Distributed Query Processing

Distributed Query Processing

- For centralized systems, the primary criterion for measuring the cost of a particular strategy is the number of disk accesses.
- In a distributed system, other issues must be taken into account:
  - The cost of a data transmission over the network.
  - The potential gain in performance from having several sites process parts of the query in parallel.

Query Transformation

- Translating algebraic queries on fragments.
  - It must be possible to construct relation *r* from its fragments
  - Replace relation *r* by the expression to construct relation *r* from its fragments
- Consider the horizontal fragmentation of the *account* relation into

  $account_1 = \sigma_{branch\_name = \text{“Hillside”}} (account)$

  $account_2 = \sigma_{branch\_name = \text{“Valleyview”}} (account)$
- The query $\sigma_{branch\_name = \text{“Hillside”}} (account)$ becomes

  $\sigma_{\text{branch\_name} = \text{“Hillside”}} (account_1 \cup account_2)$

  which is optimized into

  $\sigma_{branch\_name = \text{“Hillside”}} (account_1) \cup \sigma_{branch\_name = \text{“Hillside”}} (account_2)$

Example Query (Cont.)

- Since $account_1$ has only tuples pertaining to the Hillside branch, we can eliminate the selection operation.
- Apply the definition of $account_2$ to obtain

  $\sigma_{branch\_name\ =\ \text{“Hillside”}}\ (\sigma_{branch\_name\ =\ \text{“Valleyview”}}\ (account\ ))$

- This expression is the empty set regardless of the contents of the *account* relation.
- Final strategy is for the Hillside site to return $account_1$ as the result of the query.

Simple Join Processing

- Consider the following relational algebra expression in which the three relations are neither replicated nor fragmented

  account ⋈ *depositor* ⋈ *branch*

- *account* is stored at site S_1
- *depositor* at S_2
- *branch* at S_3
- For a query issued at site S_I, the system needs to produce the result at site S_I

Possible Query Processing Strategies

- Ship copies of all three relations to site S_I and choose a strategy for processing the entire locally at site $S_{I.}$
- Ship a copy of the account relation to site S_2 and compute $temp_1$ = *account* $\bowtie$ *depositor at* S_2. Ship $temp_1$ from S_2 to S_3, and compute $temp_2$ = $temp_1$ *branch* at S_3. Ship the result $temp_2$ to S_I.
- Devise similar strategies, exchanging the roles S_1, S_2, S_3
- Must consider following factors:
  - amount of data being shipped
  - cost of transmitting a data block between sites
  - relative processing speed at each site

Semijoin Strategy

- Let r_1 be a relation with schema R_1 stores at site S_1

  Let r_2 be a relation with schema R_2 stores at site S_2
- Evaluate the expression $r_1 \bowtie r_2$ and obtain the result at S_1.

1. Compute $temp_1 \leftarrow \Pi_{R1 \cap R2}$ (r1) at S1.

- 2. Ship $temp_1$ from S_1 to S_2.
- 3. Compute $temp_2 \leftarrow r_2 \bowtie$ temp1 at S_2
- 4. Ship $temp_2$ from S_2 to S_1.
- 5. Compute $r_1 \bowtie temp_2$ at S_1. This is the same as $r_1 \bowtie r_2$.

Formal Definition

- The **semijoin** of r_1 with r_2, is denoted by:

$$r_1 \ltimes r_2$$

- it is defined by:

$$\Pi_{R1} (r_1 \bowtie r_2)$$

- Thus, $r_1 \ltimes r_2$ selects those tuples of r_1 that contributed to $r_1 \bowtie r_2$.
- In step 3 above, $temp_2 = r_2 \ltimes r_1$.
- For joins of several relations, the above strategy can be extended to a series of semijoin steps.

Join Strategies that Exploit Parallelism

- Consider $r_1 \bowtie r_2 \bowtie r_3 \bowtie r_4$ where relation ri is stored at site S_i. The result must be presented at site S_1.
- r_1 is shipped to S_2 and $r_1 \bowtie r_2$ is computed at S_2: simultaneously r_3 is shipped to S_4 and $r_3 \bowtie r_4$ is computed at S_4
- S_2 ships tuples of $(r_1 \bowtie r_2)$ to S_1 as they produced;
  S_4 ships tuples of $(r_3 \bowtie r_4)$ to S_1
- Once tuples of $(r_1 \bowtie r_2)$ and $(r_3 \bowtie r_4)$ arrive at S_1 $(r_1 \bowtie r_2) \bowtie (r_3 \bowtie r_4)$ is computed in parallel with the computation of $(r_1 \bowtie r_2)$ at S_2 and the computation of $(r_3 \bowtie r_4)$ at S_4.

Heterogeneous Distributed Databases

- Many database applications require data from a variety of preexisting databases located in a heterogeneous collection of hardware and software platforms
- Data models may differ (hierarchical, relational , etc.)
- Transaction commit protocols may be incompatible
- Concurrency control may be based on different techniques (locking, timestamping, etc.)
- System-level details almost certainly are totally incompatible.
- A **multidatabase system** is a software layer on top of existing database systems, which is designed to manipulate information in heterogeneous databases
  - Creates an illusion of logical database integration without any physical database integration

Advantages

- Preservation of investment in existing
  - hardware
  - system software
  - Applications
- Local autonomy and administrative control
- Allows use of special-purpose DBMSs
- Step towards a unified homogeneous DBMS
  - Full integration into a homogeneous DBMS faces
    - Technical difficulties and cost of conversion
    - Organizational/political difficulties
      - Organizations do not want to give up control on their data
      - Local databases wish to retain a great deal of **autonomy**

Unified View of Data

- Agreement on a common data model
  - Typically the relational model
- Agreement on a common conceptual schema
  - Different names for same relation/attribute
  - Same relation/attribute name means different things
- Agreement on a single representation of shared data
  - E.g. data types, precision,
  - Character sets
    - ASCII vs EBCDIC
    - Sort order variations
- Agreement on units of measure
- Variations in names
  - E.g. Köln vs Cologne, Mumbai vs Bombay

Query Processing

- Several issues in query processing in a heterogeneous database
- Schema translation
  - Write a **wrapper** for each data source to translate data to a global schema
  - Wrappers must also translate updates on global schema to updates on local schema
- Limited query capabilities
  - Some data sources allow only restricted forms of selections
    - E.g. web forms, flat file data sources
  - Queries have to be broken up and processed partly at the source and partly at a different site
- Removal of duplicate information when sites have overlapping information
  - Decide which sites to execute query
- Global query optimization

Mediator Systems

- **Mediator** systems are systems that integrate multiple heterogeneous data sources by providing an integrated global view, and providing query facilities on global view
  - Unlike full fledged multidatabase systems, mediators generally do not bother about transaction processing
  - But the terms mediator and multidatabase are sometimes used interchangeably
  - The term **virtual database** is also used to refer to mediator/multidatabase systems

Transaction Management in Multidatabases

- **Local transactions** are executed by each local DBMS, outside of the MDBS system control.
- **Global transactions** are executed under multidatabase control.
- Local autonomy - local DBMSs cannot communicate directly to synchronize global transaction execution and the multidatabase has no control over local transaction execution.
  - local concurrency control scheme needed to ensure that DBMS's schedule is serializable
  - in case of locking, DBMS must be able to guard against local deadlocks.
  - need additional mechanisms to ensure global serializability

Local vs. Global Serializability

- The guarantee of local serializability is not sufficient to ensure global serializability.
  - As an illustration, consider two global transactions T1 and T2 , each of which accesses and updates two data items, A and B, located at sites S1 and S2 respectively.
  - It is possible to have a situation where, at site S1 , T2 follows T1 , whereas, at S2 , T1 follows T2, resulting in a nonserializable global schedule.
- If the local systems permit control of locking behavior and all systems follow two-phase locking
  - the multidatabase system can ensure that global transactions lock in a two-phase manner
  - the lock points of conflicting transactions would then define their global serialization order.