Database Management Systems

Winter 2003

CMPUT 391: Parallel & Distributed Databases

Dr. Osmar R. Zaïane

University of Alberta

Chapter 22 of Textbook

Course Content

- Introduction
- Database Design Theory
- Query Processing and Optimisation
- Concurrency Control
- Data Base Recovery and Security
- Object-Oriented Databases
- Inverted Index for IR
- Spatial Data Management
- XML and Databases
- Data Warehousing
- Data Mining
- **Parallel and Distributed Databases**

Objectives of Lecture 12

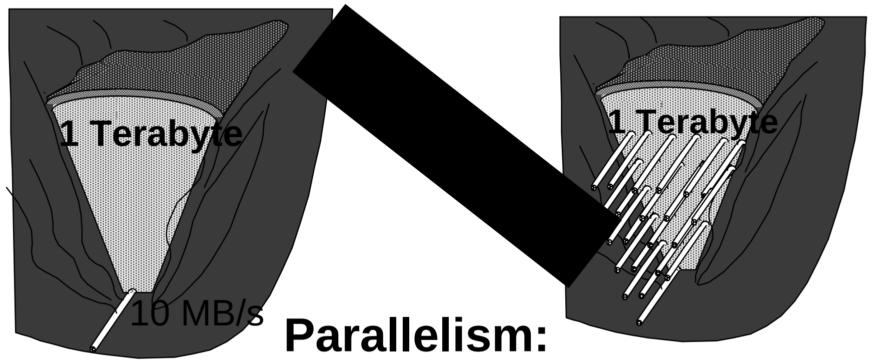
Parallel and Distributed Databases

- Get a general idea about what parallel and Distributed databases are
- Get an overview of what can be parallelized in DMBS (Query, Operations,Updating)
- Get acquainted with the existing architectures for parallel databases

Parallel & Distributed Databases

- Motivations and Architecture of Parallel Databases
- Parallel Query Evaluation and Optimization
- Distributed Databases & DBMS Architectures
- Storing Data in a Distributed DBMS
- Distributed Queries Processing
- Updating Distributed Data
- Distributed Transactions

Why Parallel Access To Data?



Parallelism:
divide a big problem into many smaller ones to be solved in parallel.

Why Parallel Access To Data?

Motivations :

- ➢Performance
- ➢Increased Availability
- ➢Distributed Access to Data
- ➢Analysis of Distributed data

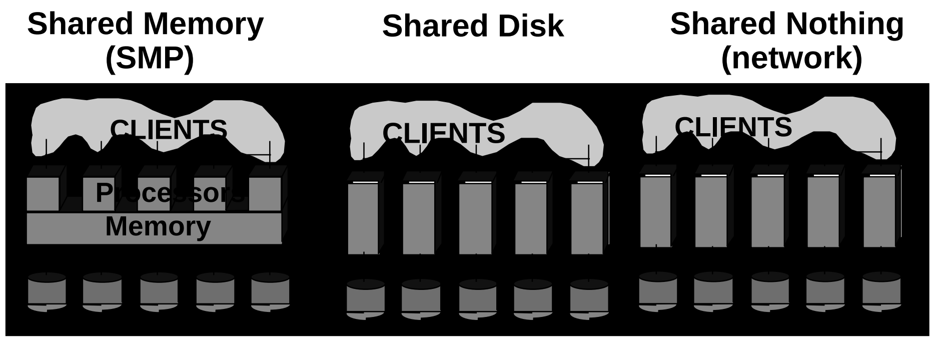
Why Parallel Access To Data?

What is a Parallel Database System ?

Is the one that seeks to improve performance through parallel implementations of various operations such as :

Loading data, Building indexes & evaluations of queries. Where the data are stored either in distributed fashion or centralized.

Architecture of Parallel Databases

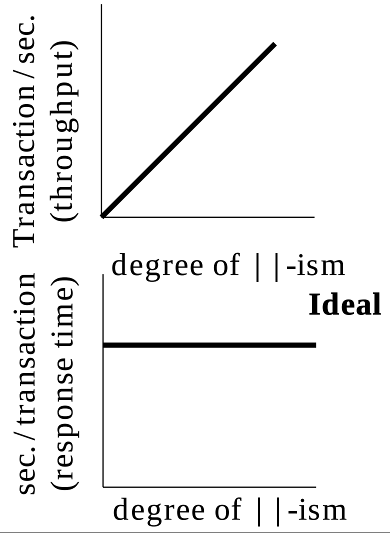


Easy to program
Expensive to build
Difficult to scaleup

Hard to program
Cheap to build
Easy to scaleup

Architecture of Parallel Databases

- Speed-Up
  - More resources means proportionally less time for given amount of data.
- Scale-Up
  - If resources increased in proportion to increase in data size, time is constant.



Parallel & Distributed Databases

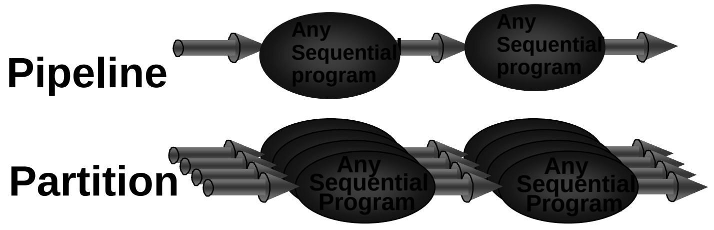
- Motivations and Architecture of Parallel Databases
- Parallel Query Evaluation and Optimization
- Distributed Databases & DBMS Architectures
- Storing Data in a Distributed DBMS
- Distributed Queries Processing
- Updating Distributed Data
- Distributed Transactions

Parallel Query Evaluation

Parallelism of Queries can be done using :

–*Pipeline parallelism:* many machines each doing one step in a multi-step process.

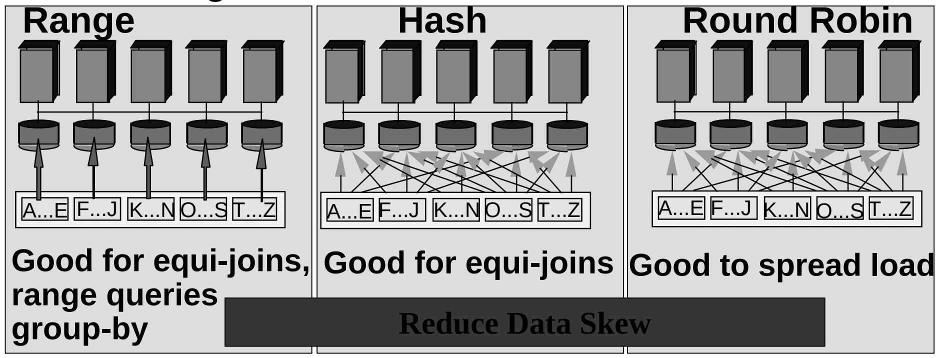
–*Partition parallelism:* many machines doing the same thing to different pieces of data.



outputs split N ways, inputs merge M ways

Parallel Query Evaluation

Partitioning a table:


Range
Hash
Round Robin
A...E F...J K...N O...S T...Z
A...E F...J K...N O...S T...Z
A...E F...J K...N O...S T...Z
Good for equi-joins, range queries group-by
Good for equi-joins
Good to spread load
Reduce Data Skew

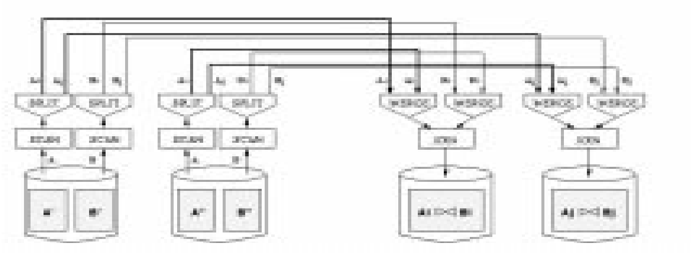
Shared disk and memory less sensitive to partitioning,
Shared nothing benefits from "good" partitioning

Parallelizing individual operations

❖Bulk Loading and Scanning : Pages can be read in parallel while scanning the relations

❖Sorting : Each Processor sorts its local portion

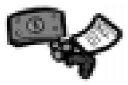
❖Joins : Join the sub results into the final one (many ways)



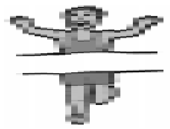
Dataflow Network for parallel Join

Parallel Query Optimization

Issues to be considered



Cost : Optimizer should estimate operation costs.



Speed : The fastest answers may not be the cheapest

Parallel & Distributed Databases

- Motivations and Architecture of Parallel Databases
- Parallel Query Evaluation and Optimization
- Distributed Databases & DBMS Architectures
- Storing Data in a Distributed DBMS
- Distributed Queries Processing
- Updating Distributed Data
- Distributed Transactions

Distributed Databases

- Data is stored at several sites, each managed by a DBMS that can run independently.
- Distributed Data Independence: Users should not have to know where data is located (extends Physical and Logical Data Independence principles).
- Distributed Transaction Atomicity: Users should be able to write transactions accessing multiple sites just like local transactions.

Distributed Databases

➢Users have to be aware of where data is located, i.e., Distributed Data Independence and Distributed Transaction Atomicity are not supported.

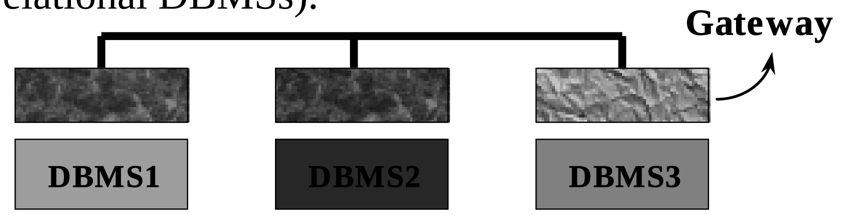
➢These properties are hard to support efficiently.

➢For globally distributed sites, these properties may not even be desirable due to administrative overheads of making location of data transparent.

Distributed Databases

Types

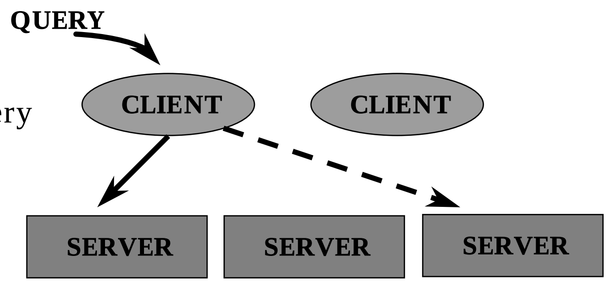
- Homogeneous: Every site runs same type of DBMS.
- Heterogeneous: Different sites run different DBMSs (different RDBMSs or even non-relational DBMSs).



Distributed DBMS Architectures

- Client-Server

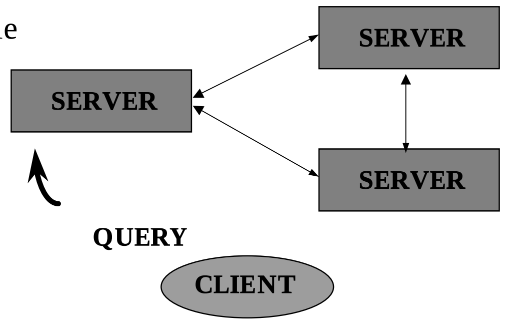
Client ships query to single site. All query processing at server.
- *Thin* vs. *fat* clients.
- Set-oriented communication, client side caching.



Distributed DBMS Architectures

Collaborating-Server

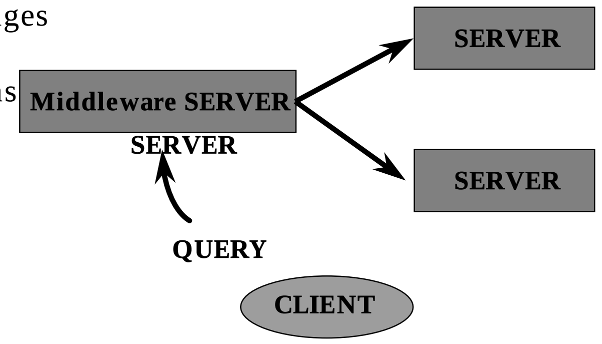
Query can span multiple sites.



Distributed DBMS Architectures

Middleware System

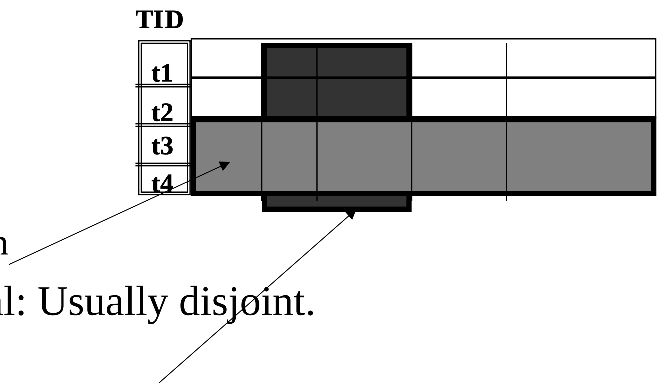
One Server manages queries and transactions spans multiple servers



Parallel & Distributed Databases

- Motivations and Architecture of Parallel Databases
- Parallel Query Evaluation and Optimization
- Distributed Databases & DBMS Architectures
- Storing Data in a Distributed DBMS
- Distributed Queries Processing
- Updating Distributed Data
- Distributed Transactions

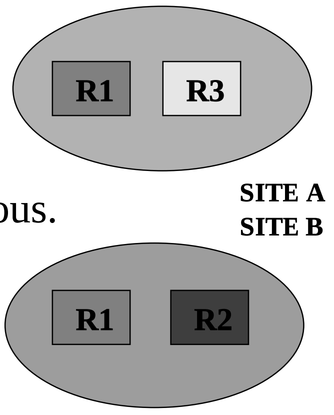
Storing Data in a Distributed DBMS



- Fragmentation
  - Horizontal: Usually disjoint.
  - Vertical: Lossless-join; tids.

Storing Data in a Distributed DBMS

•Replication
- –Gives increased availability.
- –Faster query evaluation.
- –Synchronous vs. Asynchronous.
  - •Vary in how current copies are.



Distributed Catalog Management

- Must keep track of how data is distributed across sites.
- Must be able to name each replica of each fragment. To preserve local autonomy:
  - <**local-name, birth-site**>
- Site 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.

Parallel & Distributed Databases

Distributed Queries Processing

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

Horizontally Fragmented: Tuples with rating < 5 at Shanghai, >= 5 at Tokyo.

- –Must compute SUM(age), COUNT(age) at both sites.
- –If WHERE contained just S.rating>6, just one site.

Distributed Queries Processing

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

Vertically Fragmented: *sid* and *rating* at Shanghai, *sname* and *age* at Tokyo, *tid* at both.

-Must reconstruct relation by join on *tid*, then evaluate the query.

Distributed Queries Processing

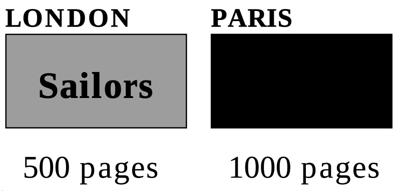
Replicated: Sailors copies at both sites.

- –Choice of site based on local costs, shipping costs.

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

Distributed Queries Processing

Distributed Joins



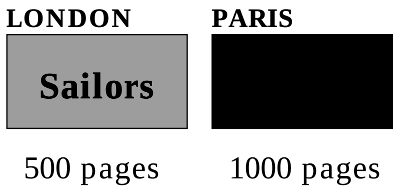
Fetch as Needed,

Page NL, Sailors as outer:

- –**Cost:** 500 D + 500 * 1000 (D+S)
- –**D** is cost to read / write page; **S** is cost to ship page.
- –If query was not submitted at London, must add cost of shipping result to query site.
- –Can also do INL at London, fetching matching Reserves tuples to London as needed.

Distributed Queries Processing

Distributed Joins, Cont



Ship to One Site:

Ship Reserves to London.

- –Cost: 1000 S + 4500 D (SM Join; cost = 3*(500+1000))
- –If result size is very large, may be better to ship both relations to result site and then join them!

Parallel & Distributed Databases

- Motivations and Architecture of Parallel Databases
- Parallel Query Evaluation and Optimization
- Distributed Databases & DBMS Architectures
- Storing Data in a Distributed DBMS
- Distributed Queries Processing
- Updating Distributed Data
- Distributed Transactions

Updating Distributed Data

- Synchronous Replication: All copies of a modified relation (fragment) must be updated before the modifying Xact commits.
  - Data distribution is made transparent to users.
- Asynchronous Replication: Copies of a modified relation are only periodically updated; different copies may get out of synch in the meantime.
  - Users must be aware of data distribution.
  - Current products follow this approach.

Updating Distributed Data

Synchronous Replication

- Voting: Transaction must write a majority of copies to modify an object; must read enough copies to be sure of seeing at least one most recent copy.
  - E.g., 10 copies; 7 written for update; 4 copies read.
  - Each copy has version number.
  - Not attractive usually because reads are common.
- Read-any Write-all: Writes are slower and reads are faster, relative to Voting.
  - Most common approach to synchronous replication.
- Choice of technique determines *which* locks to set.

Updating Distributed Data

Asynchronous Replication

- Allows modifying transaction to commit before all copies have been changed (and readers nonetheless look at just one copy).
  - Users must be aware of which copy they are reading, and that copies may be out-of-sync for short periods of time.
- Two approaches: Primary Site and Peer-to-Peer replication.
  - Difference lies in how many copies are ``updatable'' or ``master copies''.

Parallel & Distributed Databases

- Motivations and Architecture of Parallel Databases
- Parallel Query Evaluation and Optimization
- Distributed Databases & DBMS Architectures
- Storing Data in a Distributed DBMS
- Distributed Queries Processing
- Updating Distributed Data
- Distributed Transactions

Distributed Transactions

- Transactions are divides in each site as sub transactions. Coordination between these sub transactions need :
- Distributed Concurrency Control
- Distributed Recovery

Distributed Transactions

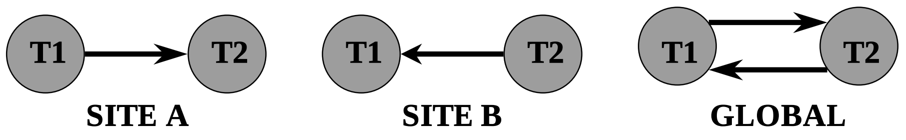
Distributed Concurrency Control

- How do we manage locks for objects across many sites?
  - Centralized: One site does all locking.
    - Vulnerable to single site failure.
  - Primary Copy: All locking for an object done at the primary copy site for this object.
    - Reading requires access to locking site as well as site where the object is stored.
  - Fully Distributed: Locking for a copy done at site where the copy is stored.
    - Locks at all sites while writing an object.

Distributed Transactions

Distributed Deadlock Detection

- Each site maintains a local waits-for graph.
- A global deadlock might exist even if the local graphs contain no cycles:



- Three solutions: Centralized (send all local graphs to one site); Hierarchical (organize sites into a hierarchy and send local graphs to parent in the hierarchy); Timeout (abort transaction if it waits too long).

Distributed Transactions

Distributed Recovery

- Two new issues:
  - New kinds of failure, e.g., links and remote sites.
  - If “sub-transactions” of a transaction execute at different sites, all or none must commit. Need a commit protocol to achieve this.
- A log is maintained at each site, as in a centralized DBMS, and commit protocol actions are additionally logged.