COP 4710: Database Systems
Spring 2006

CHAPTER 22 – Parallel and Distributed Database Systems – Part 3

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790
http://www.cs.ucf.edu/courses/cop4710/spr2006

School of Electrical Engineering and Computer Science
University of Central Florida

Query Optimization

- In a query involving a multi-site join and, possibly, a distributed database with replicated files, the distributed DBMS must decide where to access the data and how to proceed with the join. Three step process:

  1 Query decomposition - rewritten and simplified

  2 Data localization - query fragmented so that fragments reference data at only one site

  3 Global optimization -

    - Order in which to execute query fragments
    - Data movement between sites
    - Where parts of the query will be executed

Distributed Query Processing

- As we've seen in the previous section of notes, with distributed databases, the response to a query may require the DDBMS to assemble data from several different sites (remember though that location transparency will make the user unaware of this fact).

- A major decision for the DDBMS is how to process a query. How the query will be processed is affected primarily by two factors:

  1. How the user formulates the query (as we saw in the centralized case) and how it can be transformed by the DDBMS.

  2. Intelligence of the DDBMS in developing a sensible plan of execution (distributed optimization).

Distributed Query Processing – Example

- Consider the simplified version of our supplier/parts database as shown below:

  suppliers (s#, city) [located at site A, contains 10,000 tuples]

  parts (p#, color) [located at site B, contains 100,000 tuples]

  shipments (s#, p#, qty) [located at site A, contains 1,000,000 tuples]

Assumptions

- Each tuple is 100 bytes.
- There are exactly 10 red parts.
- The query is: List the supplier numbers for suppliers in Orlando who ship a red part.
- There are 100,000 tuples in the shipments relation that involve shipments from suppliers in Orlando.
- Computation time at any site is negligible compared to communication time.
- Network transfer rate is 10,000 bytes/sec.
- Access delay = 1 second (time to send a message – not a tuple from one site to another).
- T = total communication time = total access delay + (total data volume / data rate)

  = (# messages sent   x 1 sec/message) + (total # of bytes sent / 10,000)

Distributed Query Processing – Example (cont.)

Strategy #1

- Move entire parts relation to site A and process query at site A.
  - $T_1 = 1 + (100{,}000 \text{ x } 100)/10{,}000 \approx 1000$ sec = 16.7 minutes

Strategy #2

- Move supplier and shipment relations to site B and process the query at site B.
  - $T_2 = 2 + ((10{,}000 + 1{,}000{,}000) \text{ x } 100)/10{,}0000 = 10{,}100$ sec = 2.8 hours

Distributed Query Processing – Example (cont.)

Strategy #3

- Join suppliers and shipments relations at site A, select tuples from the join for which the city is Orlando, and then, for each of those tuples in turn, check site B to see if the indicated part is red. Each check requires 2 messages, a query, and a response. Transmission time for these messages is small compared to the access delay. There will be 100,000 tuples in the join for which the supplier is located in Orlando.
  – T_3 = (100,000 tuples to check) x (2) x (1 sec/message) = 200,000 sec ≈ 55 hours = 2.3 days

Strategy #4

- Select tuples from the parts relation at site B for which the color is red, and then, for each of these tuples in turn, check at site A to see if there exists a shipment of the part from an Orlando supplier. Again, each check requires two messages.
  – T_4 = (10 red parts) x (2 messages each) x (1 sec/message) = 20 sec

Distributed Query Processing – Example (cont.)

Strategy #5

- Join suppliers and shipments relations at site A, select tuples from the join for which the city is Orlando, and then, project only the s# and p# attributes and move this "qualified" relation to site B where the query processing will be completed.
  - T_5 = (1 + (100,000 tuples for Orlando) x (100 bytes/tuple)/10,000 bytes/second ≈ 1000 sec = 16.7 minutes

Strategy #6

- Select tuples from the parts relation at site B for which the color is red, then move this result to site A to complete the query processing.
  - T_4 = 1 + (10 red parts x (100 bytes/tuple) / 10,000 ≈ 1 sec

Distributed Query Processing – Example (cont.)

Summary

	Strategy	Time
1	Move parts table to site A, process query at site A.	16.7 minutes
2	Move suppliers and shipments tables to site B, process query at site B.	2.8 hours
3	Join suppliers and shipments at site A, check selected rows at site B.	2.3 days
4	Select red parts from parts tables at site B, for these tuples check at site A for a shipment of this part.	20 seconds
5	Join suppliers and parts at site A, move “qualified” rows to site B for processing.	16.7 minutes
6	Select red parts from parts table at site B, move these tuples to site A for processing.	≈1 second

Distributed Query Transformation

Horizontal fragmentation example

- Suppose we have the `shipments` table horizontally fragmented as follows:
  - shipments = SPJ1 ∪ SPJ2 where

    $$SPJ1 = \sigma_{(p\# = \text{'P1'})}(\text{shipments}) \text{ and } SPJ2 = \sigma_{(p\# \neq \text{'P1'})}(\text{shipments})$$

  - assume that SPJ1 is located at site1 and SPJ2 is located at site 2.
- A user at some site (assume its is neither site 1 or site 2) wants the answer to the query "list the supplier numbers for those suppliers who ship part P1" and issues the query expression: $\pi_{s\#}(\sigma_{(p\#=\text{'P1'})}(\text{shipments})$ to determine the results.
- Remember that the user is unaware of the fragmentation of the shipments relation.

Distributed Query Transformation (cont.)

Horizontal fragmentation example (cont.)

- Since shipments is defined as shipments = SPJ1 ∪ SPJ2 the query will be transformed into: $\pi_{s\#}(\sigma_{(p\#=\text{'P1'})}(SPJ1 \cup SPJ2))$.
- The query optimizer will initially transform the expression above into: $[\pi_{s\#}(\sigma_{(p\#=\text{'P1'})}(SPJ1)] \cup [\pi_{s\#}(\sigma_{(p\#=\text{'P1'})}(SPJ2)]$.
- Further optimization can be done since the system can determine that SPJ2 is defined as: $SPJ2 = \sigma_{(p\# \neq \text{'P1'})}(\text{shipments})$. Due to this definition, the sub-expression involving SPJ2 does not need to be evaluated as it will not contribute any values to the result set.
- Further since SPJ1 is defined as: $SPJ1 = \sigma_{(p\# = \text{'P1'})}(\text{shipments})$, the query can be further simplified to: $\pi_{s\#}(SPJ1)$.

Distributed Query Transformation (cont.)

Customer Name	Branch
Kristi	Oviedo
Debbie	Maitland
Michael	Longwood
Didi	Oviedo
Tawni	Oviedo

Initial table R

Horizontal fragments based on:

$\sigma_{(\text{Branch} = \text{'Oviedo'})}(R)$

Customer Name	Branch
Kristi	Oviedo
Didi	Oviedo
Tawni	Oviedo

Fragment #1

Customer Name	Branch
Debbie	Maitland
Michael	Longwood

Fragment #2

Consider queries such as:

(1) List customer names at branch in Oviedo.

(2) List customer names at branches not in Oviedo.

(3) List customer names at any branch.

Distributed Query Transformation

Vertical fragmentation example

- Suppose we have the `shipments` table horizontally fragmented as follows:
  - shipments = SPJ1 ∪ SPJ2 where

    SPJ1 = $\sigma_{(p\# = \text{'P1'})}$(shipments) and SPJ2 = $\sigma_{(p\# \neq \text{'P1'})}$(shipments)
  - assume that SPJ1 is located at site1 and SPJ2 is located at site 2.
- A user at some site (assume its is neither site 1 or site 2) wants the answer to the query "list the supplier numbers for those suppliers who ship part P1" and issues the query expression: $\pi_{s\#}(\sigma_{(p\#=\text{'P1'})}$(shipments) to determine the results.
- Remember that the user is unaware of the fragmentation of the shipments relation.

Distributed Query Transformation (cont.)

Vertical fragmentation example

Customer Name	Branch	Balance
Kristi	Oviedo	15,000
Debbie	Maitland	23,000
Michael	Longwood	4,000
Didi	Oviedo	50,000
Tawni	Oviedo	18,000

Initial table R

VF1: $\pi_{(name,\ branch)}(R)$

Customer Name	Branch
Kristi	Oviedo
Debbie	Maitland
Michael	Longwood
Didi	Oviedo
Tawni	Oviedo

VF2: $\pi_{(name,\ balance)}(R)$

Customer Name	Balance
Kristi	15,000
Debbie	23,000
Michael	4,000
Didi	50,000
Tawni	18,000

Query: List customer names in Oviedo with balances >= 15,000

Initial query expression: $\pi_{customer\ name}(\sigma_{(balance\ >=\ 15000\ and\ branc\ =\ 'Oviedo')}(R))$

Query will be transformed into:

$\pi_{customer\ name}[(\sigma_{(balance\ >=\ 15000)}(VF2)) \bowtie (\sigma_{(branch\ =\ 'Oviedo')}(VF1))]$

Semi Join Strategy

- In general, join operations are costly. This is especially true in a distributed environment where shipping large join tables around the network can be extremely costly.

- One technique that is commonly employed is the semi join (See Chapter 4 notes, pages 14-15).

- In a semi join, only the joining attribute is sent from one site to another, and then only the required rows are returned. If only a small percentage of the rows participate in the join, then the amount of data being transferred is minimized.

- $R1 \triangleright R2 \equiv \pi_{R1}(R1 \bowtie R2)$ (recall that $R1 \triangleright R2 \neq R2 \triangleright R1$)

Semi Join Strategy - Example

- Consider the following distributed database.

Site 1

Customer table

Customer_num	10 bytes
Customer_name	50 bytes
Zipcode	10 bytes
SIC	5 bytes

Current instance contains 10,000 rows

Site 2

Order table

Order_num	10 bytes
Customer_num	10 bytes
Order_Date	4 bytes
Order_amount	6 bytes

Current instance contains 400,000 rows

Semi Join Strategy – Example (cont.)

- Assume that a query originates at site 1 to display the Customer_name, SIC, and Order_date for all customers in a particular Zipcode range and an Order_amount above a specified value.
- Further assume that 10% of the customers fall into the particular zipcode range and 2% of the orders are above the specified value.
- Given these conditions, a semi join will work as follows:
  - A query is executed at site 1 to create a list of the Customer_num values in the desired Zipcode range. So, 1,000 rows satisfy the zipcode condition (since 10% of 10,000 = 1000) and each of these rows involves a 10-byte Customer_num field, so in total, 10,000 bytes will be sent from site 1 to site 2.

Semi Join Strategy – Example (cont.)

- A query is executed at site 2 to create a list of the Customer_num and Order_date values to be sent back to site 1 to compose the final result. If we assume roughly the same number of orders for each customer, then 40,000 rows of the order table will match with Customer_num values sent from site1. Assuming that any order is equally likely to be above the amount limit, then 800 rows (2% of 40,000) apply to this query. This means that 11,200 bytes (14 bytes/row x 800 rows) will be sent to site 1.

- The total amount of data transferred is only 21,200 bytes using the semi join strategy.

- The total data transferred that would result from simply sending the subset of each table needed to the other site would be:

Semi Join Strategy – Example (cont.)

- To send data from site 1 to site 2 requires sending the Customer_num, Customer_name, and SIC: total of 65 bytes/row for 1000 rows of the Customer table = 65,000 bytes from site 1 to site 2.
- To send data from site 2 to site 1 requires sending the Customer_num and Order_date: total of 14 bytes for 8000 rows of the Order table = 112,000 bytes.
- The semi join strategy required only 21,200 bytes to be transferred.