

# Distributed Database Systems

Olaf Hartig

David R. Cheriton School of Computer Science  
University of Waterloo

CS 640  
Principles of Database Management and Use  
Winter 2013

Notes

---

---

---

---

---

---

---

---

## Outline

### 1 Introduction

What is a Distributed Database System?  
Promises and Properties

### 2 Distributed Data Storage

### 3 Distributed Query Processing

### 4 Distributed Transactions

Distributed Concurrency Control  
Distributed Recovery

### 5 Outlook

Notes

---

---

---

---

---

---

---

---

## What is a Distributed Database System?

**Distributed Database (DDB)** A collection of *logically interrelated* data distributed over multiple sites that are connected via a *computer network*

**Distributed Database Management System (DDBMS)** A software that manages a DDB and provides an access mechanism that makes the distribution of the data *transparent* to the user

**Distributed Database System (DDBS)** DDB + DDBMS (i.e., a particular DDBMS that manages a particular DDB)

Notes

---

---

---

---

---

---

---

---

## Implicit Assumptions

- Data stored at a number of sites
- Each site *logically* consists of a single processor
- Processors at different sites are interconnected by a computer network (i.e., not a multiprocessor system\*)
  
- DDB is a database, not just a collection of distributed files
- DDBMS is a full-fledged DBMS

\* A DBMS implemented on a tightly coupled multiprocessor or multicore processor is a *Parallel DBMS*  
→ topic of our discussion next week

Notes

---

---

---

---

---

---

---

---

## Promises and Main Properties

- Improved availability/reliability
  
- Improved performance
  
- Easier and more economical system expansion
  
- Transparent management of distributed data
  - Distributed data independence
  - Distributed transaction atomicity

Notes

---

---

---

---

---

---

---

---

## Desired Properties

**Consistency:** all sites see the same data at the same time  
(i.e., distributed transaction atomicity)

**Availability:** every request to the distributed system must result in a response

**Partition Tolerance:** guaranteed properties are maintained even when some sites cannot communicate with each other (due to network failures)

### CAP Theorem

It is impossible to guarantee all three of these properties in a distributed system.

- “NoSQL database” systems usually settle for *eventual consistency*
- Our focus is more on DDBMSs that guarantee distributed transaction atomicity (and, thus, sacrifice availability during a network partition)

Notes

---

---

---

---

---

---

---

---

## Types of Distributed Database Systems

Notes

**Homogeneous DDBS:** All sites run the same DBMS software

**Heterogeneous DDBS:** Sites under the control of different DBMSs  
(also called *multidatabase systems*)

- Autonomy
- Different sites may use different *local schema*  
(challenge for query processing)

Architectures for a DDBMS:

- Client-Server Architecture
- Collaborating Server Architecture
- Middleware-based Architecture

---

---

---

---

---

---

---

---

## Distributed Data Storage

Notes

(This discussion applies primarily for homogeneous DDBSs.)  
(We assume the relational data model.)

- Different relations stored at different sites
- Relations may be *partitioned* and different sites store different partitions
  - Horizontal partitioning
  - Vertical partitioning
  - Partitions may be partitioned further (i.e., recursively)
  - Combining the partitions must result in the original relation (i.e., lossless-join decomposition for vertical partitioning)
- Replicas of a relation (or partitions thereof) may be stored at multiple sites

---

---

---

---

---

---

---

---

## Replication

Notes

- Motivation:
  - Increased availability of data
  - Faster query evaluation (parallelism, reduced data transfer)
- Replicas need to be kept consistent with one another
  - Updates become more costly
  - Concurrency control becomes more complex

**Synchronous Replication:** transactions include updating all replicas

- If some sites that hold a replica are unavailable, transaction cannot complete
- Coordinating the synchronization requires exchanging many messages

**Asynchronous Replication:** replicas are updated only periodically

- Replicas may be (temporarily) out of sync

---

---

---

---

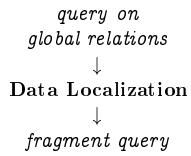
---

---

---

---

## Distributed Query Planning



Substitute each reference to a global relation by its *localization program* (i.e., the definition for reconstructing a global relation from its partitions)

Notes

---

---

---

---

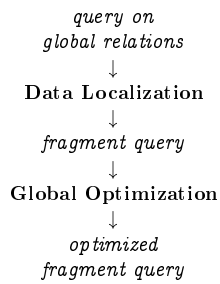
---

---

---

---

## Distributed Query Planning



- Costs (in terms of time):
  - I/O
  - CPU
  - **Communication**
- These might have different weights in different distributed environments (WAN vs. LAN)

Notes

---

---

---

---

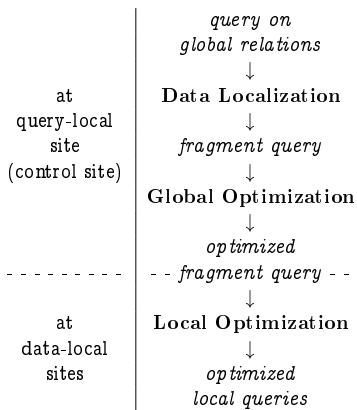
---

---

---

---

## Distributed Query Planning



Notes

---

---

---

---

---

---

---

---

## Distributed Query Execution

- Assume a relation `Employees` that is partitioned horizontally:
  - Site  $S_1$  stores all tuples with `Employees.age > 40`
  - Site  $S_2$  stores all tuples with `Employees.age <= 40`
- Executing query: `SELECT salary FROM Employees WHERE age > 30`
  - 1 Evaluate the query at both sites,  $S_1$  and  $S_2$
  - 2 Take the union of both results at the site where the query was posed
- Executing query: `SELECT AVG(salary) FROM Employees`
  - 1 Compute sum and count at  $S_1$  and  $S_2$ , respectively
  - 2 Compute the average at the query-local site
- If `Employees` is partitioned vertically, both queries can be computed completely at the site that stores the `salary` column
  - The query result might nonetheless need to be shipped.

Notes

---

---

---

---

---

---

---

---

## Distributed Query Execution (cont'd)

- Assume two relations:
  - `Employees` is stored at site  $S_1$
  - `Projects` is stored at site  $S_2$
- Query: `SELECT * FROM Projects P, Employees E WHERE P.mgrid = E.id`
- 1 Nested loops join at, e.g., site  $S_1$ 
  - If `Projects` is inner, cache it at  $S_1$
  - Query result might need to be shipped.
- 2 Ship both relations to where the query was posed and join them there

Notes

---

---

---

---

---

---

---

---

## Distributed Query Execution (cont'd)

- Assume two relations:
  - `Employees` is stored at site  $S_1$
  - `Projects` is stored at site  $S_2$
- Query: `SELECT * FROM Projects P, Employees E WHERE P.mgrid = E.id`
- 3 Semijoin
  - $S_2$  computes  $R := \pi_{mgrid}(Projects)$
  - $S_2$  ships  $R$  to  $S_1$
  - $S_1$  computes  $R' := R \bowtie_{mgrid=id} Employees$   
( $R'$  is called the *reduction* of `Employees` w.r.t. `Projects`)
  - $S_1$  ship  $R'$  back to  $S_2$
  - $S_2$  computes  $R' \bowtie Projects$
- 4 etc.

Notes

---

---

---

---

---

---

---

---

## Distributed Transactions

**Distributed transaction:** a transaction whose actions are executed at multiple sites

**Subtransaction:** a transaction that represents the part of a distributed transaction executed at a particular site

To achieve ACID properties for distributed transactions we need:

- Distributed concurrency control
  - Distributed lock management
  - Distributed deadlock detection
- Distributed recovery

### Assumption

Hereafter, we assume a collaborating server architecture.

Notes

---

---

---

---

---

---

---

---

## Distributed Lock Management

**Centralized:** A single site handles locking for all objects

**Primary Copy:** Locking for any copy of an object managed at the site that stores the *primary copy* of the object

**Fully Distributed:** Locking for a copy of an object managed at the site that stores this copy

Notes

---

---

---

---

---

---

---

---

## Distributed Deadlock Detection

- Required for primary copy locking and for fully distributed locking

**Centralized:** a single site is responsible for global deadlock detection

- periodically, all sites send their local waits-for graphs to that one site which then combines these graphs to detect global deadlocks

**Hierarchy:** sites grouped into a hierarchy

- periodically, sites send their waits-for graphs to their parent in the hierarchy
- less frequent sending in higher levels of the hierarchy

**Timeout:** simply abort any transaction that has been waiting longer than a chosen time interval

Notes

---

---

---

---

---

---

---

---

## Distributed Recovery

- Achieving atomicity and durability for a distributed transaction:
  - Either all subtransactions must commit or none must commit!
- New kinds of failures:
  - Failure of communication links
  - Failure of sites
- Transaction managers responsible for a distributed transaction:
  - Coordinator:** the transaction manager at the site where the transaction originated
  - Subordinates:** transaction managers at those sites that execute subtransactions of the transaction

Notes

---

---

---

---

---

---

---

---

## Two-Phase Commit (2PC) Protocol

Voting phase

- 1 Upon commit, coordinator sends "prepare" to each subordinate
- 2 Upon "prepare", each subordinate i) decides whether to abort or commit its subtransaction, ii) force-writes an abort or prepare log record, and iii) then sends "no" or "yes" to the coordinator

Termination phase

...

Each log record needs to identify the coordinator.

Notes

---

---

---

---

---

---

---

---

## Two-Phase Commit (2PC) Protocol (cont'd)

Termination phase

- 3 "yes" from all subordinates: coordinator force-writes commit log record and then sends "commit" to all subordinates
- 4 Upon "commit", a subordinate force-writes commit log record, sends "ack" to coordinator, and commits its subtransaction
- 5 If "ack" from all subordinates, coordinator writes end log record
- 3 "no" from a subordinate or no response from a subordinate (after a specified timeout interval): coordinator force-writes abort log record and then sends "abort" to all subordinates
- 4 Upon "abort", a subordinate force-writes abort log record, sends "ack" to coordinator, and aborts its subtransaction

Coordinator's commit or abort log record must identify subordinates.

Notes

---

---

---

---

---

---

---

---

## Properties of 2PC

- Any transaction manager involved can abort the transaction
- Transaction officially committed after the coordinator wrote its commit log record
  - Outcome of the transaction not affected by subsequent failures
- Blocking: if the coordinator fails before sending the global decision to all subordinates, the subordinates may need to wait until the coordinator recovers

Notes

---

---

---

---

---

---

---

---

## Outlook

Discussion next week: parallel database systems

- D. DeWitt and J. Gray: Parallel Database Systems: The Future of High Performance Database Systems. *Communications of the ACM* 35(6): 85-98 (1992)

Notes

---

---

---

---

---

---

---

---

Notes

---

---

---

---

---

---

---

---