Distributed Database Systems

A classic perspective

Katja Hose

Department of Computer Science
Aalborg University
khose@cs.aau.dk

Dagstuhl
June 27, 2017
Example: distributed database system

Scenario
- Company with offices in Aalborg, Berlin, New York, and Paris
- Database of products and customers
Introduction

What is distributed data management?

Example: distributed database system

Definition: distributed database

A collection of multiple, logically interrelated databases distributed over a computer network.
Challenges

- Distributed database design
  
  *Fragmentation, replication, and distribution*
Challenges

- Distributed database design
  \textit{Fragmentation, replication, and distribution}
- Distributed query processing
  \textit{Executing a query over the network in the most cost-effective way}
Challenges

- Distributed database design
  
  *Fragmentation, replication, and distribution*

- Distributed query processing
  
  *Executing a query over the network in the most cost-effective way*

- Distributed concurrency control
  
  *Synchronizing access such that integrity is maintained*
Challenges

- Distributed database design
  *Fragmentation, replication, and distribution*

- Distributed query processing
  *Executing a query over the network in the most cost-effective way*

- Distributed concurrency control
  *Synchronizing access such that integrity is maintained*

- Reliability of distributed DBMS
  *Ensure consistency, detect failures, and recover from failures*
Challenges

- Distributed database design
  *Fragmentation, replication, and distribution*

- Distributed query processing
  *Executing a query over the network in the most cost-effective way*

- Distributed concurrency control
  *Synchronizing access such that integrity is maintained*

- Reliability of distributed DBMS
  *Ensure consistency, detect failures, and recover from failures*

- Heterogeneous databases
  *Translation between database systems – data model, data language*
Challenges

- Distributed database design
  *Fragmentation, replication, and distribution*

- Distributed query processing
  *Executing a query over the network in the most cost-effective way*

- Distributed concurrency control
  *Synchronizing access such that integrity is maintained*

- Reliability of distributed DBMS
  *Ensure consistency, detect failures, and recover from failures*

- Heterogeneous databases
  *Translation between database systems – data model, data language*
1 Introduction
   - What is distributed data management?
   - Challenges

2 Fragmentation and Allocation

3 Distributed query processing
   - Data localization
   - Global Query Optimization
   - Join Order Optimization
   - Query Execution
Fragmentation and Allocation

relation R  fragments  allocation to nodes

node 1
node 2
node 3
Fragmentation

Alternatives

- Tuples vs. columns, i.e., horizontal vs. vertical

Many algorithms proposed in the literature
Allocation

Golden rules of allocation
- Place data as close as possible to where it will be used
- Use load balancing to find a global optimization of system performance

Optimal allocation
Objective:
\[
\text{Minimize storage } (\sum S) \text{ and transfer costs } (\sum T) \text{ for } P \text{ fragments and } K \text{ nodes}
\]

Optimization problem
Find minimum
\[
\sum S + \sum T
\]

Often heuristics are applied

Many algorithms proposed in the literature
Allocation

Golden rules of allocation

- Place data as close as possible to where it will be used
- Use load balancing to find a global optimization of system performance

Optimal allocation

- Objective:
  Minimize storage ($\sum S$) and transfer costs ($\sum T$) for $P$ fragments and $K$ nodes

- Optimization problem
  - Find minimum $\sum S + \sum T$
  - Often heuristics are applied
Allocation

Golden rules of allocation

- Place data as close as possible to where it will be used
- Use load balancing to find a global optimization of system performance

Optimal allocation

- Objective:
  Minimize storage ($\sum S$) and transfer costs ($\sum T$) for $P$ fragments and $K$ nodes
- Optimization problem
  - Find minimum $\sum S + \sum T$
  - Often heuristics are applied

Many algorithms proposed in the literature
Introduction
- What is distributed data management?
- Challenges

Fragmentation and Allocation

Distributed query processing
- Data localization
- Global Query Optimization
- Join Order Optimization
- Query Execution
Phases of distributed query processing

1. Query Parsing
2. Query Transformation
3. Data Localization
4. Global Query Optimization
5. Local Query Optimization
6. Local Query Execution
7. Result
Phases of distributed query processing

- Query Parsing
- Query Transformation
- Data Localization
- Global Query Optimization
- Local Query Optimization
- Local Query Execution
- Post-processing

Global Meta Data Catalog
Local Meta Data Catalogs
Data
Example – horizontal reduction

Schema

- \( \text{PROJECTS}_1 = \sigma_{\text{Budget} \leq 150.000}(\text{PROJECTS}) \)
- \( \text{PROJECTS}_2 = \sigma_{150.000 < \text{Budget} \leq 200.000}(\text{PROJECTS}) \)
- \( \text{PROJECTS}_3 = \sigma_{\text{Budget} > 200.000}(\text{PROJECTS}) \)
Example – horizontal reduction

Schema

- $\text{PROJECTS}_1 = \sigma_{\text{Budget} \leq 150.000}(\text{PROJECTS})$
- $\text{PROJECTS}_2 = \sigma_{150.000 < \text{Budget} \leq 200.000}(\text{PROJECTS})$
- $\text{PROJECTS}_3 = \sigma_{\text{Budget} > 200.000}(\text{PROJECTS})$

Reconstruction expression (horizontal fragmentation)

- $\text{PROJECTS} = \text{PROJECTS}_1 \cup \text{PROJECTS}_2 \cup \text{PROJECTS}_3$
Example – horizontal reduction

Schema

- \( \text{PROJECTS}_1 = \sigma_{\text{Budget} \leq 150.000}(\text{PROJECTS}) \)
- \( \text{PROJECTS}_2 = \sigma_{150.000 < \text{Budget} \leq 200.000}(\text{PROJECTS}) \)
- \( \text{PROJECTS}_3 = \sigma_{\text{Budget} > 200.000}(\text{PROJECTS}) \)

Reconstruction expression (horizontal fragmentation)

- \( \text{PROJECTS} = \text{PROJECTS}_1 \cup \text{PROJECTS}_2 \cup \text{PROJECTS}_3 \)

Example query

- \( \sigma_{\text{Location} = 'Aalborg' \land \text{Budget} \leq 100.000}(\text{PROJECTS}) \)
Example – horizontal reduction

Schema

- \( \text{PROJECTS}_1 = \sigma_{\text{Budget} \leq 150.000}(\text{PROJECTS}) \)
- \( \text{PROJECTS}_2 = \sigma_{150.000 < \text{Budget} \leq 200.000}(\text{PROJECTS}) \)
- \( \text{PROJECTS}_3 = \sigma_{\text{Budget} > 200.000}(\text{PROJECTS}) \)

Reconstruction expression (horizontal fragmentation)

- \( \text{PROJECTS} = \text{PROJECTS}_1 \cup \text{PROJECTS}_2 \cup \text{PROJECTS}_3 \)

Example query

- \( \sigma \text{Location} = 'Aalborg' \land \text{Budget} \leq 100.000(\text{PROJECTS}) \)

After replacing references to global relations

- \( \sigma \text{Location} = 'Aalborg' \land \text{Budget} \leq 100.000(\text{PROJECTS}_1 \cup \text{PROJECTS}_2 \cup \text{PROJECTS}_3) \)
Example – horizontal reduction

Schema

- \text{PROJECTS}_1 = \sigma_{\text{Budget} \leq 150.000}(\text{PROJECTS})
- \text{PROJECTS}_2 = \sigma_{150.000 < \text{Budget} \leq 200.000}(\text{PROJECTS})
- \text{PROJECTS}_3 = \sigma_{\text{Budget} > 200.000}(\text{PROJECTS})

Reconstruction expression (horizontal fragmentation)

- \text{PROJECTS} = \text{PROJECTS}_1 \cup \text{PROJECTS}_2 \cup \text{PROJECTS}_3

Example query

- \sigma_{\text{Location} = 'Aalborg' \land \text{Budget} \leq 100.000}(\text{PROJECTS})

After replacing references to global relations

- \sigma_{\text{Location} = 'Aalborg' \land \text{Budget} \leq 100.000}(\text{PROJECTS}_1 \cup \text{PROJECTS}_2 \cup \text{PROJECTS}_3)

Further optimization is possible!
Example – horizontal reduction

Objective

Eliminate non-necessary subqueries (contradiction between query and fragmentation predicate)

Query with fragmentation expression

$$\sigma Location='Aalborg' \land Budget \leq 100,000 (\text{PROJECTS}_1 \cup \text{PROJECTS}_2 \cup \text{PROJECTS}_3)$$

Fragment definitions

- $\text{PROJECTS}_1 = \sigma_{Budget \leq 150,000} (\text{PROJECTS})$
- $\text{PROJECTS}_2 = \sigma_{150,000 < Budget \leq 200,000} (\text{PROJECTS})$
- $\text{PROJECTS}_3 = \sigma_{Budget > 200,000} (\text{PROJECTS})$
Example – horizontal reduction

Objective

Eliminate non-necessary subqueries (contradiction between query and fragmentation predicate)

Query with fragmentation expression

\[ \sigma_{\text{Location}=\text{Aalborg}} \land \text{Budget} \leq 100.000 (\text{PROJECTS}_1 \cup \text{PROJECTS}_2 \cup \text{PROJECTS}_3) \]

Fragment definitions

- \( \text{PROJECTS}_1 = \sigma_{\text{Budget} \leq 150.000} (\text{PROJECTS}) \)
- \( \text{PROJECTS}_2 = \sigma_{150.000 < \text{Budget} \leq 200.000} (\text{PROJECTS}) \)
- \( \text{PROJECTS}_3 = \sigma_{\text{Budget} > 200.000} (\text{PROJECTS}) \)

Because of

- \( \sigma_{\text{Budget} \leq 100.000} (\text{PROJECTS}_2) = \emptyset \)
- \( \sigma_{\text{Budget} \leq 100.000} (\text{PROJECTS}_3) = \emptyset \)

We obtain the reduced query

\[ \sigma_{\text{Location}=\text{Aalborg}} \left( \sigma_{\text{Budget} \leq 100.000} (\text{PROJECTS}_1) \right) \]
Example – join reduction

Query

\textsc{Projects} \bowtie \textsc{Assignment}

After replacing global relations with reconstruction expressions

(\textsc{Projects}_1 \cup \textsc{Projects}_2 \cup \textsc{Projects}_3) \bowtie (\textsc{Assignment}_1 \cup \textsc{Assignment}_2)

After applying the distributive law

(\textsc{Projects}_1 \bowtie \textsc{Assignment}_1) \cup (\textsc{Projects}_1 \bowtie \textsc{Assignment}_2) \cup (\textsc{Projects}_2 \bowtie \textsc{Assignment}_1) \cup (\textsc{Projects}_2 \bowtie \textsc{Assignment}_2) \cup (\textsc{Projects}_3 \bowtie \textsc{Assignment}_1) \cup (\textsc{Projects}_3 \bowtie \textsc{Assignment}_2)

Further optimization is possible!
Example – join reduction

Query \( \text{PROJECTS} \bowtie \text{ASSIGNMENT} \)

After replacing global relations with reconstruction expressions

\[
(\text{PROJECTS}_1 \cup \text{PROJECTS}_2 \cup \text{PROJECTS}_3) \bowtie (\text{ASSIGNMENT}_1 \cup \text{ASSIGNMENT}_2)
\]
Example – join reduction

Query

\[ \text{Projects} \Join \text{Assignment} \]

After replacing global relations with reconstruction expressions

\[ (\text{Projects}_1 \cup \text{Projects}_2 \cup \text{Projects}_3) \Join (\text{Assignment}_1 \cup \text{Assignment}_2) \]

After applying the distributive law

\[ (\text{Projects}_1 \Join \text{Assignment}_1) \cup (\text{Projects}_1 \Join \text{Assignment}_2) \cup (\text{Projects}_2 \Join \text{Assignment}_1) \cup (\text{Projects}_2 \Join \text{Assignment}_2) \cup (\text{Projects}_3 \Join \text{Assignment}_1) \cup (\text{Projects}_3 \Join \text{Assignment}_2) \]
Example – join reduction

Query \( \text{PROJECTS} \bowtie \text{ASSIGNMENT} \)

After replacing global relations with reconstruction expressions

\[
(\text{PROJECTS}_1 \cup \text{PROJECTS}_2 \cup \text{PROJECTS}_3) \bowtie (\text{ASSIGNMENT}_1 \cup \text{ASSIGNMENT}_2)
\]

After applying the distributive law

\[
(\text{PROJECTS}_1 \bowtie \text{ASSIGNMENT}_1) \cup (\text{PROJECTS}_1 \bowtie \text{ASSIGNMENT}_2) \cup \\
(\text{PROJECTS}_2 \bowtie \text{ASSIGNMENT}_1) \cup (\text{PROJECTS}_2 \bowtie \text{ASSIGNMENT}_2) \cup \\
(\text{PROJECTS}_3 \bowtie \text{ASSIGNMENT}_1) \cup (\text{PROJECTS}_3 \bowtie \text{ASSIGNMENT}_2)
\]

Further optimization is possible!
Example – join reduction

Query with fragmentation expression

\[(\text{PROJECTS}_1 \Join \text{ASSIGNMENT}_1) \cup (\text{PROJECTS}_1 \Join \text{ASSIGNMENT}_2) \cup (\text{PROJECTS}_2 \Join \text{ASSIGNMENT}_1) \cup (\text{PROJECTS}_2 \Join \text{ASSIGNMENT}_2) \cup (\text{PROJECTS}_3 \Join \text{ASSIGNMENT}_1) \cup (\text{PROJECTS}_3 \Join \text{ASSIGNMENT}_2)\]

Some of these partial joins are empty, e.g.: 

\[\text{PROJECTS}_1 \Join \text{ASSIGNMENT}_2 = \emptyset\]

Because their fragmentation expressions contradict:

\[\text{PROJECTS}_1 = \sigma_{P_{No}='P1' \lor P_{No}='P2'}(\text{PROJECTS}) \quad \text{and} \quad \text{ASSIGNMENT}_2 = \sigma_{P_{No}='P3' \lor P_{No}='P4'}(\text{ASSIGNMENT})\]
Example – join reduction

Query with fragmentation expression

\[(\text{PROJECTS}_1 \Join \text{ASSIGNMENT}_1) \cup (\text{PROJECTS}_1 \Join \text{ASSIGNMENT}_2) \cup (\text{PROJECTS}_2 \Join \text{ASSIGNMENT}_1) \cup (\text{PROJECTS}_2 \Join \text{ASSIGNMENT}_2) \cup (\text{PROJECTS}_3 \Join \text{ASSIGNMENT}_1) \cup (\text{PROJECTS}_3 \Join \text{ASSIGNMENT}_2)\]

Some of these partial joins are empty, e.g.: 

\[\text{PROJECTS}_1 \Join \text{ASSIGNMENT}_2 = \emptyset\]

Because their fragmentation expressions contradict:

\[\text{PROJECTS}_1 = \sigma_{P\text{No}='P_1' \lor P\text{No}='P_2'}(\text{PROJECTS}) \text{ and } \]
\[\text{ASSIGNMENT}_2 = \sigma_{P\text{No}='P_3' \lor P\text{No}='P_4'}(\text{ASSIGNMENT})\]

Reduced query

\[(\text{PROJECTS}_1 \Join \text{ASSIGNMENT}_1) \cup (\text{PROJECTS}_2 \Join \text{ASSIGNMENT}_2) \cup (\text{PROJECTS}_3 \Join \text{ASSIGNMENT}_2)\]
Phases of distributed query processing

1. Query Parsing
2. Query Transformation
3. Data Localization
4. Global Query Optimization
5. Local Query Optimization
6. Local Query Execution
7. Data
8. Result

Global Meta Data Catalog
Global query optimization

Phases

1. Spanning the search space using transformation rules
   → equivalent search plans

2. Applying a search strategy and a cost model
   → choose an efficient plan

Main focus: join trees and join ordering
Search space

Tree variants for join order optimization

- **Linear join trees**
  - All inner nodes have at least one leaf node (base relation) as child
  - Reduces search space

- **Bushy trees**
  - May have inner nodes with no base relation as child
  - High potential for parallelization
Search strategy

Deterministic search strategy
- Systematic generation of query plans
- Starting with plans accessing the base relations
- Constructing complex plans by combining easier plans, e.g., joining one more relation at each step

Implementations
- Exhaustive search guarantees finding the best plan
- Dynamic programming
- Greedy algorithm

Cost model
- Detailed statistics
- Calibrated parameters
- Formulas (cardinality estimation, processing costs, communication costs, etc.)
Site selection and data transfer

Query shipping

- Query initiator (node at which the query is issued/optimized) sends the query to other nodes
- Receiver nodes compute the query result and ship the result back to the initiator
Site selection and data transfer

Data shipping

- Query remains at the initiator
- Initiator sends **data request**
  messages to other nodes
- Receiver nodes **ship all required data** to the initiator
- Initiator computes result
Hybrid shipping

- Initiator sends **partial queries** to other nodes
- Other nodes **execute some parts of the query** and send intermediate results to the initiator
- Initiator executes remaining query operations (post-processing)
Ship whole

Execution at $node_R$
Ship whole

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Execution at node$_R$

- **node$_R$**: send data request message (relation $S$) to node$_S$

\[ R \Join S \]

\[
\begin{array}{cccc}
A & B & C & D \\
1 & 1 & 5 & 1 \\
4 & 5 & 7 & 8 \\
\end{array}
\]
Execution at $node_R$

- $node_R$: send data request message (relation $S$) to $node_S$
- $node_S$: send requested data (relation $S$) to $node_R$
Ship whole

\[
\begin{array}{cc|cc}
\text{R} & \text{A} & \text{B} \\
3 & 7 & & \\
1 & 1 & & \\
4 & 6 & & \\
7 & 7 & & \\
4 & 5 & & \\
6 & 2 & & \\
5 & 7 & & \\
\end{array}
\quad
\begin{array}{ccc|cc}
\text{S} & \text{B} & \text{C} & \text{D} \\
9 & 8 & 8 & \\
1 & 5 & 1 & \\
9 & 4 & 2 & \\
4 & 3 & 3 & \\
4 & 2 & 6 & \\
5 & 7 & 8 & \\
\end{array}
\]

\[
\begin{array}{cccc|cccc}
\text{R} \bowtie \text{S} & \text{A} & \text{B} & \text{C} & \text{D} \\
1 & 1 & 5 & 1 & \\
4 & 5 & 7 & 8 & \\
\end{array}
\]

Execution at \textit{node}_R

- \textit{node}_R: send data request message (relation \textit{S}) to \textit{node}_S
- \textit{node}_S: send requested data (relation \textit{S}) to \textit{node}_R

Total costs: 2 messages, 18 attribute values
Execution at \( node_R \): 2 messages, 18 attribute values

Execution at \( node_S \): 2 messages, 14 attribute values

Execution at \( node_X \): 4 messages, \( 18 + 14 = 32 \) attribute values
Fetch as needed

Execution at $node_R$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>9</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

$R \bowtie S$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>
Fetch as needed

Execution at \( \text{node}_R \)

- \( \text{node}_R \): send data request message (tuples of relation \( S \) with \( B = '7' \)) to \( \text{node}_S \)
Fetch as needed

**Execution at node** $R$

- *node*$_R$: send data request message (tuples of relation $S$ with $B = '7'\) to *node*$_S$
- *node*$_S$: send requested data (0 tuples of relation $S$ with $B = '7'\) to *node*$_R$
Fetch as needed

Execution at node\(_R\)

- \(\text{node}_R\): send data request message (tuples of relation \(S\) with \(B = '7'\)) to \(\text{node}_S\)
- \(\text{node}_S\): send requested data (0 tuples of relation \(S\) with \(B = '7'\)) to \(\text{node}_R\)
- \(\text{node}_R\): send data request message (tuples of relation \(S\) with \(B = '1'\)) to \(\text{node}_S\)

### Relations:

\[\begin{array}{c|c|c|c|c|c}
R & A & B & C & D \\
3 & 3 & 7 & 8 & 12 \\
1 & 1 & 6 & 4 & 1 \\
4 & 4 & 7 & 9 & 7 \\
7 & 5 & 4 & 4 & 2 \\
6 & 6 & 2 & 5 & 3 \\
5 & 7 & 7 & 7 & 8 \\
\end{array}\]

\[\begin{array}{c|c|c|c|c|c}
S & B & C & D \\
9 & 9 & 8 & 8 \\
1 & 1 & 5 & 1 \\
4 & 4 & 4 & 2 \\
9 & 9 & 3 & 3 \\
4 & 4 & 2 & 6 \\
5 & 5 & 7 & 8 \\
\end{array}\]

\[\begin{array}{c|c|c|c|c|c}
R \bowtie S & A & B & C & D \\
1 & 1 & 5 & 1 \\
4 & 5 & 7 & 8 \\
\end{array}\]
Distributed Database Systems
Distributed query processing
Query Execution

Fetch as needed

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R ⋊ S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Execution at $node_R$

- $node_R$: send data request message (tuples of relation $S$ with $B = '7'$) to $node_S$
- $node_S$: send requested data (0 tuples of relation $S$ with $B = '7'$) to $node_R$
- $node_R$: send data request message (tuples of relation $S$ with $B = '1'$) to $node_S$
- $node_S$: send requested data (1 tuple of relation $S$ with $B = '1'$) to $node_R$
Fetch as needed

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R (\bowtie S)</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Execution at \(node_R\)

- \(node_R\): send data request message (tuples of relation \(S\) with \(B = '7'\)) to \(node_S\)
- \(node_S\): send requested data (0 tuples of relation \(S\) with \(B = '7'\)) to \(node_R\)
- \(node_R\): send data request message (tuples of relation \(S\) with \(B = '1'\)) to \(node_S\)
- \(node_S\): send requested data (1 tuple of relation \(S\) with \(B = '1'\)) to \(node_R\)
- ...
Distributed Database Systems

Distributed query processing

Query Execution

Fetch as needed

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R ⋈ S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Execution at node<sub>R</sub>

- **node<sub>R</sub>:** send data request message (tuples of relation S with \(B = '7'\)) to node<sub>S</sub>
- **node<sub>S</sub>:** send requested data (0 tuples of relation S with \(B = '7'\)) to node<sub>R</sub>
- **node<sub>R</sub>:** send data request message (tuples of relation S with \(B = '1'\)) to node<sub>S</sub>
- **node<sub>S</sub>:** send requested data (1 tuple of relation S with \(B = '1'\)) to node<sub>R</sub>
- ...
Fetch as needed

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R ⋈ S</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Execution at node\(_R\)

- node\(_R\): send data request message (tuples of relation \(S\) with \(B = '7'\)) to node\(_S\)
- node\(_S\): send requested data (0 tuples of relation \(S\) with \(B = '7'\)) to node\(_R\)
- node\(_R\): send data request message (tuples of relation \(S\) with \(B = '1'\)) to node\(_S\)
- node\(_S\): send requested data (1 tuple of relation \(S\) with \(B = '1'\)) to node\(_R\)
- ...
Fetch as needed

Execution at node\(_R\)

- **node\(_R\)**: send data request message (tuples of relation \(S\) with \(B = '7'\)) to \(node\(_S\)\)
- **node\(_S\)**: send requested data (0 tuples of relation \(S\) with \(B = '7'\)) to \(node\(_R\)\)
- **node\(_R\)**: send data request message (tuples of relation \(S\) with \(B = '1'\)) to \(node\(_S\)\)
- **node\(_S\)**: send requested data (1 tuple of relation \(S\) with \(B = '1'\)) to \(node\(_R\)\)
- ...
Fetch as needed

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Execution at node₂ₐₜₜ

- nodeₐₜₜ: send data request message (tuples of relation S with \( B = '7' \)) to nodeₘₜₜ₉ₜ
- nodeₘₜₜ₉ₜ: send requested data (0 tuples of relation S with \( B = '7' \)) to node₂ₐₜₜ
- node₂ₐₜₜ: send data request message (tuples of relation S with \( B = '1' \)) to nodeₘₜₜ₉ₜ
- nodeₘₜₜ₉ₜ: send requested data (1 tuple of relation S with \( B = '1' \)) to node₂ₐₜₜ
- ...
Fetch as needed

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Execution at node$_R$

- node$_R$: send data request message (tuples of relation $S$ with $B = '7'$) to node$_S$
- node$_S$: send requested data (0 tuples of relation $S$ with $B = '7'$) to node$_R$
- node$_R$: send data request message (tuples of relation $S$ with $B = '1'$) to node$_S$
- node$_S$: send requested data (1 tuple of relation $S$ with $B = '1'$) to node$_R$
- ...

Total costs: $7 \cdot 2 = 14$ messages, $7 + 2 \cdot 3 = 13$ attribute values
Fetch as needed

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th></th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Execution at node $R$

- $node_R$: send data request message (tuples of relation $S$ with $B = '7'$) to $node_S$
- $node_S$: send requested data (0 tuples of relation $S$ with $B = '7'$) to $node_R$
- $node_R$: send data request message (tuples of relation $S$ with $B = '1'$) to $node_S$
- $node_S$: send requested data (1 tuple of relation $S$ with $B = '1'$) to $node_R$
- ...

Total costs: $7 \cdot 2 = 14$ messages, $7 + 2 \cdot 3 = 13$ attribute values

Execution at $node_S$: $6 \cdot 2 = 12$ messages, $6 + 2 \cdot 2 = 10$ attribute values
Conclusion

- Fetch as needed results in a high number of messages
- Ship whole results in high amounts of transferred data
Ship whole vs. fetch as needed

Conclusion

- Fetch as needed results in a high number of messages
- Ship whole results in high amounts of transferred data

More advanced join execution algorithms

- Semijoin
- Bitvector join
- ...
Semijoin

\[ Q = R \bowtie S' \]

\[ R' = \pi_B(R) \]

\[ S' = S \bowtie_B (R') \]
Bitvector join

\[ node_R \]

\[
\begin{array}{c|c|c}
R & A & B \\
4 & 3 & 1 \\
2 & 5 & 9 \\
8 & 6 & 2 \\
\end{array}
\]

\[ h(\pi_B(R)) = [0011011] \]

\[ h(v) = v \mod 7 \]

\[ Q = R \bowtie S' \]

\[
\begin{array}{c|c|c}
Q & A & B & C \\
4 & 3 & 1 & 6 \\
8 & 6 & 7 & 2 \\
\end{array}
\]

\[ node_S \]

\[
\begin{array}{c|c|c}
S & B & C \\
4 & 3 & 1 \\
3 & 6 & 8 \\
2 & 8 & 6 \\
8 & 1 & 7 \\
6 & 2 & 7 \\
\end{array}
\]

\[ B \quad h(B) \quad \text{hit} \]

\[
\begin{array}{c|c|c}
4 & 4 & - \\
3 & 3 & + \\
2 & 2 & + \\
8 & 1 & - \\
6 & 6 & + \\
\end{array}
\]
Query optimization in distributed database systems benefits from **global knowledge and control**.

- We can decide on fragmentation and allocation
- Create detailed statistics
- Finetune a cost model
- Optimize join processing (Semijoin, Bitvector join, ...)
- ...
Query optimization in distributed database systems benefits from **global knowledge and control**.

- We can decide on fragmentation and allocation
- Create detailed statistics
- Finetune a cost model
- Optimize join processing (Semijoin, Bitvector join, ... )
- ...

Further interesting topics
- Distributed concurrency control
- Reliability and failure
- Heterogeneity and data integration
- ...
This talk involves material based on the following sources:


Kai-Uwe Sattler *Vorlesung: Verteilte Datenbanksysteme*. 