Distributed Databases

Anne Denton

Department of Computer Science
North Dakota State University

Outline

Table of Contents

Motivations for distributed databases

- Contingency planning through replication
  - Replicated databases consist of multiple identical copies that are kept consistent
- Integrating multiple sites through a distributed database system
  - A "pure distributed" database maintains a single copy of each item of information on the same type of infrastructure (homogeneous)
  - Federated databases may be built on heterogeneous infrastructure and allow remote sites some autonomy
  - May or many not show some level of replication
  - Multi-databases add a level on top of independent databases
- Massive parallelism through No-SQL databases
  - Diverse objectives

Replicated databases

- Motivation
  - Recover from catastrophic failures that affect the entire data center
- Typical characteristics
  - Reads only involve a single replicate, i.e., are fast
  - Writes are done to all replicates, i.e., are slow
  - All copies are identical, i.e. no heterogeneity
- Example
  - At NDSU, Blackboard uses a replicated PostgreSQL database
- Cloud implementation
  - Allow replication across multiple regions

Pure distributed databases

- Motivation
  - Efficient querying of local data with good access to remote data
- Typical characteristics
  - Same type of hardware at different sites
  - Reads and writes may involve multiple sites
  - Distributed commit of transactions results in performance hits even for matching hardware
- Examples
  - Companies with branches in different locations
- Can be combined with replication

Federated database systems

- Motivation
  - Integration of data from multiple sources
- Typical characteristics
  - Allow heterogeneity at different sites
  - Not all sites need access to all remote data
  - Allow autonomy at the different sites
- Example
  - Company with independent subsidiaries
- Drawback
  - Yet slower than pure distributed database systems

Multi-database systems

- Motivation
  - System on top of multiple independent databases
- Characteristics
  - Highest level of autonomy of individual sites
  - Local transactions can be completed without involving the multi-database system
  - Global transactions executed by multi-database system
- Example
  - Company after merger
- Drawback
  - Slowest alternative for global transactions, since multi-database system adds full level

Question 1

Which of the following is best when contingency planning is the primary objective?

1. Centralized database system
2. Replicated database system
3. Pure distributed database system
4. Federated database system
5. Multi-database system

Question 2

Which of the following is best when the ability to continue using multiple existing systems is the primary objective?

1. Centralized database system
2. Replicated database system
3. Pure distributed database system
4. Federated database system
5. Multi-database system

Question 3

Which of the following is best when fast updates are the primary objective?

1. Centralized database system
2. Replicated database system
3. Pure distributed database system
4. Federated database system
5. Multi-database system

Question 4

Which of the following is best when the objective is to build an integrated system that offers a tradeoff of high performance and autonomy between multiple sites?

1. Centralized database system
2. Replicated database system
3. Pure distributed database system
4. Federated database system
5. Multi-database system

Comparison

Comparison of database types

	Autonomy of remote site	Best-case speed	Worst-case speed	Size	Contingency preparation
Centralized DB system	none	high	high	small	none
Replicated DB system	none	high	medium	large	high
Pure distributed DB system	little	high	low	small	none by default
Federated DB system	medium	medium	low	depends	none by default
Multi-database	high	low	low	depends	none by default

Table of Contents

Problems with distributed databases

- All types of distributed database systems have performance bottlenecks at some level
  - Network delays multiple orders of magnitude slower than secondary storage
- Quick fix
  - Big stand-alone servers
  - Even virtualization results in performance bottlenecks due to network-attached storage
  - Databases remained on stand-alone servers longer than most other applications (rather than being deployed in to virtualized environments)

Performance bottlenecks

- Need for distributed queries
  - In a distributed setting, data may have to be transferred
  - Problematic for joins of tables on different systems
  - Also problematic when tables straddle multiple locations
  - Not a problem for replicated databases, since all data are in all locations
- ACID properties of transactions
  - Atomicity, Consistency preservation, isolation, and durability are much harder to preserve
  - Problematic in a replicated and distributed setting

Distributed select-project-join queries

- In distributed databases (whether pure, federated, or multi-databases)
  - Some tables may only exist at some locations
  - Some tables have different subsets of records in different locations
- May require transferring of data

Question 5 (Multiple answers can be correct)

For which of the following functions, over distributed sets of records, do the actual records or fields of all records have to be transmitted, as opposed to only aggregates?

1. Listing of records from a remote table
2. Listing the join of two tables
3. Computing the sum of an attribute
4. Computing the median of an attribute

Table of Contents

Distributed commit

- Most replicated and distributed database systems have some queries that can be evaluated locally, e.g.
  - Read-only queries in replicated databases
  - Queries of local data in pure distributed databases
- Some queries involve multiple or all locations, e.g.
  - Write queries for replicated databases
  - Global queries in most distributed databases
- Maintaining ACID properties requires that global transactions are committed across all sites

Two-phase commit

- Coordinator site communicates with each of the remote sites ("cohort members")
- Maintaining atomicity requires two phases
  1. In the voting phase the coordinator requests a vote from each cohort member
  2. Once all cohort members reply that they are ready to commit the coordinator sends a commit message that instructs all remote sites commit the transaction
- Problem: If the coordinator and a cohort member fail, it can be unclear whether the commit phase had been reached
- Solution: Three-phase commit

Question 6

The voting in a distributed commit tests whether

1. any
2. a majority of
3. all

cohort members are ready to commit.

Three-phase commit

- Introduces an intermediate (pre-commit) phase between the voting and the commit phase
- Helps during recovery
  - If at least one cohort member is in the commit phase, all can be committed
  - If at least one cohort member had not yet gotten the pre-commit notification, all can be rolled back
- Note that every remote site now has to go through three phases and each has to communicate receive and send a notification across the network to a remote location during each of phases

Question 7

Distributed systems require a multi-phase commit. Do they also require a multi-phase abort?

1. Yes, the situation is the same for abort as for commit
2. No, if any one of the cohort members received an abort notification, all will be aborted
3. No. While a commit has to apply to all nodes, aborts affect only individual cohort members.

Question 8 (multiple answers can be correct)

The long duration of network transfers in distributed databases

1. Increases the duration of the distributed transactions overall
2. Increases the probability of deadlock

Table of Contents

Motivation

- Join queries
  - When tables are distributed, join queries can create much network traffic
- ACID properties of transactions
  - Implementations of atomic distributed commits is computationally expensive
  - Consistency preservation for referential integrity difficult
- There are many, very different types of No-SQL databases but they generally
  - Do not allow joining tables
  - Do not have foreign keys

Some types of No-SQL databases

- Key-value store
  - Very basic organization but allows massively distributed data
- Document store
  - Also organized by key, but documents have structure
- Wide-column store
  - Rows and columns, but no joins
- Graph databases
  - Represent data as network

Key-value stores

- Large associative array (sometimes called map or dictionary)
- Among the simplest non-trivial data models
- Multiple models of maintaining consistency exist
  - Some only require eventual consistency
  - Some allow serializability
  - Serializability is easier to achieve without complex constraints
- Example
  - Riak (open source implementation of Amazon Dynamo)

Document store

- Organized through a unique key
- Format can be XML, JSON, YAML
- Examples
  - MongoDB
  - ElasticSearch
- Typical uses
  - Organizing and searching log files

Wide-column store

- Rows and columns, where columns can differ between rows
- Can be interpreted as 2-dimensional key-value store
- Organized by column families
  - Stored by column families
  - Stored row-wise within column families
- Example
  - HBase (open source implementation of Google Bigtable)
- Not to be confused with column stores that use columnar organization
  - Store data by column
  - Similar to dataframe data structures in Python-Pandas and R

Question 9

Which of the following No-SQL databases is best when the data somewhat fit a table format?

1. Key-value store
2. Document store
3. Wide-column store

Question 10

Which of the following No-SQL databases is best when nothing is known about the structure of the data?

1. Key-value store
2. Document store
3. Wide-column store

Question 11

Which of the following No-SQL databases is best for semi-structured data?

1. Key-value store
2. Document store
3. Wide-column store

Comparison with RDBMSs

- No-SQL databases relevant when data widely distributed
- Millions or billions of records
- Not competitive for classic RDBMS applications
- Log storage is the only practical use for which I personally know people who use No-SQL databases professionally