Distributed Databases
Data Warehousing
CPS 216
Advanced Database Systems

Review
Distributed DBMS
- Top-down approach
  - Data partitioning
  - Query processing
  - Query optimization
  - Concurrency control and recovery
- Bottom-up approach
  - Query processing and optimization

Optimizing distributed queries
What is different from optimizing centralized queries?
- New strategies: parallel joins, semijoins, …
- Plans have a new property: “interesting sites”
- Communication cost is a big factor besides I/O
  - Per-message cost, per-byte cost, CPU cost to pack/unpack data
- Parallelism: response time versus total resource consumption

Concurrency control & recovery in Distributed DBMS
- Rich and interesting field
- We will just sample the field by looking at the problem of distributed transaction commit

Example: two-step optimization
- Step 1 (compile time): decide the join order, join methods, and access paths
  - Same complexity as in a centralized DBMS
- Step 2 (run time): decide where to execute each operator
  - Can cope with changing load and network characteristics
  - Can use data that has been dynamically allocated to a site (caching or replication)

Two-phase commit
- Notation: Incoming message
  - * = everyone

Plan A
Plan B

Plan B

Better:

Site 2
Site 1
Site 1
Key points of 2PC

- By sending OK a participant promises the coordinator to commit
  - But it can only commit when instructed to do so by the coordinator
  - The coordinator could tell it to abort instead
- After sending NOK a participant can abort unilaterally
- Coordinator can decide to commit only if all participants have responded OK
- Logging of all messages are required at each site

Bottom-up approach to Building a distributed DBMS

- Data already in various sources
- Build a distributed DBMS to provide global, uniform access to all data
  - How to integrate data?
  - How to deal with heterogeneous and autonomous sources?
    » Mediation approach

Wrapper/mediator architecture

![Diagram of a wrapper/mediator architecture](diagram)

Mediator

- Accept queries from clients
- Rewrite and optimize queries
- Send subplans to be executed by wrappers
- Combine results from wrappers and perform any additional local processing necessary
- Mediator catalog stores global schema and external schema of sources as exported by wrappers
  » No source-specific code in a mediator!

Query optimization with wrappers

Basic questions

- Capability: What types of subplans can be handled by a wrapper?
  - How do we enumerate valid plans?
- Cost: What is the cost of executing a subplan by a wrapper?
  - How do we pick the optimal plan?

Wrapper

- Hide heterogeneity away from mediator
- Translate mediator requests so that they are understood by sources
  - Example: SELECT * FROM Books WHERE title LIKE '%Databases'; → a form-based search request for books with title matching "**Databases"
- Translate results returned by a source so that they are compliant with its external schema
  - Example: result HTML page → Books tuples
Example: Garlic query optimization

- Haas et al., VLDB 1997
- Incorporated in DB2

- Rules for generating valid plans
  - Supplied by wrappers and mediator
  - Plugged into the optimizer
- Plans have “interesting properties”
  - Order (as in Selinger)
  - Site (where the output is produced)
  - Columns (in the output)
  - Predicates (that have been applied)
  - Cost, etc.

Example rules for a DBMS source

- \( \text{wrap-access}(\text{table}, \text{columns}, \text{predicates}) = \) SCAN\(_{\text{DBMS}}\)(\text{table}, \text{columns}, \text{predicates})
  - Condition: \text{table} is at my site
  - I can handle any projection and selection (by converting them to a single-table SELECT-FROM-WHERE SQL statement)

- \( \text{wrap-join}(\text{subplan}_1, \text{subplan}_2, \text{predicates}) = \) JOIN\(_{\text{DBMS}}\)(\text{subplan}_1, \text{subplan}_2, \text{predicates})
  - Condition: \text{subplan}_1, \text{site} = \text{subplan}_2, \text{site} = \text{my site}
  - I can handle any local join (by converting it to a multi-table SELECT-FROM-WHERE SQL statement)

Example rules for a Web source

- \( \text{wrap-access}(\text{table}, \text{columns}, \text{predicates}) = \) FETCH\(_{\text{Web}}\)(\text{Books}, \text{title LIKE string})
  - Condition: \text{table} = \text{Books}, (\text{title LIKE string}) \in \text{predicates}
  - I can search books by title (with wildcards); no projection

- \( \text{wrap-access}(\text{table}, \text{columns}, \text{predicates}) = \) FETCH\(_{\text{Web}}\)(\text{Books}, \text{author = string})
  - Condition: \text{table} = \text{Books}, (\text{author = string}) \in \text{predicates}
  - I can search books by exact author names; no projection
  - I cannot search books by title and author at the same time

- No wrap_join rule
  - I cannot handle process joins

Example rules for the mediator

- \( \text{med-pushdown}(\text{subplan}) = \) RECEIVE\((\text{SEND}(\text{subplan}))\)
  - Condition: \text{subplan}.\text{site} \neq \text{mediator}
- \( \text{med-pushdown}(\text{subplan}) = \) \text{subplan}
  - Condition: \text{subplan}.\text{site} = \text{mediator}

- \( \text{med-access}(\text{table}, \text{columns}, \text{predicates}) = \) \( \forall \text{plan} \in \text{wrap-access}(\text{table}, \text{columns}, \text{predicates})\):
  \( \text{FILTER}_{\text{med}}(\text{med-pushdown}(\text{plan})), \text{predicates} \in \text{plan}.\text{predicates} \)
  - I can get the result of a single-table scan from a wrapper and then evaluate remaining selection predicates

More rules for the mediator

- \( \text{med-join}(\text{subplan}_1, \text{subplan}_2, \text{predicates}) = \) \( \forall \text{plan} \in \text{wrap-join}(\text{subplan}_1, \text{subplan}_2, \text{predicates})\):
  \( \text{med-pushdown}(\text{plan}) \)
  - Condition: \text{subplan}_1.\text{site} = \text{subplan}_2.\text{site} \neq \text{mediator}
  - I can push down a join to a wrapper

- \( \text{med-join}(\text{subplan}_1, \text{subplan}_2, \text{predicates}) = \) JOIN\(_{\text{med}}\)(\text{med-pushdown}(\text{subplan}_1), \text{med-pushdown}(\text{subplan}_2), \text{predicates})
  - I also can handle a join locally

- And more…

Plan enumeration

- Call all \text{wrap-access} and \text{med-access} rules to generate single-table access plans
- Repeatedly call all \text{wrap-join} and \text{med-join} rules to generate multi-table join plans
- Example final plans
  - \( \text{FILTER}_{\text{med}}(\) RECEIVE\((\text{SEND}(\text{FETCH}_{\text{Web}}(\text{Books, title LIKE string}))), \text{author = string}), \) versus
    \( \text{FILTER}_{\text{med}}(\) RECEIVE\((\text{SEND}(\text{FETCH}_{\text{Web}}(\text{Books, author = string}))), \text{title LIKE string})\)
  - \( \text{JOIN}_{\text{med}}(\) RECEIVE\((\text{SEND}(\text{R})), \text{RECEIVE}(\text{SEND}(\text{S})))\)

13
14
15
16
17
18
Costing

- Wrapper-supplied cost model
  - Lots of work for wrapper developers
- Calibration
  - Define a generic cost model with parameters for all wrappers
    - Example: cost = \( c \cdot (\# \text{ of tuples}) \)
  - Run test queries to measure the parameters for each wrapper
- Learning curve
  - Use recent statistics to adjust cost estimates
    - Example: cost = average over last three runs

Summary of wrapper/mediator

Not all sources are created equal!

- What’s in a source?
  - Wrapper: source schema ↔ external schema
  - Mediator: external schema ↔ global schema
- What can it do?
  - Wrappers and mediators supply rules describing their query processing capabilities
- How much does it cost?
  - Wrappers supply cost model, or
  - Mediator calibrates or learns the cost model

Data warehousing

- Data resides in many distributed, heterogeneous OLTP (On-Line Transaction Processing) sources
  - Sales, inventory, customer, …
  - NC branch, NY branch, CA branch, …
- Need to support OLAP (On-Line Analytical Processing) over an integrated view of the data
  - Store the integrated data at a central repository called the data warehouse

OLTP versus OLAP

<table>
<thead>
<tr>
<th>OLTP</th>
<th>OLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mostly updates</td>
<td>Mostly reads</td>
</tr>
<tr>
<td>Short, simple transactions</td>
<td>Long, complex queries</td>
</tr>
<tr>
<td>Clerical users</td>
<td>Analysts, decision makers</td>
</tr>
<tr>
<td>Goal: ACID, transaction throughput</td>
<td>Goal: fast queries</td>
</tr>
</tbody>
</table>

Warehousing versus mediation

<table>
<thead>
<tr>
<th>Warehousing</th>
<th>Mediation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eager “integration”</td>
<td>Lazy “integration”</td>
</tr>
<tr>
<td>In advance: before queries</td>
<td>On demand: at query time</td>
</tr>
<tr>
<td>Answer could be stale</td>
<td>Answer is more up-to-date</td>
</tr>
<tr>
<td>Copy data from sources</td>
<td>Leave data at sources</td>
</tr>
<tr>
<td>Need to maintain consistency</td>
<td>No need to maintain consistency</td>
</tr>
<tr>
<td>Query processing is local to the warehouse</td>
<td>Sources participate in query processing</td>
</tr>
<tr>
<td>Faster</td>
<td></td>
</tr>
<tr>
<td>Can operate when sources are unavailable</td>
<td></td>
</tr>
</tbody>
</table>

Maintaining a data warehouse

Buzz word: the “ETL” process

- Extraction: extract relevant data and/or changes from sources
- Transformation: transform data to match the warehouse schema
- Loading: integrate data/changes into the warehouse
  - Can still use a wrapper/mediator architecture
Warehouse data = materialized views

- If the transformation process can be captured by SQL, then warehouse data can be seen as a view
  - CREATE VIEW warehouse_table AS
    SELECT ...
    FROM source_table1, source_table2, ...
    WHERE ...;
- Except the view is materialized
  - That is, the result is stored
  - And needs to be maintained when source data changes

Maintaining materialized views

\[ V_{\text{old}} = Q(R_{\text{old}}, ...) \]
Change detected: \[ R_{\text{new}} \leftarrow R_{\text{old}} - \nabla R \cup \Delta R \]
What is \( V_{\text{new}} \)?
- Recomputation: \[ V_{\text{new}} = Q(R_{\text{new}}, ...) \]
  - Done periodically, e.g., every “night”
  - What if there is no “night,” e.g., an international organization?
  - What if recomputation takes longer than a day?
- Incremental maintenance
  - Compute only the changes to \( V \):
    - \( \nabla V \) and \( \Delta V \)
  - Apply the changes to \( V_{\text{old}} \):
    - \( V_{\text{new}} = V_{\text{old}} - \nabla V \cup \Delta V \)
  - Potentially much faster if changes are small

Incremental maintenance

Example: \( V = \sigma_p R \)
- Change: \( R_{\text{new}} \leftarrow R_{\text{old}} - \nabla R \)
  - \( V_{\text{new}} = \sigma_p R_{\text{new}} = \sigma_p (R_{\text{old}} - \nabla R) = \sigma_p R_{\text{old}} - \sigma_p \nabla R \)
  - \( V_{\text{old}} - \sigma_p \nabla R \)
- Change: \( R_{\text{new}} \leftarrow R_{\text{old}} \cup \Delta R \)
  - \( V_{\text{new}} = \sigma_p R_{\text{new}} = \sigma_p (R_{\text{old}} \cup \Delta R) = \sigma_p R_{\text{old}} \cup \sigma_p \Delta R \)
  - \( V_{\text{old}} \cup \sigma_p \Delta R \)

Change propagation equations

Self-maintainability

- A warehouse is self-maintainable if it can be maintained without accessing the sources
- Self-maintainable: \( V = \sigma_p R \)
- Not self-maintainable: \( V = R \bowtie S \)
  - \( \Delta R \) and \( \nabla R \) need to be joined with \( S \)
  - \( \Delta S \) and \( \nabla S \) need to be joined with \( R \)
  - Problem: need to query the source for maintenance
    - What if the source/network is slow?
    - What if the source/network is down?
    - What if the source has been updated again?

Making warehouse self-maintainable

- Add auxiliary views
  Example: Order \( \bowtie\bowtie O.\text{OID} = L.OID \) AND Month = "nov" AND Product = "book" Lineitem
- Naïve approach: add base tables \( O \) and \( L \)
- Better approach: push selections down and then add selection views \( \sigma_{\text{month}} = \text{"nov"} \) \( O \) and \( \sigma_{\text{product}} = \text{"book"} \) \( L \)
- Use constraints
  - The join is a foreign-key join (\( L.OID \) references \( O.\text{OID} \)), so only \( \sigma_{\text{month}} = \text{"nov"} \) \( O \) is needed
  - If we only insert matching orders and lineitems together, then no auxiliary view is needed
Next time

- Warehouse design
- Data cube
- ROLAP versus MOLAP