DISTRIBUTED QUERY PROCESSING

Distributed Query Processing

- There are various steps that are followed for query processing.
- A distributed database query is processed in stages as follows:

1. **Query Mapping:**

- The input query on distributed data is specified formally using a query language.
- It is then translated into an algebraic query on global relations.
- This translation is done by referring to the global conceptual schema.
- This translation is largely identical to the one performed in a centralized DBMS.
- It is first normalized, analyzed for semantic errors, simplified, and finally restructured into an algebraic query.

Distributed Query Processing

2. **Localization:**

◦ This stage maps the distributed query on the global schema to separate queries on individual fragments using data distribution and replication information.

3. **Global Query Optimization:**

◦ Optimization consists of selecting a strategy from a list of candidates that is closest to optimal.

◦ A list of candidate queries can be obtained by permuting the ordering of operations within a fragment query generated by the previous stage.

◦ The total cost is a weighted combination of costs such as CPU cost, I/O costs, and communication costs.

4. **Local Query Optimization.**

◦ This stage is common to all sites in the DDB.

◦ The techniques are similar to those used in centralized systems.

The first three stages discussed above are performed at a central control site, while the last stage is performed locally.

Distributed Query Processing

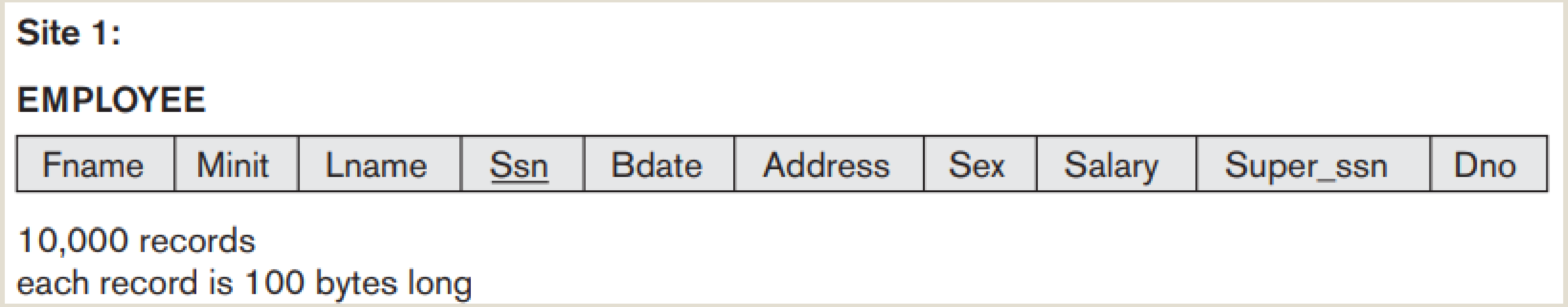
Data Transfer Costs of Distributed Query Processing

- In a distributed system, several additional factors further complicate query processing.
- The first is the cost of transferring data over the network.
- This data includes intermediate files that are transferred to other sites for further processing, as well as the final result files that may have to be transferred to the site where the query result is needed.
- These costs may not be very high if the sites are connected via a high-performance local area network, they become quite significant in other types of networks.
- DDBMS query optimization algorithms consider the goal of reducing the amount of data transfer as an optimization criterion in choosing a distributed query execution strategy.

Distributed Query Processing

Example:

- Suppose that the EMPLOYEE and DEPARTMENT relations are distributed at two sites.
- Suppose that each record in the query result is 40 bytes long.

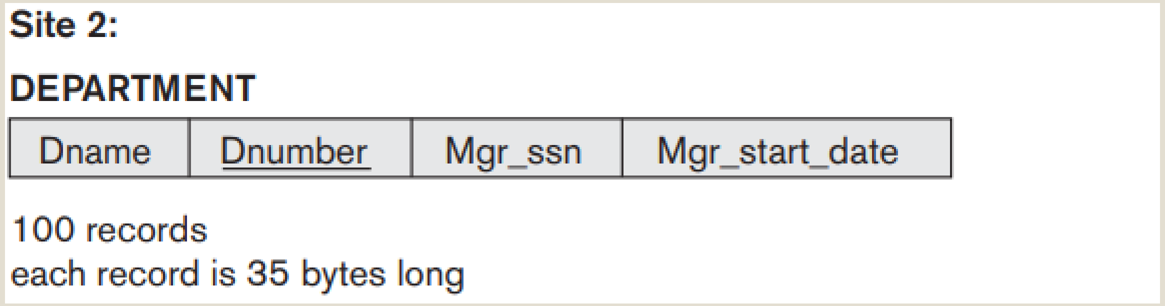


Site 1:

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

10,000 records
each record is 100 bytes long



Site 2:

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

100 records
each record is 35 bytes long

Distributed Query Processing

Example:

- The query is submitted at a distinct site 3, which is called the result site because the query result is needed there.
- Neither the EMPLOYEE nor the DEPARTMENT relations reside at site 3.
- There are three simple strategies for executing this distributed query:

1. First:

- Transfer both the EMPLOYEE and the DEPARTMENT relations to the result site, and perform the join at site 3.
- In this case, a total of 1,000,000 + 3,500 = 1,003,500 bytes must be transferred.

Distributed Query Processing

Example:

2. Second:

- Transfer the EMPLOYEE relation to site 2, execute the join at site 2, and send the result to site 3.
- Transfer the EMPLOYEE relation to site 2, execute the join at site 2, and send the result to site 3.
- The size of the query result is 40 * 10,000 = 400,000 bytes, so 400,000 + 1,000,000 = 1,400,000 bytes must be transferred.

3. Third:

- Transfer the DEPARTMENT relation to site 1, execute the join at site 1, and send the result to site 3.
- In this case, 400,000 + 3,500 = 403,500 bytes must be transferred.

*If minimizing the amount of data transfer is our optimization criterion, we should choose strategy 3.

Distributed Query Processing

- Now consider another query Q:
- For each department, retrieve the department name and the name of the department manager.
- This can be stated as follows in the relational algebra:

$$Q': \pi_{\text{Fname,Lname,Dname}}(\text{DEPARTMENT} \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE})$$

- Suppose that the query is submitted at site 3.
- The same three strategies for executing query Q apply to Q', except that the result of Q' includes only 100 records, assuming that each department has a manager:

1. First:

- Transfer both the EMPLOYEE and the DEPARTMENT relations to the result site, and perform the join at site 3.
- In this case, a total of 1,000,000 + 3,500 = 1,003,500 bytes must be transferred.

Distributed Query Processing

2. Second:

- Transfer the EMPLOYEE relation to site 2, execute the join at site 2, and send the result to site 3.
- The size of the query result is 40 * 100 = 4,000 bytes, so 4,000 + 1,000,000 = 1,004,000 bytes must be transferred.

3. Third:

- Transfer the DEPARTMENT relation to site 1, execute the join at site 1, and send the result to site 3.
- In this case, 4,000 + 3,500 = 7,500 bytes must be transferred.

Distributed Query Processing Semi Join

- The idea behind distributed query processing using the semijoin operation is to reduce the number of tuples in a relation before transferring it to another site.
- Intuitively, the idea is to send the joining column of one relation R to the site where the other relation S is located; this column is then joined with S.
- Following that, the join attributes, along with the attributes required in the result, are projected out and shipped back to the original site and joined with R.
- Hence, only the joining column of R is transferred in one direction, and a subset of S with no extraneous tuples or attributes is transferred in the other direction.
- If only a small fraction of the tuples in S participate in the join, this can be quite an efficient solution to minimizing data transfer.

Distributed Query Processing Semi Join

◦ Consider the following strategy for executing Q or Q:

1. Project the join attributes of DEPARTMENT at site 2, and transfer them to site 1. For Q, we transfer F = $\pi_{Dnumber}$(DEPARTMENT), whose size is 4 * 100 = 400 bytes, whereas, for Q', we transfer F'= π_{Mgr_ssn}(DEPARTMENT), whose size is 9 * 100 = 900 bytes.
2. Join the transferred file with the EMPLOYEE relation at site 1, and transfer the required attributes from the resulting file to site 2. For Q, we transfer R = $\pi_{Dno, Fname, Lname}$(F $\bowtie_{Dnumber=Dno}$ EMPLOYEE), whose size is 34 * 10,000 = 340,000 bytes, whereas, for Q', we transfer R' = $\pi_{Mgr\_ssn, Fname, Lname}$ (F' $\bowtie_{Mgr\_ssn=Ssn}$ EMPLOYEE), whose size is 39 * 100 = 3,900 bytes.

Distributed Query Processing Semi Join

3. Execute the query by joining the transferred file R or Rwith DEPARTMENT, and present the result to the user at site

- Using this strategy, we transfer 340,400 bytes for Q and 4,800 bytes for Q'.
- We limited the EMPLOYEE attributes and tuples transmitted to site 2 in step 2 to only those that will actually be joined with a DEPARTMENT tuple in step 3.
- For query Q, this turned out to include all EMPLOYEE tuples, so little improvement was achieved.
- However, for Q' only 100 out of the 10,000 EMPLOYEE tuples were needed.

Distributed Query Processing Semi Join

- The semijoin operation was devised to formalize this strategy.
- A semijoin operation $R \ltimes_{A=B} S$, where A and B are domain-compatible attributes of R and S, respectively, produces the same result as the relational algebra expression $\pi_R(R \bowtie_{A=B} S)$.
- In a distributed environment where R and S reside at different sites, the semijoin is typically implemented by first transferring $F = \pi_B (S)$ to the site where R resides and then joining F with R.