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

Introduction & Motivations

- **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.**

Introduction & Motivations (cont'd)

- **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.**

Introduction & Motivations (cont'd)

- **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.

Introduction & Motivations (cont'd)

- **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.**

Automatic Computing

- 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

Name	City	Salary	Occupation	Birthday
Smith	Houston	50K	Programmer	1-1-1960
Boyd	Dallas	100K	Professor	12-7-1955
Warner	Miami	80K	Researcher	1-1-1960
Black	Houston	50K	Teacher	9-1-1970
White	Houston	100K	Architecture	1-1-1965
Miller	NYC	80K	Enigneer	8-2-1973

TABLE EMPLOYEE

Horizontal Partitioning

Name	City	Salary	Occupation	Birthday
Smith	Houston	50K	Programmer	1-1-1960
Black	Houston	50K	Teacher	9-1-1970
White	Houston	100K	Architecture	1-1-1965
Boyd	Dallas	100K	Professor	12-7-1955
Warner	Miami	80K	Researcher	1-1-1960
Miller	NYC	80K	Enigneer	8-2-1973

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**

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.**

Vertical Partitioning

Name	City	Salary	Occupation	Birthday
Smith	Houston	50K	Programmer	1-1-1960
Boyd	Dallas	100K	Professor	12-7-1955
Warner	Miami	80K	Researcher	1-1-1960
Black	Houston	50K	Teacher	9-1-1970
White	Houston	100K	Architecture	1-1-1965
Miller	NYC	80K	Enigneer	8-2-1973

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

Automatic Clustering

- **(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.**

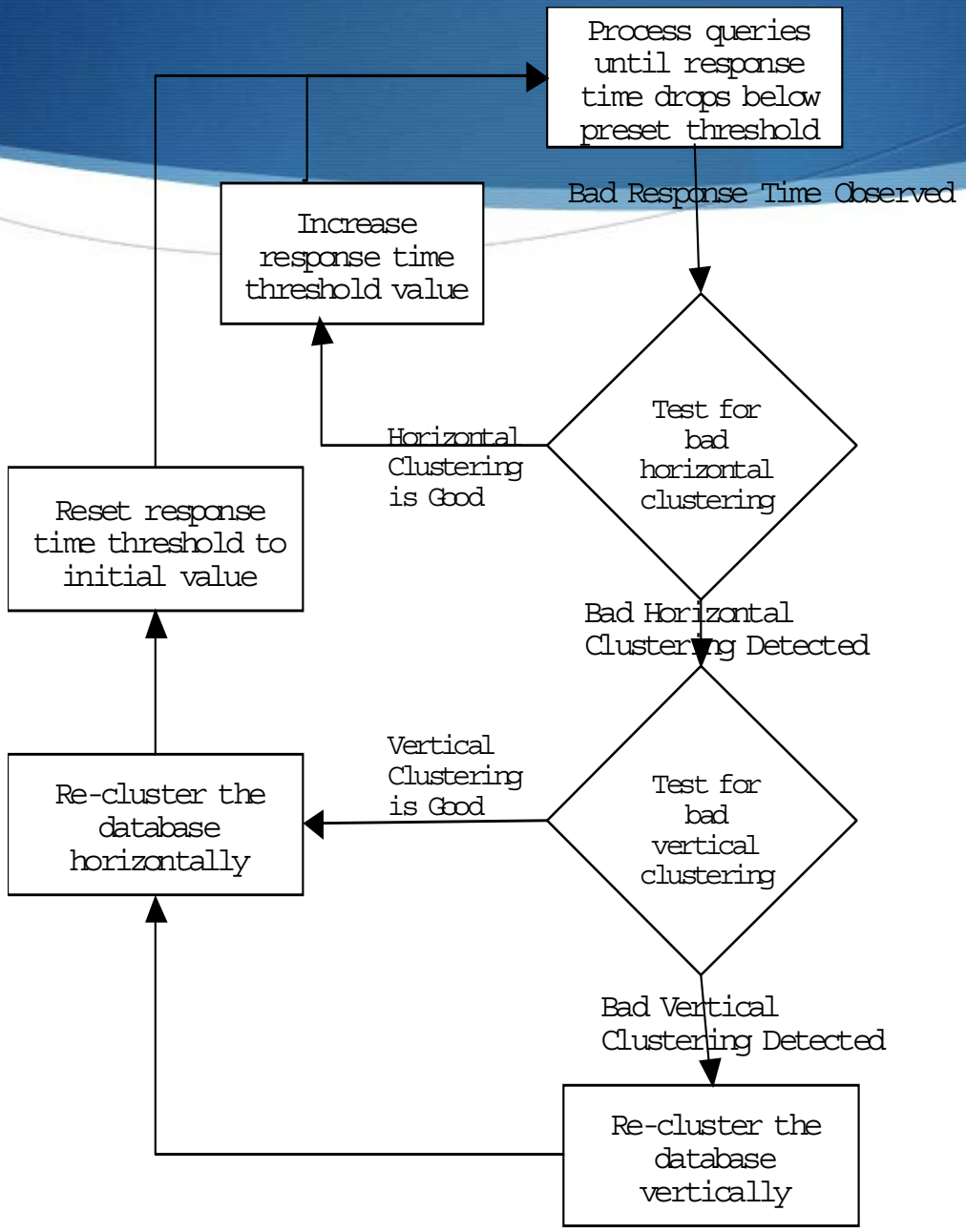
*** Lots of user-defined parameters.**

Automatic Clustering

	Dynamic / Static	Auto	Param & hints	Stats	Trigger
Cactis	S	N	N	Y	N/A
ORION	S	N	Y	N	N/A
CK	S	N	N	N	N/A
McIver	D	Y	Y	Y	Dissimilarity btw clusterings
StatClust	D	Y	Y	Y	Chunks accessed in buffer / chunks accessed
DRO	D	Y	Y	Y	Page rate usage

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
Test for bad horizontal clustering
Horizontal Clustering is Good
Reset response time threshold to initial value
Bad Horizontal Clustering Detected
Vertical Clustering is Good
Test for bad vertical clustering
Re-cluster the database horizontally
Bad Vertical Clustering Detected
Re-cluster the database vertically

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.

Attribute Name	Table	Data Type
A (PK)	Table 1	Short
B	Table 1	Short
C	Table 1	Long
D	Table 1	Byte
E	Table 1	Short
F	Table 1	Text

Step 1: Build a frequency-weighted attribute usage matrix

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

Queries	Attributes						Query Frequency (%)
	A	B	C	D	E	F	
Q_1	1	0	1	1	0	0	10
Q_2	1	1	1	0	1	0	20
Q_3	0	1	0	0	1	0	30
Q_4	0	1	1	0	1	0	40

Step 1: Build a frequency-weighted attribute usage matrix

- Step 1c:
  Build the ***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

How to interpret the 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

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

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

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.

Step 2: Mining Frequent Closed Item Sets

- **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 = ($\mathcal{O}$, $\mathcal{I}$, $\mathcal{R}$) be a data mining context, $\mathcal{O}$ a set of transactions, $\mathcal{I}$ a set of items, and $\mathcal{R}$ a binary relation between transactions and items. For O ∈ $\mathcal{O}$ and I ∈ $\mathcal{I}$, we define:
  - ✓ f(O) = { i ∈ $\mathcal{I}$ | for all o ∈ O, (o,i) ∈ $\mathcal{R}$ }
  - ✓ g(I) = { o ∈ $\mathcal{O}$ | for all i ∈ I, (o,i) ∈ $\mathcal{R}$ }
- **f(O) associates with O, items common to all transactions o ∈ O.**
- **g(I) associates with I, transactions related to all items i ∈ I.**
- **The operators $h = fog$ and $h' = gof$ are the Galois closure operators (Pasquier, 1999).**
- **Let X be an item set, X ∈ $\mathcal{I}$. X is a closed item set iff h(X) = X.**

Step 2: Mining Frequent Closed Item Sets

Qs	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

$g(\{B,C\}) = \{q_2,q_4\}$
$f(\{q_2,q_4\}) = \{B, C, E\}$ } x<> f(g(x))

{B,C} is NOT a CIS

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

{B,C,E} is a CIS.

In our example the closed item sets are:

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

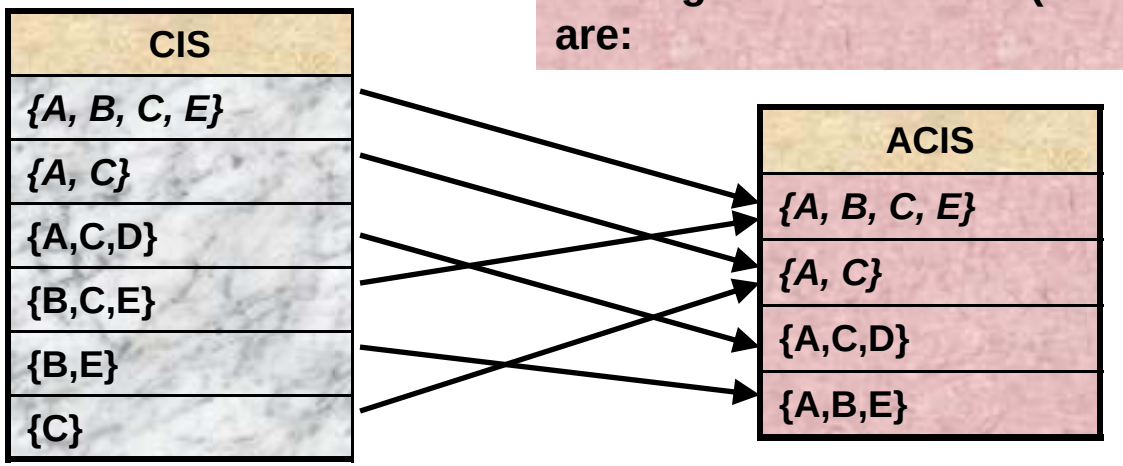
Step 3: Filtering the closed item sets

- **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:

the augmented closed (ACIS) sets are:



* 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 (n= ||I||), where I is the set of all attributes.**

$$b_{n+1} = \sum_{k=0}^{n} b_k \binom{n}{k}$$

- **AutoClust greatly reduces the search space by considering only solutions that contain at least one cluster in ACIS.**

Step 4: Find the Best Vertical Clustering of Attributes Using CIS

- **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

// {{a,b},{c}} ∪ {d} = {{a,b},{c},{d}}

// In our example NP={P}, hence {AF} is a cluster

// since PK = {A} is the sole primary key.

CCS = CCS ∪ (PK ∪ attr)

END FOR

// call recursively completeCandidateSolution

completeCandidateSolution(CCS, P ,ACIS, PK)

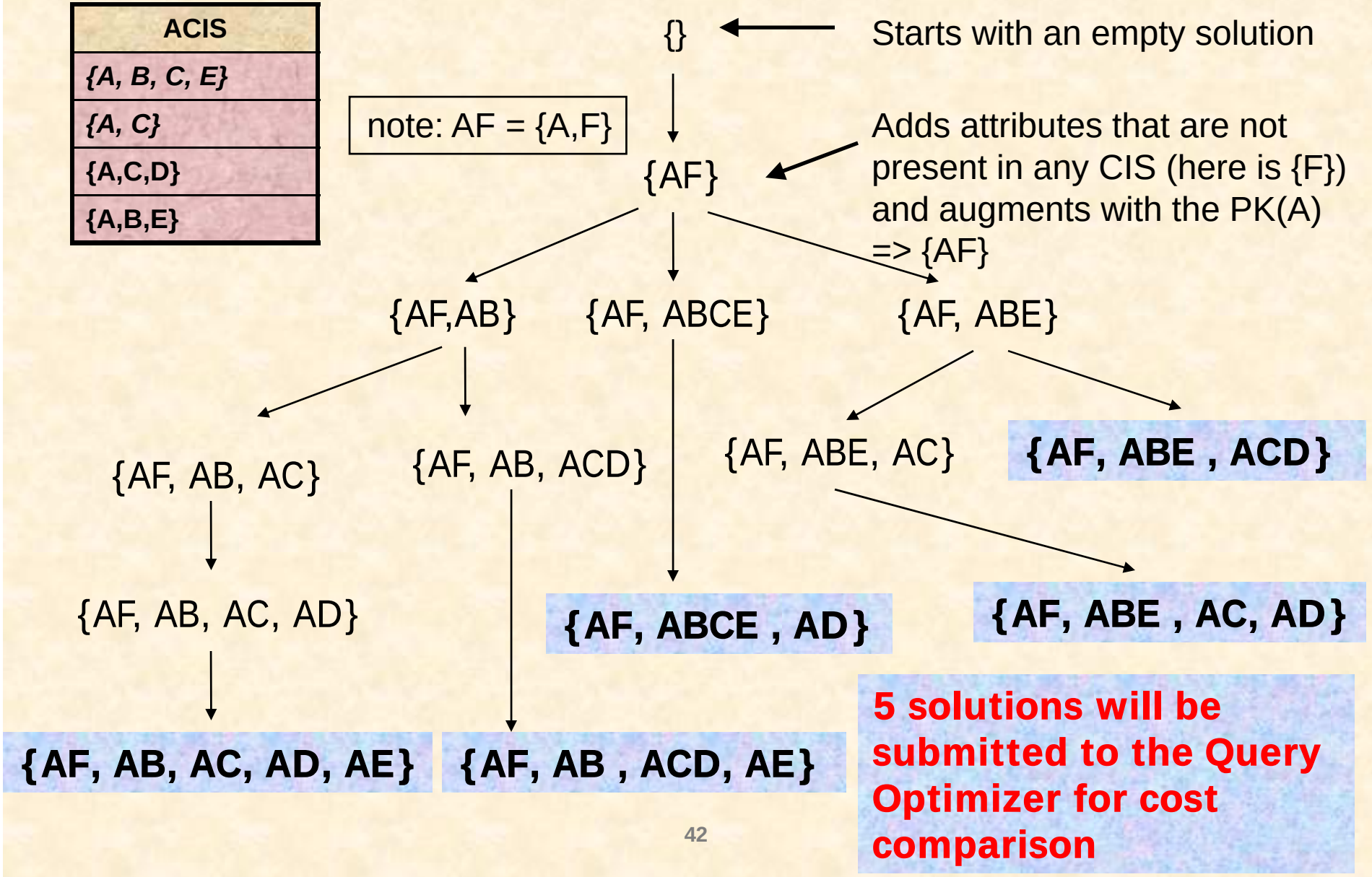
END MAIN

```
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
THEN
    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.
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

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

note: AF = {A,F}



5 solutions will be submitted to the Query Optimizer for cost comparison

Experimental Results

- 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.
  - $S_1 = \{\{A, F\}, \{A, B\}, \{A, C\}, \{A, D\}, \{A, E\}\}$
  - $S_2 = \{\{A, F\}, \{A, B\}, \{A, C, D\}, \{A, E\}\}$
  - $S_3 = \{\{A, F\}, \{A, B, C, E\}, \{A, D\}\}$
  - $S_4 = \{\{A, F\}, \{A, B, E\}, \{A, C\}, \{A, D\}\}$
  - $S_5 = \{\{A, F\}, \{A, B, E\}, \{A, C, D\}\}$

Experimental Results

- 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

Solutions	Estimated Individual Query Cost				Aggregate Cost
	Q1	Q2	Q3	Q4	
S1	3.857	5.601	3.121	5.601	468.27
S2	2.658	5.779	3.121	5.779	466.95
S3	4.403	3.026	3.026	3.026	316.38
S4	3.857	4.406	1.926	4.406	360.79
S5	2.657	4.584	1.926	4.584	359.47

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.**

Performance Study

- 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.

Performance Study

- 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 Study

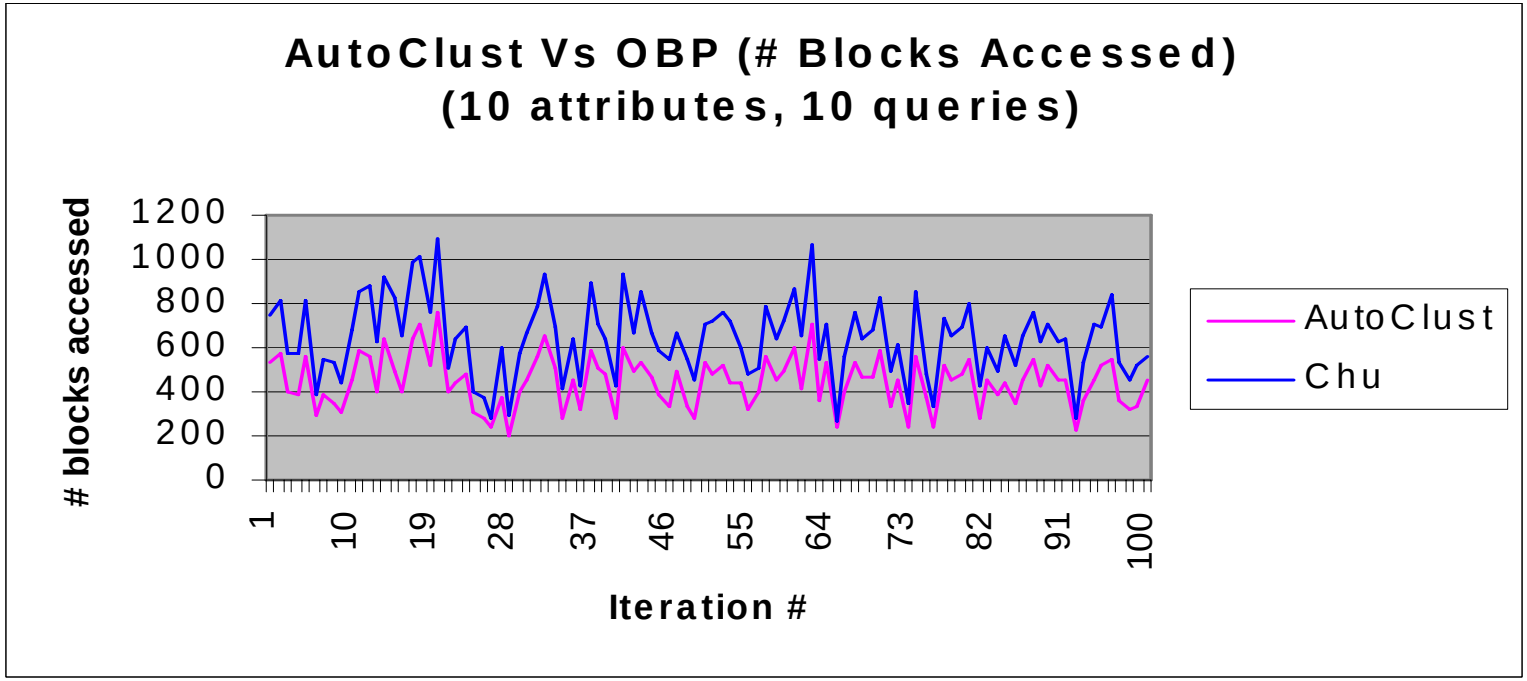
- **Performance Metrics:**
  - **number of disk accesses in response to the set of queries running against the database;**
  - **running time of the technique.**

Simulation Results

- **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.**

Simulation Results

- First data set

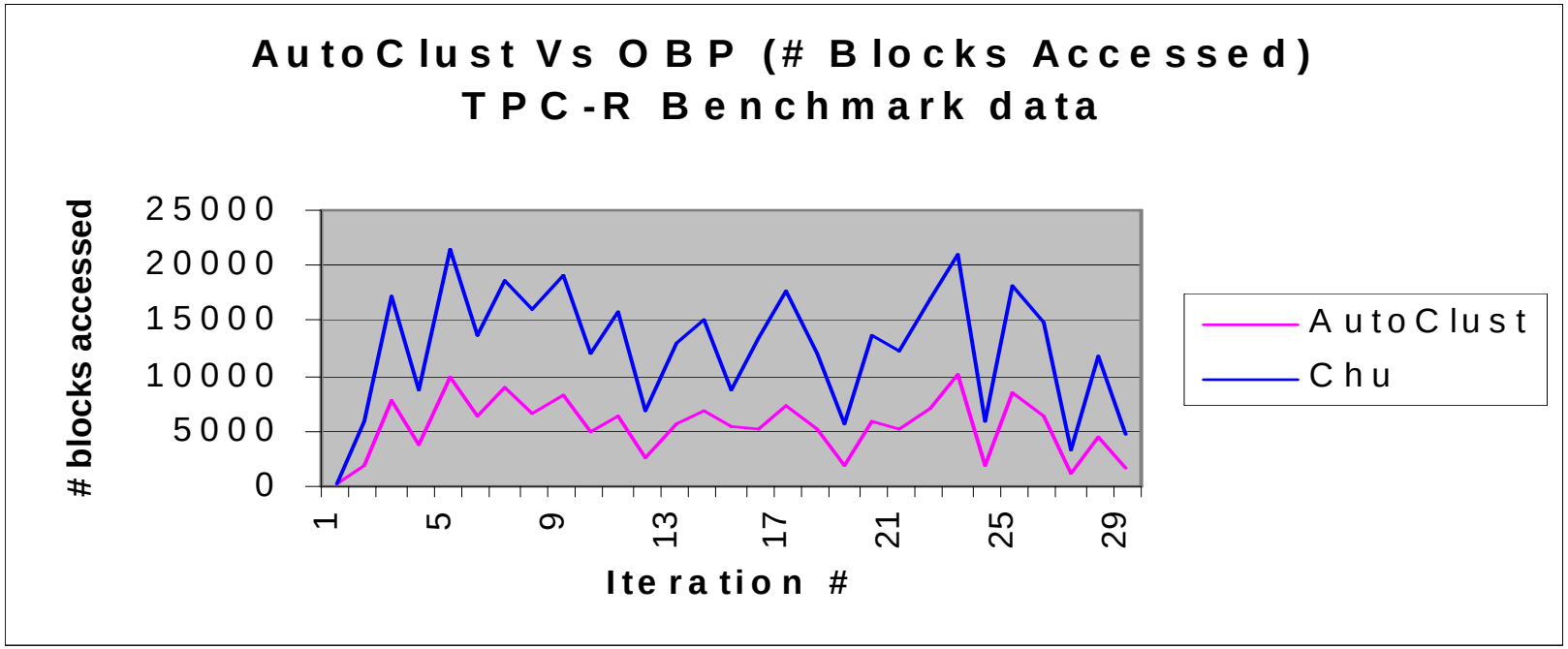


Simulation Results

- **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 enourmous size of the search space.**

Simulation Results

- Second data set

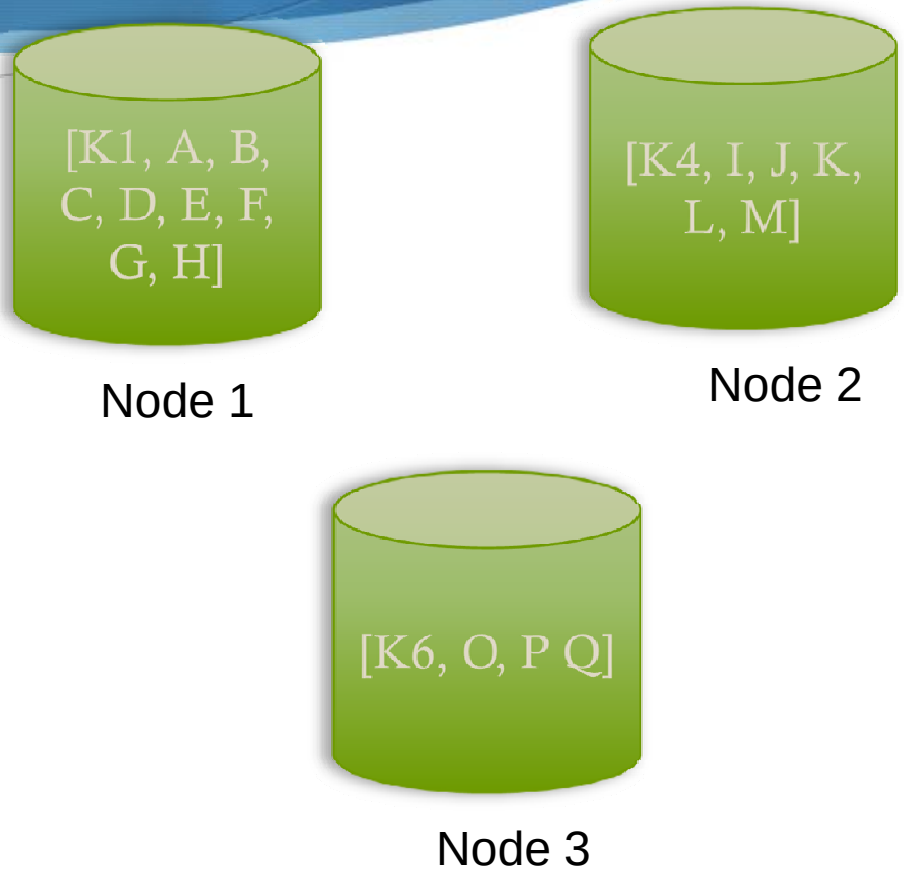


AutoClust and Cluster Computing

- 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

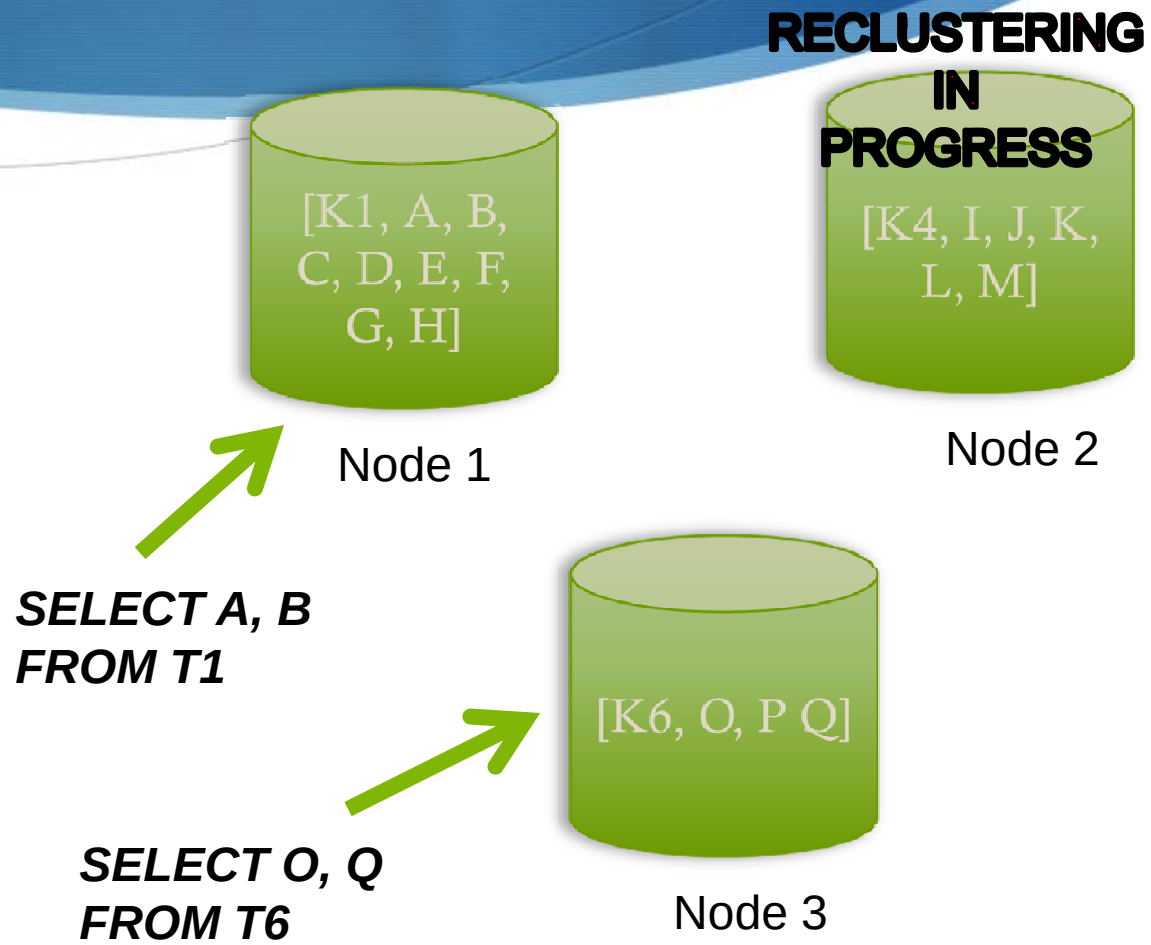
Cluster Computing w/o Replication

- Imagine the following database relations:
  - T1=(K1, A, B, C, D)
  - T2=(K2, E, F)
  - T3=(K3, G, H)
  - T4=(K4, I, J)
  - T5=(K5, K, L, M)
  - T6=(K6, O, P, Q)
- Let us assume further that there is a 1-to-1 relationship between
  - K1, K2, and K3
  - K4 and K5
- 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.



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

Solution	Estimated Individual Query Cost				Aggregate Cost
	Q1	Q2	Q3	Q4	
S1	3.85	5.60	3.12	5.60	468.27
S2	2.65	5.77	3.12	5.77	466.95
S3	4.40	3.02	3.02	3.02	316.38
S4	3.85	4.40	1.92	4.40	360.79
S5	2.65	4.58	1.92	4.58	359.47

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

Cluster Computing with Replication

- Each query can run on a different node based on expected query 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

Cluster Computing with Replication

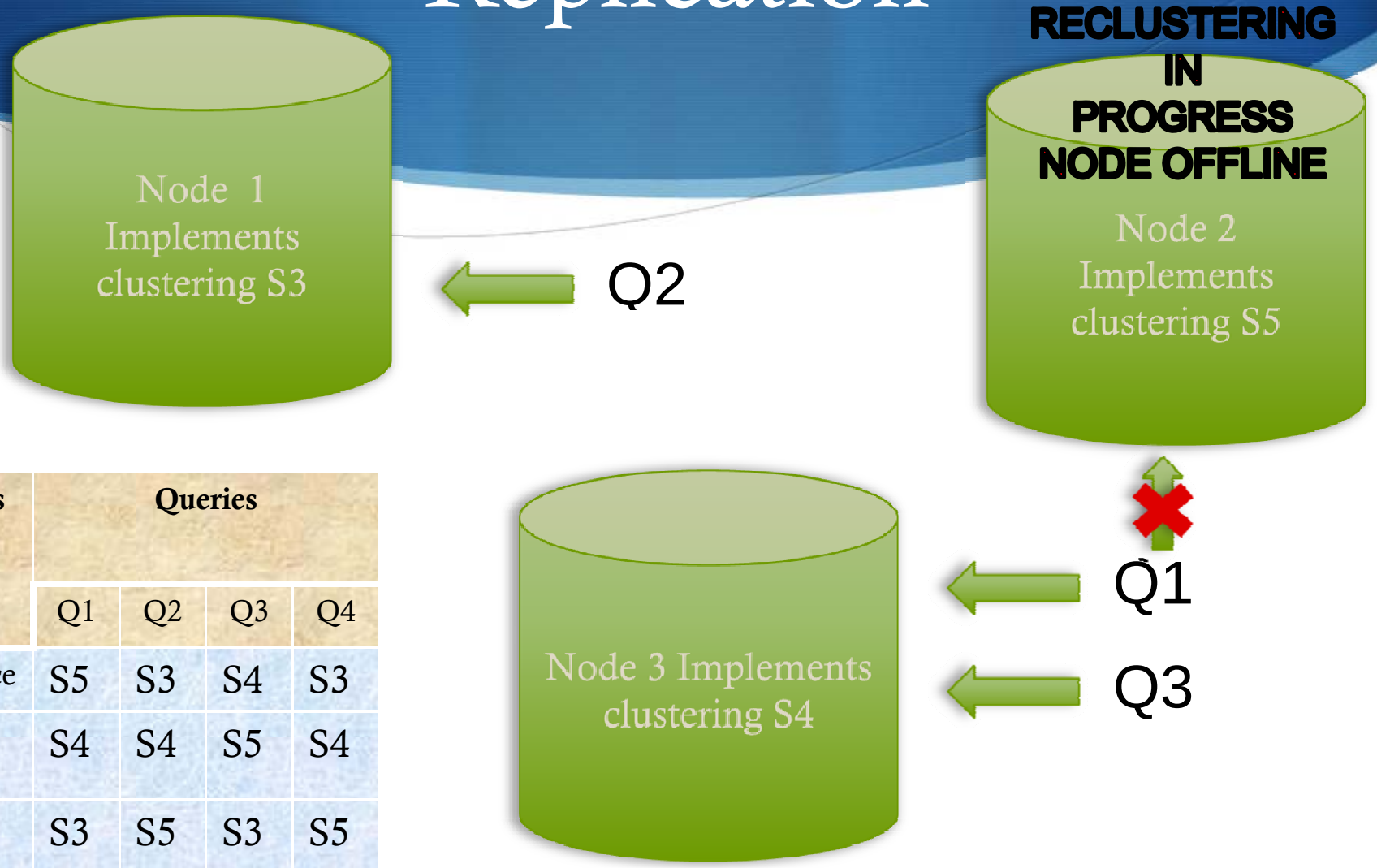
- The routing table is obtained by inverting the clustering result table produced by AutoClust.

Solutions	Estimated Individual Query Cost				Aggregate Cost
	Q1	Q2	Q3	Q4	
~~S1~~	~~3.85~~	~~5.60~~	~~3.12~~	~~5.60~~	~~468.27~~
~~S2~~	~~2.65~~	~~5.77~~	~~3.12~~	~~5.77~~	~~466.95~~
S3	4.40	3.02	3.02	3.02	316.38
S4	3.85	4.40	1.92	4.40	360.79
S5	2.65	4.58	1.92	4.58	359.47

Choices	Queries			
	Q1	Q2	Q3	Q4
1st Choice	S5	S3	S4	S3
2nd Choice	S4	S4	S5	S4
3rd Choice	S3	S5	S3	S5

Routing Table

Cluster Computing with Replication



Choices	Queries			
	Q1	Q2	Q3	Q4
1st Choice	S5	S3	S4	S3
2nd Choice	S4	S4	S5	S4
3rd Choice	S3	S5	S3	S5

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.