Distributed Database Design and Distributed Query Execution

Designing with distribution in mind: top-down
Data Fragmentation and Placement

- **Fragmentation:**
  - How to split up the data into smaller fragments?

- **Placement:**
  - Where to put the fragments?

### Horizontal Data Fragmentation

- **Problem Statement:**
  - Given a table $R(a_1, a_2, a_3, ...)$, containing tuples $t_1, t_2, ...$
  - How to distribute $R$ across several sites

- **Horizontal Fragmentation**
  - For each tuple $t_i$, place $t_i$ on one of the sites
  - Fragmentation given through selection
  - Range partitioning: $\sigma_{a_1 < 10}(R)$, $\sigma_{10 \leq a_1 < 20}(R)$, $\sigma_{20 \leq a_1}(R)$
  - loc = 'Montreal', loc = 'Zurich'

- **Correctness:** Each tuple (also future tuples) is member of exactly one fragment (completeness and disjointness)
Referencing partition

- Assume relations Emp(eid, name, dep) and Dep(did, name, location); dep is foreign key referencing Dep
- Dep is partitioned into Dep_i according to location
  - \( \pi_{\text{did}} (\text{Dep}_i) \) are disjoint sets
- Emp is partitioned according to Dep
  - \( \text{Emp}_i = \text{Emp} \cap \pi_{\text{did}} (\text{Dep}_i) \)

SELECT eid, name, dep
FROM Emp, Dep_i
WHERE Emp.dep = Dep_i.did
\( \text{Emp}_i \) are also disjoint

Vertical Data Fragmentation

- Vertical Fragmentation
  - For each attribute \( a_i \), place for each tuple \( t_i \), the value of its attribute \( a_i \) on one of the sites.
  - In order to identify tuples, the key is replicated at each fragment!
  - determined by projection: \( \pi_{K,A_1,A_2}(R) \), \( \pi_{K,A_3,A_4}(R) \).
Query Processing I

- SELECT A, B FROM R WHERE E = 3
  - Generate Execution Tree

\[ \pi_{A,B} \bigcap_{E = 3} R \]
Query Processing II

 Localization 1: Replace Relations by Fragments

\[ \pi_{A,B} \]
\[ \bigcup \]
\[ E = 3 \]
\[ \bigcup \]
\[ [R1: E < 10] \]
\[ [R2: E >= 10] \]

---

Query Processing III

 Localization II

 Push \( \bigcup \) up
 Push \( \pi \) and \( \sigma \) down

\[ \pi_{A,B} \]
\[ \bigcup \]
\[ E = 3 \]
\[ \bigcup \]
\[ [R1: E < 10] \]
\[ [R2: E >= 10] \]
Query Processing IV

- Localization III
  - Simplify
    - Eliminate unnecessary operations

\[ E = 3 \]

\[ \pi_{A,B} \]
\[ E = 3 \]

\[ \text{[R1: } E < 10 \text{]} \]
\[ \text{[R2: } E \geq 10 \text{]} \]
\[ = \emptyset \]

Fragmentation, Data Placement and Query Execution

- LAN
  - Fragment such that your typical queries are executed fast
  - Full Scan
    - SELECT * FROM R
    - Many result tuples => distribute work
    - Idea: distribute data evenly among several sites and read data in parallel
    - Expect scanning time long => assemble and communication overhead negligible
  - Point Query
    - SELECT * FROM R WHERE a1 = 5
    - Few result tuples = localize work
    - Idea: if all relevant data on one site, only one site has to scan its partition; no work on the other sites
      - Expect search time short: assemble and communication overhead have too big impact

- WAN
  - Each query should be executed solely locally
Range Partitioning

- Partitioning Vector indicating domain ranges of an attribute $V=(4;7)$

- Parallel Query Execution
  - $\star$ + good for full scan if even distribution
  - $\star$ + good for range queries on partition attribute
  - $\star$ + ok for point queries
  - $\star$ - not good for range/point queries not on partition attribute
  - Need for good vector otherwise data skew and execution skew
  - Oracle provides range partitioning

- Local Query Execution
  - $\star$ Good if partition related to geography

Query Execution in p2p system

Example:


- $\star$ Data sharing: each node has the data it is interested in;
- $\star$ Query: first decides on nodes that query will run on
- $\star$ Query execution: distribute the execution of the query across the nodes
Data Storage

- Data Units are Relations
- Each node might have local tables (referred to as LT)
- Homogeneous set of global tables
  - Each node stores a subset of the existing tuples of a global table
    - Fragmentation with replication
    - Each node stores the tuples it is interested in for local use
    - Each global table T has special column node: a tuple T at node n1 stores n1 in this column
- Distributed Table:
  - given a set of nodes each having a local instance of a global table T
    - DT(T) is the union of all local instances
      - Duplicates possible if node column not considered
- Operations
  - SQL like queries over entire schema
    - Select/project/join/aggregation(sum,max,...)

Query Execution

- Query submitted at one node N
- Build logical routing tree with N as root
  - Tree covers all nodes query should run on
  - Uses Chord to generate tree
- Query execution in several wave cycles
  - Wave down the tree from root to leaves
  - Wave up from leaves to root
- Local Operations
  - Standard database operators on LT or local fragment of DT
- Global Unary Operations on DT
  - Select:
    - Down wave: execute select locally, forward query to children
    - Up wave: Union of results received from children and own result
Wave execution

Query Execution II

★ Agg Up wave:
  ▲ Receive results from children
  ▲ Merge with own results (e.g., max: take the highest, min, sum, count,
  • avg: requires children to send not only their avg but also the number of
tuples)

★ Order Up wave:
  ▲ perform merge sort of children data and own data at each node, or
  ▲ simple union (merge done at root)

☑ Join
★ Broadcast join LT against DT
  • Query node forwards LT, each node performs join with that LT and local
  fragment of DT
★ Foreignkey
  • Assume DT1 and DT2, DT2 has foreign key f to DT1
  • For each tuple at node n of fragment DT2, the referenced tuple is in DT1
  • Each node performs local join; union results (similar to select)
Joins

- \( \text{Join}(L,T, \text{Local DT}) \)
- \( \text{Join}(L,T, \text{Local DT}) \)
- \( \text{Join}(L,T, \text{Local DT}) \)
- \( \text{Join}(L,T, \text{Local DT}) \)

Partitioning

- Some operators will be faster if no replication
  - Transform DT into PT (partitioned table) without replication
    - Logical: not real deletion of replicas
  - Execute operation on PT instead of DT
Partitioning

3 cycle partitioning

Complex SQL Queries

- Combination of select/join/etc can lead to many wave cycles and data transfer
- Combine execution of operators if possible
- Query optimization
- Use DHT special indexes to speed up queries
Heterogeneous Query Processing

- Mediator / Middleware
- catalog
- Wrapper
- DB1
- DB1
- DB2

Work distribution

- Mediator
  - Schema Integration
    - must know the schemas of the component DB (external schemas)
    - find a reasonable global schema for the separate schemas
    - translate the external local schemas of a component DB into global schema and vs. versa
    - this is its whole own research area
    - very complex if data sources have different data models (Relational, OO, XML, html, file-system)
  - Own data dictionary (catalog)
    - global schema, translation mechanisms, information about local schemas, ...
    - some statistics of the component DB
      - size of tables, query capacity of component DB, etc.
Work distribution

- **Mediator**
  - global query execution
    - query parsing, rewrite, query optimization
    - executes some of the operations (joining, extra operations on the data retrieved from data sources)
  - might cache data

- **Wrapper (adaptor)**
  - translates every request of the mediator into request that is understood by the underlying DB
  - translates the results into format understood by the mediator
  - Wrappers for relational DB, file systems, web-based DB
  - wrapper is client of DB source: can cache data and execute parts of the query by itself (e.g. if data source only returns html pages)
  - Expect to have many wrappers for many different data sources
    - new data source -> new wrapper, but same mediator
  - Key feature of being extensible

- **Query optimization and execution**
  - distributed between mediator, wrapper and data source