CIS 700/005
Networking Meets Databases

Boon Thau Loo
Spring 2007
Lecture 2

Announcements

- Reminder:
  - Introduction email (year, background, research interest, advisor, audit/enroll)
  - Office hours: Wed 3-4 pm (605 Levine)
  - http://www.cis.upenn.edu/~boonloo/cis700-sp07/ideas/ideas.html
  - All slides will be online (within UPenn and Drexel)

Last Week

- Review of Databases
  - SQL, query plan, relational algebra, architecture of a database
  - Reference for students with systems background:
    - http://redbook.cs.berkeley.edu

Today

- Overview of Distributed Databases
- Overview of Parallel Databases

Why is this relevant to CIS 700?

- Understand the motivation and design of distributed and parallel databases
- Relate to the design choices in the rest of the course? For example,
  - Is PIER/P2/TinyDB more similar compared to a parallel or distributed database?
  - What techniques carry over?
  - What are we gaining/sacrificing as we scale up to millions, or scale down to motes?

Distributed Databases

- Background textbooks:
  - Ramakrishnan and Gehrke, Database Management Systems, 3rd edition, Chapter 22
- Papers:
  - C. Mohan, B. Lindsay, and R. Obermarck, Transaction Management in the R* Distributed Database Management System. ACM Transactions On Database Systems 11(4), 1986
- Great lecture notes:
  - http://www.stanford.edu/class/cs347/
Distributed DBMS

- Important: many forms and definitions
- Our definition: “shared nothing” infrastructure
  - Multiple machines connected with a network

Reasons for a Distributed DBMS

- Scalability (eg: Amazon, eBay, Google)
  - Many small servers cheaper than large mainframe
  - Need to scale incrementally
- Inherent distribution
  - Large organizations have data at multiple locations (different offices) -> original motivation
  - Web-based and Internet-based applications
  - Different types of data in different DBMSs

Ideals of a Distributed DBMS

- Recall data independence: Users write SQL, do not worry about how data is physically stored.
- Ideal:
  - Distributed data independence:
  - Location transparency
  - Fragmentation transparency
  - Performance transparency
    - Distributed query optimizer ensures good performance no matter where query is submitted
    - Distributed transaction atomicity

Distributing Data

- Fragmentation:
  - Horizontal
  - Vertical: Lossless-join, TIDs
- Replication:
  - Gives increased availability
  - Faster query evaluation
  - Synchronous vs Asynchronous
    - Vary in how current copies are

Distributed queries

```
SELECT AVG(S.age) FROM Sailors S
WHERE S.rating > 3
AND S.rating < 7
```

Sailors(SID, name, rating, age)

- Horizontally fragmentation: Tuples with rating < 5 at site A, >= 5 at site B
- If WHERE contains just rating > 6, run the query only at site B
- Vertical fragmentation:
  - TableA(SID, rating), TableB(SID, name, age)
  - Must reconstruct relation by join on SID, then evaluate the query

Review: Query Evaluation

```
SELECT Name FROM Supplier S, Suppliers S1
WHERE S.sno = S1.sno AND S.sity = “Seattle”
```

Steps of query evaluation

| Query Parser | (On-the-fly) |
| Query Rewrite | (On-the-fly) |
| Optimizer | (Nested loop) |
| Executor | Supplier (File scan) |
| | Supplies (File scan) |
**Review:**

**Query Optimization**

- Enumerate alternative plans:
  - Many possible equivalence trees, e.g., join order, pushing selections/projections down the query plan
  - Many implementations of each operator
- Estimate cost of each plan:
  - Compute the number of I/Os and CPU utilization
  - Based on statistics
- Choose plan with the lowest cost

---

**Distributed Query Optimization**

- Search space is larger: must also select sites
- Minimize resource utilization:
  - I/O, CPU and communication costs
- Could also minimize response time at the expense of communication costs:
  - Least cost plan ≠ Fastest plan

---

**Example:**

**Distributed Joins**

- Sailors(SID, name, rating, age)
- Reserves(RID, S ID, boatID)

SELECT name, boatID
FROM Sailors S, Reserves R
WHERE S.SID = R.SID

---

**Example:**

**Distributed Joins**

- Option 1: Fetch as needed
  - For every Sailor tuple @ London, fetch matching Reserves tuple @ Paris.
  - Or vice versa
  - Ship back to query node
- Option 2: Ship entire table
  - E.g., send the entire Reserves table to London
  - Or vice versa
  - Ship back to query node
- Cost-based optimization: which table to ship? If results is large, ship both tables to query node?

---

**Join Optimizations**

```
SELECT name, boatID
FROM Sailors S, Reserves R
WHERE S.SID = R.SID
```

- Semi-joins
  - Let’s just ship Sailors.SID from London to Paris
  - Find all matches for Reserves at Paris.
  - Ship matching Reserves back to London to complete join
- Bloom-joins
  - Same idea, but use a bit-vector (bloom filter)
  - Some false positives
- Tradeoffs: Latency and bandwidth, additional projection

---

**Updates to Distributed DBMS**

- Same concerns as other replication systems
- Replication:
  - Synchronous vs Asynchronous
  - Primary site replication vs peer-to-peer replication
- Distributed transactions:
  - Single unit of action (“All or never”):
    - Insert 11 into tableA at site A,
    - Delete 12 into tableB at site B,
    - Insert 13 into tableC at site C
  - Distributed locking – how to detect deadlocks?
    - Global wait for graph (not very practical)
  - Timeouts, abort least costly local transaction
  - Expensive “two-phase” commit protocol with a single co-ordinator and multiple co-ordinates
Limitations and Challenges

- Autonomy: different administrative domains
  - Cannot always assume full cooperation
  - Do not require distributed transactions
- Heterogeneity:
  - Different capabilities at different location
  - Different data types, different semantics
- Large-scale
  - Internet-scale query processor

Mariposa – Federated Databases
http://mariposa.cs.berkeley.edu/

- Goal is to get rid of the single-administrative-domain assumption:
  - Dynamic data allocation
  - Multiple administrative structures
  - Heterogeneity of nodes
- Interesting idea based on economic models
  - Processing sites buy and sell data and query processing services.
  - Sites declare their local costs for a query based on:
    - Estimates of resource consumption
    - Runtime constraints (e.g., current system load)
    - Relationships with competition sites

What happened to Mariposa?

- Cohera -> PeopleSoft -> Oracle
- From the Redbook:
  - "The Mariposa system was commercialized as Cohera (later bought by PeopleSoft) and was demonstrated to work across administrative domains in fields"
  - "...flexibility and efficiency of its computation economy ideas have yet to be significantly tested"
  - "...unclear whether corporate IT is ready for significant investments in federated query processing."
  - "It is possible that we will see ideas from Mariposa re-emerge in the peer-to-peer space, where there is significant grassroots interest."

Parallel Databases

- Background:
  - Ramakrishnan and Gehrke, Database Management Systems, Chapter 22

Different Goals

- Performance:
  - Loading data, building indexes, executing queries
  - Data is distributed within single site
  - Distributed solely for performance
Parallel Data Management

- Parallelism is natural to DBMS processing:
  - Pipelined parallelism: many machines, each doing one step in a multi-step process
  - Partition parallelism: many machines doing the same thing to different pieces of the data

Architecture Issue: Shared What?

Different Types of DMBS //-ism

- Intra-operator parallelism
  - Get all machines to compute a given operation (scan, sort join)
- Inter-operator parallelism
  - Each operator runs at a different site
- Intra-query parallelism
  - Different queries run at different sites

Data Partitioning

Parallel Sorting

- Main idea:
  - Scan in parallel
  - Range re-partition
  - At each receiving node, local sorting
  - Problem: Skew!
  - Solution: “sample” the data at start to determine partition points

DBMS: The // Success Story

- DBMS are the most successful application of parallelism
  - Every major DBMS vendor has some // product
  - Key concepts are modified and reused in all major search engines
- Reasons for success:
  - Natural pipelining
  - Partitioned parallelism
  - Inexpensive hardware
  - Users/app-programmers don’t have to think in parallelism
Parallel Hash Join

- Given two tables A and B partitioned across N machines
- A(\text{aid}, \text{jid}), B(\text{bid}, \text{joinKey})
- How can we join them?
  - Hash-based re-partitioning based on joinKey

Search Engine Infrastructures

- Later this semester, if time permits…
- Search engine infrastructures:
  - Google and Inktomi Search Engines
  - BigTable,
  - MapReduce

Next lecture

- Next Tuesday:
  - Chord (SIGCOMM ’01)
  - Distributed Hash Tables (CACM ’03)
- Due on Tuesday noon:
  - Paper summaries