Lecture 20
Introduction to Distributed Databases

Chapter 25.1-25.4: Elmasri & Navathe, 5ED
Chapter 22.6-22-10: Ramakrishnan & Gehrke, 3ED

Demetris Zeinalipour
http://www.cs.ucy.ac.cy/~dzeina/courses/epl446
Lecture Outline
(Introduction to Distributed Databases)

- **Introduction** to Distributed Databases
- **Types** of Distributed Databases
  - Homogeneous, Heterogeneous (Federated, MultiDBs)
- Distributed Databases **Architectures**
  - Client Server, Collaboration Server, Middleware
- **Data Fragmentation & Replication**
  - Horizontal, Vertical and Mixed Fragmentation.
  - Synchronous vs. Asynchronous
- Distributed **Catalog Management**
- Distributed **Query Processing**
  - Centralized, Ship-to-one-site, Semi-join, Bloom-join & Bloom Filters
Distributed Catalog Management
(Κατανεμημένη Διαχείριση Καταλόγου)

- Catalog in a distributed DBMS keeps track of how data is **distributed (fragmented) across** sites.
  - These are in addition to **schema, authorization and statistical information** these catalogs usually contain.

- Must be able to **name each replica** of each **fragment**.
  - To preserve local autonomy each object is named according to the following:
    - `<local-name, birth-site>`

- **Site (Local) Catalog**: Describes **all objects (fragments, replicas) at a site** + Keeps track of **replicas of relations created** at this site.
  - To find a relation, **look up its birth-site** catalog.
    - Birth-site never changes, even if relation is moved.
Query Processing in Distributed Databases
(Επεξεργασία Επερωτήσεων σε Κατανεμημένες ΒΔ)

• The Problem:
  – Transferring data (files and results) over the network is an expensive operation.
  – Additionally, many queries can be localized e.g., SELECT * FROM Employees WHERE address=“New York” (no need to ask all distributed sites)
  – Consequently, some query optimization is necessary

• Example Scenario
  – Employee at Site1
    • 10,000 rows \(\times\) Row_size = 100 bytes yields Table size = \(10^6\) bytes ~ 1MB
      
      | Fname | Minit | Lname | SSN | Bdate | Address | Sex | Salary | Superssn | Dno |
      |-------|-------|-------|-----|-------|---------|-----|--------|----------|-----|
  – Department at Site2
    • 100 rows \(\times\) Row_size = 35 bytes yields Table size = \(3,500\) bytes ~3,5KB
      
      | Dname | Dnumber | Mgrssn | Mgrstartdate |
      |-------|---------|--------|-------------|
  – Query (Q1): For each employee, retrieve employee name (Fname, Lname) and department name (Dname) where the employee works.
  – i.e., \(\Pi_{Fname,Lname,Dname} (Employee \otimes Dno = Dnumber \text{ Department})\)
• **Example Scenario (contd.)**
  
  • The query is submitted at some *Query Processor (QP) at Site3*
  
  • The *result tuple* is *40 bytes long* (i.e., due to the three fields *Fname, Lname, Dname*).
  
  – The result to this query will have *10,000 tuples* (i.e., *10,000 employees*).
    
    • assuming that each of the *10,000 employees* is related to a *department.*
Query Processing in Distributed Databases

(Επεξεργασία Επερωτήσεων σε Κατανεμημένες ΒΔ)

- **Execution Strategies** (Optimization: minimizing data transfer):
  1. **Centralized:** Transfer **Employee** and **Department** to site3.
     - Total transfer bytes = 1,000,000 + 3,500 = 1,003,500 bytes.
  2. **Ship-to-Site2:** i) Transfer **Employee** to site 2, ii) execute **join** at site 2 and iii) send the result to site 3.
     - Total transfer size = 1,000,000 + **400,000** = 1,400,000 bytes.
     - Query result @ Site2 = 40 (result size) * 10,000 (employees) = 400,000B
  3. **Ship-to-Site1:** i) Transfer **Department** relation to site 1, ii) execute the join at site 1, iii) and send the result to site 3.
     - Total bytes transferred = 3,500 + **400,000** = 403,500 bytes.
     - Query result @ Site1 = 40 (result size) * 10,000 (employees) = 400,000B

**Preferred Approach:** Approach 3!
Let us now consider Query 2:

- Query 2: For each department, retrieve the department name and the name of the department manager
- Query 2: $\Pi_{\text{Fname, Lname, Dname}} (\text{Employee } \bowtie_{\text{Mgrssn} = \text{SSN}} \text{ Department})$

Let us assume that the result of this query will have 100 tuples,

assuming that every department has a manager.

The length of each result tuple is still 40 bytes (i.e., due to the three fields FName, LName, Dname)
Query Processing in Distributed Databases
(Επεξεργασία Επερωτήσεων σε Κατανεμημένες ΒΔ)

• **Execution Strategies** (Optimization: data transfer):

  1. **Centralized**: Transfer Employee and Department to the result site and perform the join at site 3.
     - Total bytes transferred = 1,000,000 + 3,500 = **1,003,500 bytes**.

  2. **Ship-to-Site2**: Transfer Employee to site 2, execute join at site 2 and send the result to site 3.
     - Total transfer size = 1,000,000 + 4,000 = **1,004,000 bytes**.
     - Query result size at Site2= 100 result tuples * 40 tuple-length = 4,000 bytes.

  3. **Ship-to-Site1**: Transfer Department relation to site 1, execute join at site 1 and send the result to site 3.
     - Total transfer size = 3,500 + 4,000 = **7,500 bytes**.

**Preferred Approach: Approach 3!**

---

EPL446: Advanced Database Systems - Demetris Zeinalipour (University of Cyprus)
Problems with Ship-to-One-Site?

- Some attributes (or tuples) in Department relation **might** not be necessary for performing the join (e.g.,).
  - e.g., attribute Mgrstartdate for query
    - \( \pi_{\text{Fname, Lname, Dname}} (\text{Employee} \times_{\text{Mgrssn = SSN}} \text{Department}) \)
  - e.g., tuples with Address != “New York” for query
    - \( \sigma_{\text{Address = “New York”}} (\text{Employee} \times_{\text{DNumber = DNO}} \text{Department}) \)
- Consequently, we **transfer** much data that will **never** be necessary during the join.
- There are **two basic solutions** to avoid shipping unnecessary data:
  - **Semi-join (Ημι-Σύζευξη)**, send join column from one site to other site, then use this column to reduce the size of the “Ship-Relation”
  - **Bloom-join (Σύζευξη Bloom)**, like Semi-Join with the difference that we send a compact representation of the join column (rather than the column itself).
    - This compact representation can be utilized for member queries.
Semi-Join Algorithm
(Αλγόριθμος Ημι-Σύζευξης)

- **Semi-join (Ημι-Σύζευξη):**
  - Objective is to reduce the number of tuples in a relation before transferring it to another site.
  - **Semi-Join Basic Idea:** send join column from one site to other site, then use this column to reduce the size of the “Ship-Relation”

- **Example execution of Q2 (π_{Fname,Lname,Dname} (Employee \Join_{SSN=MgrSSN} Department))**
  1. **@DPT:** Project the join attribute MgrSSN of Department and **transfer it to Employees** (i.e., smaller than shipping SSN of Employees).
     - For Q2 F’=π_{MgrSSN} (Department), whose size is 9 (size of SSN) * 100 (managers) = 900 bytes are transferred.
  2. **@EMP:** Join F’ with the Employee relation and **then transfer the results Department** [ F’”=π_{MgrSSN,Fname,Lname} (F’ \Join_{MgrSSN=SSN} Employee) ]
     - For Q2, 39 (size of MgrSSN,Fname,Lname) * 100 (managers) = 3900 bytes are transferred.
  3. **@DPT:** Join F’” with the Department relation and return results to user [ π_{Fname,Lname,Dname} (F’” \Join_{SSN=MgrSSN} Department) ].
     - 3,900 bytes (ship “Fname, Lname, Dname” to Query Processor)
     - Total Cost: 900 + 3900 + 3900 = 8,700 bytes are transferred (better than previous 7,800 bytes!)
Bloom-Join Algorithm
(Αλγόριθμος Σύζευξης Bloom)

- **Bloom-Join (Σύζευξη Bloom):**
  - Reduce the number of tuples in a relation even more than the basic Semi-Join idea.
  - **Bloom-Join Basic Idea:** send compact representation of join column from one site to other site, then use this info to reduce the size of the “Ship-Relation”

- The Bloom Join algorithm is quite similar to the Semi-join with the difference that the former utilizes a Bit-vector (Δυαδικό Διάνυσμα) with K positions that can be utilized to perform membership queries (ερωτήσεις συμμετοχής)
  - Π.χ., Έστω ότι το πεδίο SSN της σχέσης Employees περιέχει τις ακόλουθες τιμές \{1134,4544,5342,22453,66433\}
  - Εάν οι υπάλληλοι 4544 και 66433 είναι Managers, τότε αυτό μπορεί να αναπαρασταθεί από το διάνυσμα $V_{SSN}$=[0,1,0,0,1] (δηλ., 0 όπου δεν ισχύει και 1 όπου ισχύει η ιδιότητα Manager)
  - Στην πρώτη φάση του Bloom Join, το $V_{SSN}$ μπορεί να αποσταλεί με μόνο 1 byte αντί 5*4bytes = 20 bytes τα οποία θα απαιτούσε ο αλγόριθμος του Semi-Join.
Bloom-Join Algorithm
(Αλγόριθμος Σύζευξης Bloom)

• Example execution of Q2 with Bloom Join

\[
(\pi_{\text{Fname}, \text{Lname}, \text{Dname}} (\text{Employee} \bowtie_{\text{SSN}=\text{Mgrssn}} \text{Department}))
\]

1. **@DPT: Create Bloom Filter on the join attribute** MgrSSN of Department and **transfer it to Employees**
   - For Q2 \( F' = \text{hash}(\pi_{\text{MgrSSN}} \text{(Department)}) \), whose size is 100 bits (managers) \( \sim 13 \) bytes

2. **@EMP: Utilize** \( F' \) to construct the join with the Employee relation and **then transfer the results Department** \( [ F''=\pi_{\text{MgrSSN}, \text{Fname}, \text{Lname}} (F' \bowtie_{\text{MgrSSN}=\text{SSN}} \text{Employee}) ] \) (i.e., use \( F' \) for the lookup)
   - For Q2, 39 (size of MgrSSN,Fname, Lname) * 100 (managers) = 3900 bytes

3. **@DPT: Join** \( F'' \) with the Department relation and return results to user \( [ \pi_{\text{Fname}, \text{Lname}, \text{Dname}} (F'' \bowtie_{\text{SSN}=\text{MgrSSN}} \text{Department}) ] \)
   - 3,900 bytes (ship “Fname, Lname, Dname” to Query Processor)
   - Total Cost: 13 + 3900 + 3900 = 7,813 bytes

---

19-12

EPL446: Advanced Database Systems - Demetris Zeinalipour (University of Cyprus)
Bloom-Join Algorithm
(Αλγόριθμος Σύζευξης Bloom)

• If the Universe is very large (e.g., 1M Employees) then the bit vector might become quite large ~125KBytes for 1M Employees!

• A way to improve the basic Bloom Filter idea (deployed in the first phase of the Bloom-Join Algorithm) is as follows:
  – Use a Vector \( V \) of \( m \)-bits (e.g., \( m=1,000 \))
  – Utilize \( k \) different hash functions \( K1, K2, \ldots, Kk \) and flip each hash every tuple in \( D=\{d1,d2,\ldots,dn\} \) using all \( k \) hash functions (flipping the respective bit on)
  – To identify if \( dx \) is part of the collection we utilize all \( k \) hash functions
    • if \( (h1(dx) \text{ and } h2(dx) \text{ and } \ldots \text{ and } hk(dx) == TRUE) \) then
      – Object might be part of the collection (False Positives are possible)
    else
      – Object is certainly NOT part of the collection (False Negatives are not possible!)