16.00%+7867*3.79%+1491*1.00%+1491*0.11%+1491 *16.35%+6895*3.62%+13693*2.13%+7867*6.14%+4548*13.8 8%+1491*5.72%=3003).

Comparing with the cost of NoPartition, which is 6633 as estimated by the query optimizer, we have an improvement of 55% when accessing data from the ORDERS table that is partitioned using the attribute clustering solution produced by AutoClust. We did experiments for all database tables in the TPC-H benchmark, the results of which are shown in Table 10. From the results we can see that AutoClust can significantly reduce the estimate query cost when queries accessing data from different tables. In other words, query response time will be reduced if we use AutoClust to do vertical partitioning before we process queries.

We also conduct tests on two other major commercial databases, SQL Server and DB2. We got very similar results showing that AutoClust can greatly reduce query cost.

4.2. Test results for AutoClust on cluster computers

We evaluate AutoClust performance on cluster computers by executing the TPC-H queries on the super computer OSCER [20] located at the University of Oklahoma using 2, 4, 8, 16, 32 computer nodes running the Oracle database system for 10,000 queries. We study the impacts of number of computer nodes and database table size on the performance. From Table 10 we can see that the SUPPLIER and ORDERS tables have the improvement close to the average, but the ORDERS table has more rows and attributes, so we select the ORDERS table in our test to evaluate the impact of number of computer nodes.

Table 9. Solution selected for the ORDERS table

Solution	[{O_CLERK,O_COMMENT,O_ORDERSTATUS,O_ORDERKEY},{O_CUSTKEY,O_ORDERDATE,O_SHIPPRIORITY,O_ORDERKEY}, {O_ORDERPRIORITY,O_ORDERKEY},{O_TOTOALPRICE,O_ORDERKEY}]											
Query	Q3	Q4	Q5	Q7	Q8	Q9	Q10	Q12	Q13	Q18	Q21	Q22
Frequency (%)	14.27	16.98	16.00	3.79	1.00	0.11	16.35	3.62	2.13	6.14	13.88	5.72
Cost for Each Query	1491	1491	1491	7867	1491	1491	1491	6895	13693	7867	4548	1491

Table 10. Comparison of AutoClust and NoPartition on a single computer

Table Name	Number of Rows	Number of Attributes	Row size (Bytes)	% Cost Improvement of AutoClust over NoPartition
REGION	5	3	118	0
NATION	25	4	98	0
SUPPLIER	10,000	7	145	53%
CUSTOMER	150,000	8	160	10%
PART	200,000	9	133	23%
PARTSUPP	800,000	9	144	85%
ORDERS	1,500,000	9	112	55%
LINEITEM	6,000,000	16	127	10%

A. Impacts of number of computer nodes

When we run AutoClust on the ORDERS table, we get 4 candidate attribute clustering solutions that have good improvement comparing with NoPartition, which we show in Table 11. Note that in order to simplify the description, we use letters A, B, C, etc. to represent the attributes in the ORDERS table. When the number of nodes equals to 2, the first two candidate solutions which have the best aggregate costs are implemented on the two nodes. When the number of nodes equals to 4, 8, 16 or 32, all candidate solutions are implemented on the nodes. As shown in Figure 5, when the number of nodes increases, both algorithms perform better, but on average AutoClust is 50% better than NoPartition.

B. Impact of database table size

We test all database tables in the TPC-H benchmark on 16 nodes. For the two smallest tables, Region and Nation, AutoClust and NoPartition yield the same performance. For all other tables which have bigger sizes, AutoClust always performs better than NoPartition, ranging from an improvement from 10% to 85%.

Table 11.AutoClust's candidate attribute clustering solutions for the ORDERS table

Candidate solutions	Aggregate cost
[{ABGE},{CDHE},{FE},{IE}]	3003
[{ABGE},{CDE},{FE},{HE},{IE}]	4516
[{ABGE},{CDIE},{FHE}]	4604
[{ABGE},{CDE},{FHE},{IE}]	4815

Figure 5. Impacts of number of computer nodes in a cluster computer

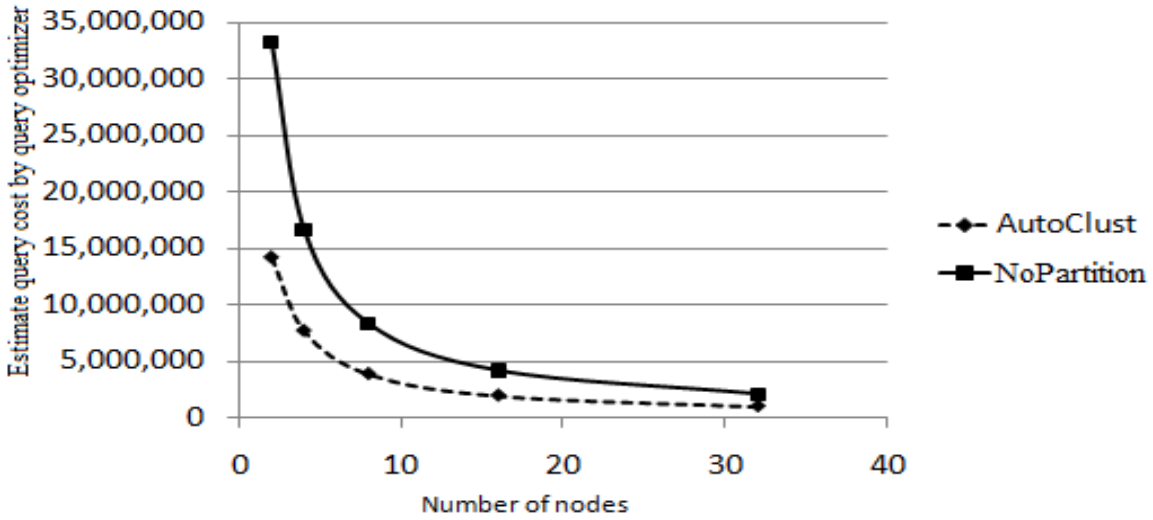
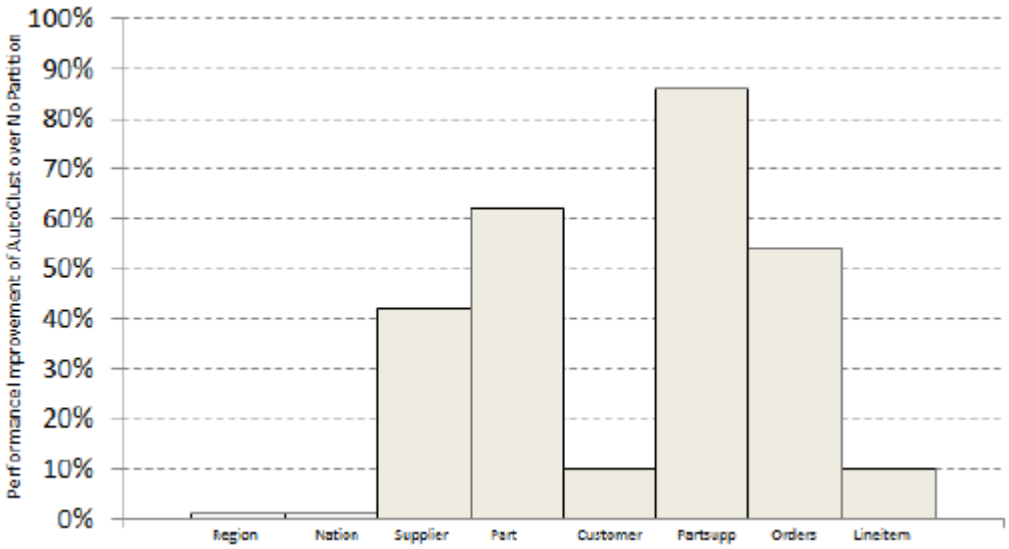


Figure 6. Impacts of database table size: AutoClust improvement over NoPartition on a 16 nodes computer for different database tables



5. Conclusions

In this paper, we presented an efficient algorithm, AutoClust, for database attribute clustering (also known as database vertical partitioning) on single computers as well as on cluster computers. This algorithm uses data mining to find closed item sets based on a set of queries performed on a database table and uses those itemsets to discover how to put the attributes of the table into clusters, i.e. how to vertically partition the table. AutoClust can generate multiple attribute clustering solutions which can then be implemented on a cluster computer to deal with different queries so that the database system can always run queries on the best node having the least cost solution. Experimental results using the TPC-H benchmark and the commercial database management system, Oracle, show that AutoClust performs much better than the NoPartition algorithm for both single computers and cluster computers.

For future work, we will conduct experiments to compare AutoClust with some other vertical partitioning algorithms, such as [1].AutoClust should be able to run automatically and periodically to test the performance of the database and decide whether or not a new database partition structure should be implemented. So we plan to incorporate into AutoClust a mechanism to automatically detect the change in the system performance and conduct appropriate database reclustering. Currently we do not consider the band cost since our test was done on a super computer with high speed LAN connection. But in real life two database servers may be located very far from each other; so in our future work, we will use the Message Passing Interface (MPI) [13] model and take into account other costs [14] such as latency and bandwidth apart from the execution time itself for each query.

References

[1]Abuelyaman, E., S., *An Optimized Scheme for Vertical Partitioning of a Distributed Database*, IJCSNS International Journal of Computer Science and Network Security, Vol.8, No.1, 2008.

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

[3]Agrawal S., Chaudhuri S., Kollar L., Marathe A., Narasayya V., Syamala M., *Database Tuning Advisor for Microsoft SQL Server 2005*: Demo, SIGMOD 2005, June 2005.

[4] Akal, F., Bohm, K., and Schek, H. –J. *OLAP Query Evaluation in a database Cluster: A performance Study on Intra-query Parallelism*. The 6th East European Conference on Advances in Database and Information Systems. London, UK, pp. 218-231, 2002.

[5]Baker, M. *"Cluster Computing at a Glance"* Chapter 1, High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice Hall, 1st edition, Editor Buyya, R., May 1999.

[6]Chaudhuri, S. and Narasayya, V., *Self-Tuning Database Systems: A Decade of Progress*, VLDB 2007, September 2007.

[7] Das, S., Agrawal, D., and Abbadi, A. E. ElasTraS: *An Elastic Transactional Data Store in the Cloud*. In USENIX HotCloud, June 2009.

[8] DeWitt, D. and Gray J., *Parallel Database Systems: The Future of High Performance Database Systems*, Communications of ACM, Volume 35 Issue 6, June 1992.

[9] 2nd International Workshop on Self-Adaptive and Autonomic Computing Systems. DEXA 2004.

[10]Guinepain, S. andGruenwald, L., *Using Cluster Computing to support Automatic and Dynamic Database Clustering*, IWAPT 2008.

[11] Horowitz, E. and Sahni, S., *Fundamentals of Computer Algorithms*, Rockville, MD: Computer Science Press, 1978.

[12] 3rd International Workshop on Self-Managing Database Systems, ICDE 2008.

[13] Message Passing Interface Forum. *MPI: A Message Passing Interface. In Proc. of Supercomputing '93*, pages 878–883. IEEE Computer Society Press, November 1993.

[14] Lei Chai, *High Performance and Scalable MPI Intra Node Communication Middleware for Multi Core Clusters*, Ohio State University, 2009

[15] Lima, A., Mattoso, M., Valduriez, P., *Adaptive Virtual Partitioning for OLAP Query Processing in A Database Cluster*, Brazilian Symposium on Databases (SBBD) 2004

[16] 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.

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

[18]Navathe, S. and Ra M., Vertical Partitioning for *Database Design: A Graph Algorithm*, ACM SIGMOD International Conference on Management of Data, 1989

[19] "*How the CBO Optimizes SQL Statements for Fast Response*" retrieved from http://docs.oracle.com/cd/B10500_01/server.920/a96533/optimops.htm#51613

[20]http://oscer.ou.edu

[21] Papadomanolakis, S., Dash, D. and Ailamaki, A., *Efficient Use of the Query Optimizer for Automated Physical Design*, VLDB 2007, Proceedings of the 33rd International Conference Very Large Databases, September 2007.

[22]Pasquier, N., Bastidem, Y., Taouil, R. and Lakhal, L. *Efficient Mining of Association Rules Using Closed Itemset Lattices*, Information Systems, Vol. 24, No. 1, 1999.

[23] Pukdesree S., Lacharoj V., Sirisang P., *Performance Evaluation of Distributed Database on PC Cluster Computers*, WCECS 2010, October , 2010.

[24]Rodriguez, L. and Li, X., *A Dynamic Vertical Partitioning Approach for Distributed Database System, Systems*, Man, and Cybernetics (SMC), IEEE International Conference 2011.

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

[26] 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.