Using Cluster Computing to Support Automatic and Dynamic Database Clustering

Sylvain Guinepain
Le Gruenwald

The University of Oklahoma
School of Computer Science
Norman, Oklahoma, U.S.A
ggruenwald@ou.edu
Plan

- Introduction & Motivations
- Research Objectives
- Literature Review
- Proposed Solution: AutoClust
  - Vertical Partitioning in AutoClust
- AutoClust and Cluster Computing
Growing popularity of data warehouses and temporal databases => amount of data stored in databases is increasing dramatically.

Size of the database increases => query response time increases.

Need for more efficient access methods and storage techniques.
Indexing, buffering, parallelism and clustering all aim at reducing the cost of disk I/Os.

Indexing, buffering, parallelism are inefficient without clustering.

Clustering can be static or dynamic, manual or automatic.
• Static clustering: objects are assigned to disk block at creation time and are never moved again.
  – Need to know how to cluster the data.
  – Need to observe the system for a while until queries patterns are discovered (system must run without clustering for a while).
  – Trends and patterns can and will change.

Static clustering is inefficient for evolving databases.
Dynamic Clustering: objects are relocated on disk if the current clustering becomes inadequate (e.g. due to a change in query patterns or database size).

Issues with dynamic clustering:
- What part of the database to re-cluster.
- How to re-cluster.
- When to re-cluster.
Introduction & Motivations (cont’d)

- Manual Clustering vs. Automatic Clustering
  - Manual clustering has to be started by DBA.
    - Need to know when to trigger the re-clustering.
    - Inconvenient.

- Automatic clustering => the process is automated. The algorithm determines what, how and when to re-cluster.
The field of automatic computing is getting a lot of attention

Several conferences on the subject
- ICAC'08 (Int. Conf. On Autonomic Computing)
- SMDB’07(Self-Managing Database Magt. Sys. Workshop)
- SAACS‘06

Microsoft AutoAdmin (Agrawal, 04; Chaudhuri, 05,07).

All major Database Vendors are working on automating database administration (e.g. MS Sequel Server, IBM’s DB2).
Research Objectives

- Develop an automatic clustering technique for relational databases that will re-cluster a database with little or no intervention from a DBA.

- Develop algorithms to partition a database vertically and horizontally based on patterns identified by mining the set of queries running against the database.

- Develop a technique that will trigger the re-clustering automatically when needed.
Relational Database Partitioning

- Database partitioning: vertical and horizontal.
- Vertical partitioning: groups the attributes of a database relation together.
- Horizontal partitioning: groups database records together.
- Using both horizontal and vertical partitioning => mixed partitioning.
### Horizontal Partitioning

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
<th>Occupation</th>
<th>Birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Houston</td>
<td>50K</td>
<td>Programmer</td>
<td>1-1-1960</td>
</tr>
<tr>
<td>Boyd</td>
<td>Dallas</td>
<td>100K</td>
<td>Professor</td>
<td>12-7-1955</td>
</tr>
<tr>
<td>Warner</td>
<td>Miami</td>
<td>80K</td>
<td>Researcher</td>
<td>1-1-1960</td>
</tr>
<tr>
<td>Black</td>
<td>Houston</td>
<td>50K</td>
<td>Teacher</td>
<td>9-1-1970</td>
</tr>
<tr>
<td>White</td>
<td>Houston</td>
<td>100K</td>
<td>Architecture</td>
<td>1-1-1965</td>
</tr>
<tr>
<td>Miller</td>
<td>NYC</td>
<td>80K</td>
<td>Enigneer</td>
<td>8-2-1973</td>
</tr>
</tbody>
</table>
## Horizontal Partitioning

<table>
<thead>
<tr>
<th>Name</th>
<th>City</th>
<th>Salary</th>
<th>Occupation</th>
<th>Birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>Houston</td>
<td>50K</td>
<td>Programmer</td>
<td>1-1-1960</td>
</tr>
<tr>
<td>Black</td>
<td>Houston</td>
<td>50K</td>
<td>Teacher</td>
<td>9-1-1970</td>
</tr>
<tr>
<td>White</td>
<td>Houston</td>
<td>100K</td>
<td>Architecture</td>
<td>1-1-1965</td>
</tr>
<tr>
<td>Boyd</td>
<td>Dallas</td>
<td>100K</td>
<td>Professor</td>
<td>12-7-1955</td>
</tr>
<tr>
<td>Warner</td>
<td>Miami</td>
<td>80K</td>
<td>Researcher</td>
<td>1-1-1960</td>
</tr>
<tr>
<td>Miller</td>
<td>NYC</td>
<td>80K</td>
<td>Engineer</td>
<td>8-2-1973</td>
</tr>
</tbody>
</table>

**Select**  * from Employee where City = ‘Houston’;
Limitations Of Horizontal Partitioning

- Not all queries require all attributes of a tuple.
- Some attributes may never be queried at all.
- Retrieving entire tuples from a horizontally partitioned database leads to poor performance.

- Solution: Vertical partitioning
Attributes of a relation are divided into groups based on affinity.

Groups consist of smaller records.

Fewer pages from secondary memory are accessed to process transactions that retrieve or update only some attributes from the relation, instead of the entire tuples => improve transaction performance.
Select Name, City From Employee;
Select Salary, Occupation, Birthday from Employee
Limitations of Vertical Partitioning

- Only attribute access frequency, not tuple frequency, is considered => data records needed to answer a frequent query could be scattered at random across multiple disk blocks.

Solution: Mixed Partitioning
(McIver)

3 modules:

- Statistics collection
  - heat (simple object references) and tension (co-references)
- Cluster analyzer
  - Evaluate the dissimilarity of two clustering (threshold)
- Reorganizer
  - Stores objects next to highly co-referenced objects.
  - Only objects accessed since last reorganization
  - Interruptible

* Uses user-defined parameter
Automatic Clustering

- **StatClust**
  - 5 modules:
    - Statistics collection
      - Heat and tension
      - Read/write access ratio at class level
      - Buffering process
    - Storage size determination
    - Bad Clustering Detection
      - Ratio of chunks accessed in memory buffer over total number of chunks accessed must be greater than threshold
      - Reliable stats (90% confidence interval)
Automatic Clustering

- New clustering determination
  - ‘m’ most reached objects of classes the statistics of which are good enough (confidence interval)
- Reorganizer
  - Stores objects next to highly co-referenced objects.
  - Only ‘m’ most reached object of each class
  - READ/WRITE ratio allows duplicates

* Uses many parameters and hints
Automatic Clustering

- **DRO** (Detection & Reclustering of Objects)
  - Collecting statistics
    - Heat
    - Page usage rate (triggers re-clustering)
  - Re-clustering can be triggered manually or automatically
  - Re-clustering objects that are frequently accessed and belong to a page that is NOT frequently accessed.

* Lots of user-defined parameters.
## Automatic Clustering

<table>
<thead>
<tr>
<th>Dynamic / Static</th>
<th>Auto</th>
<th>Param &amp; hints</th>
<th>Stats</th>
<th>Trigger</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cactis</td>
<td>S</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>ORION</td>
<td>S</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>CK</td>
<td>S</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Mclver</td>
<td>D</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>StatClust</td>
<td>D</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>DRO</td>
<td>D</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

21
Proposed Solution: AutoClust

- Automatic
- Dynamic
- Mixed Partitioning
- Using Data Mining
Process queries until response time drops below preset threshold.

Bad Response Time Observed

Increase response time threshold value

Reset response time threshold to initial value

Re-cluster the database horizontally

Test for bad horizontal clustering

Horizontal Clustering is Good

Re-cluster the database vertically

Test for bad vertical clustering

Vertical Clustering is Good

Bad Horizontal Clustering Detected

Bad Vertical Clustering Detected

23
Vertical Partitioning in AutoClust

4 steps:
1. Build a frequency-weighted attribute usage matrix;
2. Mine closed item sets;
3. Filter the closed item sets;
4. Find the best clustering of attributes.
Step 1: Build a frequency-weighted attribute usage matrix

- Step 1a: gather the list of all database attributes from the database schema.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Table</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (PK)</td>
<td>Table 1</td>
<td>Short</td>
</tr>
<tr>
<td>B</td>
<td>Table 1</td>
<td>Short</td>
</tr>
<tr>
<td>C</td>
<td>Table 1</td>
<td>Long</td>
</tr>
<tr>
<td>D</td>
<td>Table 1</td>
<td>Byte</td>
</tr>
<tr>
<td>E</td>
<td>Table 1</td>
<td>Short</td>
</tr>
<tr>
<td>F</td>
<td>Table 1</td>
<td>Text</td>
</tr>
</tbody>
</table>
Step 1: Build a frequency-weighted attribute usage matrix

- Step 1b: Build the *attribute usage matrix* using the query log

<table>
<thead>
<tr>
<th>Queries</th>
<th>Attributes</th>
<th>Query Frequency (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>Q₁</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Q₂</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Q₃</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Q₄</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Step 1: Build a frequency-weighted attribute usage matrix

- Step 1c:
  Build the *frequency-weighted attribute usage matrix*

<table>
<thead>
<tr>
<th>Queries</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>Q₁</td>
<td>10</td>
</tr>
<tr>
<td>Q₂</td>
<td>20</td>
</tr>
<tr>
<td>Q₃</td>
<td>0</td>
</tr>
<tr>
<td>Q₄</td>
<td>0</td>
</tr>
</tbody>
</table>
How to interpret the frequency-weighted attribute usage matrix

<table>
<thead>
<tr>
<th>Queries</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>Q₁</td>
<td>10</td>
</tr>
<tr>
<td>Q₂</td>
<td>20</td>
</tr>
<tr>
<td>Q₃</td>
<td>0</td>
</tr>
<tr>
<td>Q₄</td>
<td>0</td>
</tr>
</tbody>
</table>

The Interpretation of this matrix is as follows: 30% (20% + 10%) of the queries run access attribute ‘A’.

90% ( = 20% + 30% + 40%) access ‘B’.

60% ( = 20% + 40%) access ‘B’, ‘C’ and ‘E’.

So is \{A,C,D\} a pretty common group of attribute?.
Step 2: Mining Closed Item Sets

- **Frequent 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 item sets.
Step 2: Mining Frequent Item Sets

<table>
<thead>
<tr>
<th>Queries</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>Q_1</td>
<td>10</td>
</tr>
<tr>
<td>Q_2</td>
<td>20</td>
</tr>
<tr>
<td>Q_3</td>
<td>0</td>
</tr>
<tr>
<td>Q_4</td>
<td>0</td>
</tr>
</tbody>
</table>

The item set \{B, C\} is present in 60% (= 20% + 40%) of the queries.

The item set \{B, C\} has a support of 60%.

The item set \{A, C, D\} is present in 10% of the queries.

The item set \{B, C\} has a support of 60%.

If we set the support threshold at 20%, \{B, C\} would be a frequent item set but \{A, C, D\} would not be.
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.

The set of frequent closed Item sets is a subset of the set of frequent item sets => fewer frequent closed item sets than frequent item sets.
Step 2: Mining Frequent Closed Item Sets

- 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 $O \in O$ and $I \in I$, we define:
  - $f(O) = \{ i \in I \mid \text{for all } o \in O, (o, i) \in R \}$
  - $g(I) = \{ o \in O \mid \text{for all } i \in I, (o, i) \in R \}$

- $f(O)$ associates with $O$, items common to all transactions $o \in O$.
- $g(I)$ 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 $h(X) = X$. 

32
Step 2: Mining Frequent Closed Item Sets

In our example the closed item sets are:

- \( g(\{B,C\}) = \{q_2,q_4\} \)
- \( f(\{q_2,q_4\}) = \{B, C, E\} \)
- \( f(\{q_2,q_4\}) = \{B, C, E\} \)

\( \{B, C\} \) is NOT a CIS

\( \{B,C,E\} \) is a CIS.

<table>
<thead>
<tr>
<th>Qs</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>Q_1</td>
<td>10</td>
</tr>
<tr>
<td>Q_2</td>
<td>20</td>
</tr>
<tr>
<td>Q_3</td>
<td>0</td>
</tr>
<tr>
<td>Q_4</td>
<td>0</td>
</tr>
</tbody>
</table>

CIS

- \( \{A, B, C, E\} \)
- \( \{A, C\} \)
- \( \{A, C, D\} \)
- \( \{B, C, E\} \)
- \( \{B, E\} \)
- \( \{C\} \)
Attributes have to be clustered in such a way that the original tuples can be reconstructed later. We restrict the set of closed item sets (CIS) considered to:

- CIS containing attributes from the same relation.
- CIS containing attributes from several relations as long as the relationship cardinality between all the relations containing these attributes is 1 to 1.
- In addition, every CIS that does not contain the primary key of the relation will be augmented with the primary key attribute(s).
Step 3: Filter the closed item sets

In our example the closed item sets are:

<table>
<thead>
<tr>
<th>CIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A, B, C, E}</td>
</tr>
<tr>
<td>{A, C}</td>
</tr>
<tr>
<td>{A,C,D}</td>
</tr>
<tr>
<td>{B,C,E}</td>
</tr>
<tr>
<td>{B,E}</td>
</tr>
<tr>
<td>{C}</td>
</tr>
</tbody>
</table>

the augmented closed (ACIS) sets are:

<table>
<thead>
<tr>
<th>ACIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A, B, C, E}</td>
</tr>
<tr>
<td>{A, C}</td>
</tr>
<tr>
<td>{A,C,D}</td>
</tr>
<tr>
<td>{A,B,E}</td>
</tr>
</tbody>
</table>

* Note that attribute A was the primary key of Table1=(A, B, C, D, E, F)
Step 4: Finding the Best Vertical Clustering of Attributes Using CIS

- Branch and bound type algorithm that examines all clustering solutions of attributes such that a solution contains at least one cluster that is an augmented closed item set.

- The solution with the lowest cost is selected.
Step 4: Find the Best Vertical Clustering of Attributes Using ACIS

- A candidate clustering solution is a union of attribute clusters such that at least one cluster is an ACIS and all attributes are present in the solution.

- All sets in the solution are disjoint: no attribute will be duplicated except for the primary key attributes.

- Example: \{ \{A, C, D\}, \{A, B, E\}, \{A, F\} \} is a valid attribute clustering containing 3 clusters.
Step 4: Find the Best Vertical Clustering of Attributes Using CIS

- The total number of possible partitions (clusters) is equal to the Bell number that follows the following recurrence relation \( b_n + 1 = \sum_{k=0}^{n} b_k \binom{n}{k} \), where \( I \) is the set of all attributes.

\[
\begin{align*}
    b_n + 1 &= \sum_{k=0}^{n} b_k \binom{n}{k} \\
    &= b_n + 1
\end{align*}
\]

- AutoClust greatly reduces the search space by considering only solutions that contain at least one cluster in ACIS.
The algorithm starts with an empty solution and adds clusters of attributes that are ACIS to the solution until it forms a complete partition of the set of attributes.

When the solution is complete, its cost is measured by running all the queries through the query optimizer which estimates the number of I/Os => best solution has the smallest cost.

Note that the queries are not actually run.

All possible combinations of ACIS as clusters will be considered.
Algorithm for Determining Clustering Candidate Solutions

BEGIN MAIN

// Inputs: I = set of all items (attributes). ACIS = Augmented Closed Item Sets. PK = the set of all the primary keys
// Output: the vertical clustering solution with the lowest cost

P = All attributes present in any set in ACIS // Find all attributes present in any set in ACIS
NP = I – P // Find all attributes NOT present in any set in ACIS
P = P – PK // Remove all the primary keys from P since all ACIS are already augmented

// Find all candidate clustering solutions (CCS). Attributes not in any ACIS are clustered separately
FOR all attributes attr in NP DO
    CCS = CCS ∪ (PK ∪ attr)
END FOR // call recursively completeCandidateSolution
completeCandidateSolution( CCS, P ,ACIS, PK)

END MAIN
completeCandidateSolution( CCS, P, ACIS, PK )

// Role: Incrementally builds a clustering solution of the set of attributes by adding a set of ACIS
// or a 1-item set to the current incomplete solution

// Output: A partition of the set of attributes, complete or not.

att = first attribute in P

FOR each set S in (ACIS U (att U PK)) containing att DO
IF ( CCS U S ) contains all the attributes in I
    // then the solution is complete
    CCS = CCS U S
    // Run every query in Q and multiply their cost (EstimateIO) by their frequency and
    // compute the aggregate cost of the clustering.
    // If the cost is the best save it as current best solution.
THEN
ELSE
    // The solution is incomplete, we need to prune P and ACIS, then call recursively.
    remove from P all elements contained in S.
    remove from ACIS all sets containing elements in S.
    completeCandidateSolution( CCS U S, P, ACIS, PK )
END IF
END FOR

END completeCandidateSolution( CCS, P, ACIS, PK )
Execution Tree of AutoClust Vertical Partitioning Algorithm

<table>
<thead>
<tr>
<th>ACIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A, B, C, E}</td>
</tr>
<tr>
<td>{A, C}</td>
</tr>
<tr>
<td>{A, C, D}</td>
</tr>
<tr>
<td>{A, B, E}</td>
</tr>
</tbody>
</table>

{} → \{AF\}  
- Starts with an empty solution

\{AF\} → \{AF, AB\}  
- Adds attributes that are not present in any CIS (here is \{F\}) and augments with the PK(A) => \{AF\}

\{AF, AB\} → \{AF, AB, AC\}  
\{AF, AB, ACD\} → \{AF, AB, AC, AD\}  
\{AF, AB, AC, AD, AE\}  

\{AF, ABCE\} → \{AF, ABCE, AD\}  
\{AF, ABE\}  

\{AF, ABE, AC\} → \{AF, ABE, AC, AD\}  
\{AF, ABE, AC, AD\}  
\{AF, ABE, ACD\}  

5 solutions will be submitted to the Query Optimizer for cost comparison
AutoClust Identifies 5 candidate clustering solutions. Each solution is then tested and queries are sent to the query optimizer. Note that the queries are not actually run.

- $S_1 = \{\{A, F\}, \{A, B\}, \{A, C\}, \{A, D\}, \{A, E\}\}$
- $S_2 = \{\{A, F\}, \{A, B\}, \{A, C, D\}, \{A, E\}\}$
- $S_3 = \{\{A, F\}, \{A, B, C, E\}, \{A, D\}\}$
- $S_4 = \{\{A, F\}, \{A, B, E\}, \{A, C\}, \{A, D\}\}$
- $S_5 = \{\{A, F\}, \{A, B, E\}, \{A, C, D\}\}$
The aggregate cost is computed by running each query against each solution and multiplying the resulting cost by the query frequency.

The solution with the lowest aggregate cost is S3

<table>
<thead>
<tr>
<th>Solutions</th>
<th>Estimated Individual Query Cost</th>
<th>Aggregate Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
<td>Q2</td>
</tr>
<tr>
<td>S1</td>
<td>3.857</td>
<td>5.601</td>
</tr>
<tr>
<td>S2</td>
<td>2.658</td>
<td>5.779</td>
</tr>
<tr>
<td>S3</td>
<td>4.403</td>
<td>3.026</td>
</tr>
<tr>
<td>S5</td>
<td>2.657</td>
<td>4.584</td>
</tr>
</tbody>
</table>
Performance Study

- Compared AutoClust vertical partitioning technique with the Optimal Binary Partitioning algorithm proposed in literature and the global optimum solution.

- Implemented all techniques in Java 1.4.1.

- Used the TPC-R benchmark data sets and queries to carry out our tests.
Ran on a 1.2 GHz DELL computer with 576 MB of RAM.

Randomly assigned frequencies to the queries in the data set.

Ran all clustering algorithms and determined the best vertical clustering according to each one.

At each time step the query frequencies were reassigned randomly.

The set of queries remained the same.
Conducted our test on two separate data sets.

The first data set has 10 attributes spread across 5 tables and 10 queries.

The second one is the TPC-R benchmark with 61 attributes in 8 tables and 22 queries.

For all tests the support level threshold was set at 1% (=> forces AutoClust to use all frequent closed item sets => would return the best solution but would also take the longest time to run.)
Performance Metrics:
- number of disk accesses in response to the set of queries running against the database;
- running time of the technique.
First data set

- The quality of the solution produced by AutoClust was 31% better on average than OBP;
- the running times were nearly identical;
- Autoclust finds the global optimal solution every time in about 0.05 seconds while an exhaustive search of the solution space takes about 45 seconds on average.
First data set

Simulation Results

AutoClust Vs OBP (# Blocks Accessed)
(10 attributes, 10 queries)
Simulation Results

- Second data set
  - The quality of the solution produced by AutoClust was 57% better on average than OBP;
  - The average running time for AutoClust (5s) was more than 500 times faster than that of OBP (2600s);
  - An exhaustive search for the global optimum was not possible because of the enormous size of the search space.
Simulation Results

- Second data set

AutoClust Vs OBP (# Blocks Accessed)
TPC-R Benchmark data

<table>
<thead>
<tr>
<th>Iteration #</th>
<th>AutoClust</th>
<th>Chu</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10000</td>
<td>52</td>
</tr>
<tr>
<td>5</td>
<td>15000</td>
<td>25</td>
</tr>
<tr>
<td>9</td>
<td>20000</td>
<td>17</td>
</tr>
<tr>
<td>13</td>
<td>25000</td>
<td>13</td>
</tr>
<tr>
<td>17</td>
<td>30000</td>
<td>9</td>
</tr>
<tr>
<td>21</td>
<td>35000</td>
<td>5</td>
</tr>
<tr>
<td>25</td>
<td>40000</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>45000</td>
<td></td>
</tr>
</tbody>
</table>
AutoClust can be adapted to greatly benefit from cluster computing.

- Improved performance and/or availability
- 2 different architectures
  - Cluster computing with no data replication
  - Cluster computing with data replication
Imagine the following database relations:
- \( T_1 = (K_1, A, B, C, D) \)
- \( T_2 = (K_2, E, F) \)
- \( T_3 = (K_3, G, H) \)
- \( T_4 = (K_4, I, J) \)
- \( T_5 = (K_5, K, L, M) \)
- \( T_6 = (K_6, O, P, Q) \)

Let us assume further that there is a 1-to-1 relationship between
- \( K_1, K_2, \) and \( K_3 \)
- \( K_4 \) and \( K_5 \)

The data can be stored on 3 separate nodes.
Cluster Computing w/o Replication

- Queries can be run in parallel on each node. Simultaneously or asynchronously.
- Each node implements its best possible clustering.
- Performance gain is due to parallel execution of queries and re-clustering process.

Node 1: [K1, A, B, C, D, E, F, G, H]
Node 2: [K4, I, J, K, L, M]
Node 3: [K6, O, P, Q]

SELECT A, B FROM T1
SELECT O, Q FROM T6
Cluster Computing with Replication

- Also called high availability clusters
- Same data are stored on each node but using a different clustering
- Let us assume we have 3 nodes.
  - Node 1 implements clustering S3
  - Node 2 implements clustering S5
  - Node 3 implements clustering S4

Clustering solutions S3, S5, and S4 have the best costs

<table>
<thead>
<tr>
<th>Solution</th>
<th>Estimated Individual Query Cost</th>
<th>Aggregate Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
<td>Q2</td>
</tr>
<tr>
<td>S1</td>
<td>3.85</td>
<td>5.60</td>
</tr>
<tr>
<td>S2</td>
<td>2.65</td>
<td>5.77</td>
</tr>
<tr>
<td>S3</td>
<td>4.40</td>
<td>3.02</td>
</tr>
<tr>
<td>S4</td>
<td>3.85</td>
<td>4.40</td>
</tr>
<tr>
<td>S5</td>
<td>2.65</td>
<td>4.58</td>
</tr>
</tbody>
</table>
Each query can run on a different node based on expected response time.

Same data are stored on each node but using a different clustering.

Queries are directed to a node based on a routing table.

Each node has a copy of the routing table.

Data are always available if one node is being re-clustered.

Queries can run in parallel.
The routing table is obtained by inverting the clustering result table produced by AutoClust.

<table>
<thead>
<tr>
<th>Solutions</th>
<th>Estimated Individual Query Cost</th>
<th>Aggregate Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
<td>Q2</td>
</tr>
<tr>
<td>S1</td>
<td>3.85</td>
<td>5.60</td>
</tr>
<tr>
<td>S2</td>
<td>2.65</td>
<td>5.77</td>
</tr>
<tr>
<td>S3</td>
<td>4.40</td>
<td>3.02</td>
</tr>
<tr>
<td>S4</td>
<td>3.85</td>
<td>4.40</td>
</tr>
<tr>
<td>S5</td>
<td>2.65</td>
<td>4.58</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Choices</th>
<th>Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st Choice</td>
<td>S5</td>
</tr>
<tr>
<td>2nd Choice</td>
<td>S4</td>
</tr>
<tr>
<td>3rd Choice</td>
<td>S3</td>
</tr>
</tbody>
</table>
Cluster Computing with Replication

Node 1 Implements clustering S3

Node 2 Implements clustering S5

Node 3 Implements clustering S4

<table>
<thead>
<tr>
<th>Choices</th>
<th>Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
</tr>
<tr>
<td>1st Choice</td>
<td>S5</td>
</tr>
<tr>
<td>2nd Choice</td>
<td>S4</td>
</tr>
<tr>
<td>3rd Choice</td>
<td>S3</td>
</tr>
</tbody>
</table>
Thank you!
Mine Frequent Closed Item Sets

- A subset of the set of frequent item sets is the set of frequent closed item sets defined as follows:
  - 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.