Distributed Query Processing

Saptarshi Pyne
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Jodhpur, Rajasthan, India 342030

CSL4030 Data Engineering Lecture 30
October 27th, 2023

What we discussed in the last class

- Data Warehousing
  - OLTP (online transaction processing)
  - OLAP (online analytic processing)

The key factor in centralized query processing

- The number of disk accesses

The key factors in distributed query processing

- The number of disk accesses
- The cost of data transmission over the network
- The possibility of having several sites processing parts of the query in parallel

Query processing with multiple replicas

Case study: (banking database)

SELECT * from accounts;

If (all sites have a complete replica of the ‘accounts’ table)

Choose a site with the lowest data transmission cost.
Access the table from that site.

Query processing with multiple shards (horizontal fragments)

Case study: (the SBI database)

SELECT * from accounts WHERE custID='C005';

Suppose, the 'accounts' table has been sharded across two servers: the Jodhpur branch and the Jaipur branch.

The customer may have accounts in both the branches.

Query processing with multiple shards (contd.)

As a result, the query will be transformed into two subqueries. (**Query transformation**)

Query:

SELECT * from accounts WHERE custID='C005';

Subqueries:

SELECT * from accounts_jdh WHERE custID='C005';

SELECT * from accounts_jai WHERE custID='C005';

Query processing with multiple shards (contd.)

The corresponding **relational-algebra expressions** are as follows.

accounts = (accounts_jdh U accounts_jai)

Therefore,

$\sigma_{custID='C005'}$ (accounts)

= $\sigma_{custID='C005'}$ (accounts_jdh U accounts_jai)

= $\sigma_{custID='C005'}$ (accounts_jdh) U $\sigma_{custID='C005'}$ (accounts_jai)

Join query processing

Query: (initiated at site S1)

accounts ⋈ depositor ⋈ branch

(‘⋈’ = natural join = join based on common attributes)

Table locations:

accounts in site S1

depositor in site S2

branch in site S3

Join query processing (contd.)

Strategy 1

S1: Retrieve copies of ‘depositor’ and ‘branch’ from S2 and S3, respectively. Then join all three tables locally.

Strategy 2

S2: Retrieve a copy of ‘accounts’.

S2: temp1 = (accounts ⋈ depositor).

S2: Transmit ‘temp1’ to S3.

S3: temp2 = (temp1 ⋈ branch).

S3: Transmit ‘temp2’ to S1.

Join query processing (contd.)

Strategy 1

S1: Retrieve copies of ‘depositor’ and ‘branch’ from S2 and S3, respectively. Then join all three tables locally.

Strategy 2

S2: Retrieve a copy of ‘accounts’.

S2: temp1 = (accounts ⋈ depositor).

S2: Transmit ‘temp1’ to S3.

S3: temp2 = (temp1 ⋈ branch).

S3: Transmit ‘temp2’ to S1.

Which one is the best strategy?

Comparative analysis of the two strategies

Analysis of Strategy 1

A natural join requires searching for the matching values of the common attributes of two tables. We can speed up the searching process if we have the index structures of the two tables.

The index structure of a table is stored where the master copy of the table is stored.

In Strategy 1, we are copying the ‘depositor’ and ‘branch’ tables to site S1. Therefore, we should also copy their index structures to S1. Otherwise, the join (accounts ⋈ depositor ⋈ branch) would be slower.

Alternatively, we can rebuild their index structures at S1. However, rebuilding index structures requires a large of disk accesses.

Comparative analysis of the two strategies (contd.)

Analysis of Strategy 2

Here, we are only copying the ‘accounts’ table to its non-master site. The other two tables are being joined at their master sites where their index structures are present.

On the other hand, we are transmitting large tables ‘temp1’ and ‘temp2’ over the network. Moreover, their transmissions are not parallelizable.

In Strategy 1, we are transmitting smaller tables ‘depositor’ and ‘branch’ over the network. At the same time, their transmissions are parallelizable.

Hence, Strategy 1 has lower **network cost** than that of Strategy 2. On the other hand, Strategy 2 has less **disk access cost** than that of Strategy 1. We should choose a strategy based on **what is more important to us**. For example, if we have a high-speed network and slower disks, we should choose Strategy 2.

Join strategies to exploit parallelism

Query: (initiated at site S1)

$r1 \bowtie r2 \bowtie r3 \bowtie r4$

Locations of the relations/tables:

r1 in site S1

r2 in site S2

r3 in site S3

r4 in site S4

Join strategies to exploit parallelism (contd.)

Query: (initiated at site S1)

r1 ⋈ r2 ⋈ r3 ⋈ r4

S1, S3: S1 brings a copy of r2. In parallel, S3 brings a copy of r4.

S1, S3: S1 performs temp1 = (r1 ⋈ r2). In parallel, S3 performs temp2 = (r3 ⋈ r4).

S3: Transmits ‘temp2’ to S1.

S1: result = (temp1 ⋈ temp2).

The semijoin strategy

Query: (initiated at site S1)
r1 ⋈ r2

Locations of the relations/tables:
r1 in site S1
r2 in site S2

We should apply the semijoin strategy if

- r2 has a large number of tuples and
- it is expected that only a small fraction of tuples in r2 will contribute to the join. In other words, a small number of tuples in r2 has matching values with r1 on common attributes.

The semijoin strategy (contd.)

Semijoin ($r1 \bowtie r2$) by selecting only the contributing tuples from r2

$= r1 \ltimes r2$

[A similar notation applies to the semijoin ($r1 \bowtie r2$) by selecting only the contributing tuples from r1

$= r1 \rtimes r2$]

The semijoin strategy (contd.)

$r1 \ltimes r2$

Suppose,

R1 is the set of attribute names of r1 and

R2 is the set of attribute names of r2.

The semijoin strategy (contd.)

$r1 \ltimes r2$

S1: temp1 = $\Pi_{R1 \cap R2}(r1)$.
S1: Transmit ‘temp1’ to S2.
S2: temp2 = temp1 $\bowtie$ r2.
S2: Transmit ‘temp2’ to S1.
S1: result = r1 $\bowtie$ temp2.

The semijoin strategy (contd.)

Is $(r1 \ltimes r2)$ faster than $(r1 \bowtie r2)$?

S1: temp1 = $\Pi_{R1 \cap R2}(r1)$.
S1: Transmit ‘temp1’ to S2. [Loss of time]
S2: temp2 = temp1 $\bowtie$ r2.
S2: Transmit ‘temp2’ to S1. [Gain in time]
S1: result = r1 $\bowtie$ temp2.

Yes, when the gain exceeds the loss.

References

- Section 19.7 ‘Distributed Query Processing’, A. SILBERSCHATZ, H.F. KORTH, S. SUDARSHAN (2011), Database System Concepts, McGraw Hill Publications, 6th Edition.

Thank you