Database Architectures

CSCI 220: Database Management and Systems Design

Slides adapted from
Simon Miner
Gordon College

Practice Quiz: Locking Protocols

- Working with a neighbor:
  - Describe the difference between shared and exclusive locks
  - Describe the two phases of the "two-phase locking protocol"
  - Identify the default isolation level used by PostgreSQL:

Table 13.1. Transaction Isolation Levels

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read uncommitted	Allowed, but not in PG	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible	Possible
Repeatable read	Not possible	Not possible	Allowed, but not in PG	Possible
Serializable	Not possible	Not possible	Not possible	Not possible

Today you will learn…

- How parallelism is used to increase database performance
- Some approaches used in distributed databases

Agenda

- Parallelism
- Distributed Databases
- Introduction to NoSQL

Parallelism

We Need More Power!

- Servers need to support many clients
- CPU and disk bottlenecks can be parallelized
- Old approach: acquire a single fast, expensive computer (e.g., a mainframe)
  - Not sufficiently scalable for many workloads
- Modern approach: acquire a lot of commodity hardware
  - Scaling is easier: buy more servers, expand the RAID array

Speed Up

- Make individual transactions process faster
- Multiple CPUs (and disks) can cooperate to complete a single expensive transaction

Scale Up

- Handle more work in the same amount of time
- **Transaction scale up:** increase the volume of transactions
  - Each CPU handles its own transaction
  - Process more transactions per unit of time
  - For example: handling more website visitors
- **Batch scale up:** increase the size of transactions
  - CPUs cooperate to complete larger transactions
  - For example: as the database grows, calculating analytics requires more processing power

Shared Resources that Enable Parallelism

- Shared memory: multiple CPUs sharing common memory (while also having their own cache/private local memory)
- Shared disk (cluster): multiple CPUs share a disk system
- Shared nothing: each CPU has its own memory and disk

I/O Parallelism

I/O Parallelism

- Reduce the time required to retrieve relations from disk by partitioning the relations on multiple disks.
- **Horizontal partitioning:** tuples of a relation are divided among many disks such that each tuple resides on one disk.
- Definitions:
  - Point query: query to look up a single tuple (i.e. age = 25)
  - Range query: query to look up a range of values (i.e. age > 25 and age < 60)

Partitioning Techniques: Round Robin

- Assume we have *n* disks
- **Round-robin**: Send the I^{th} tuple inserted in the relation to disk *i* mod *n*.
  - Good for sequential reads of entire table
  - Even distribution of data over disks
  - Range queries are expensive

Partitioning Techniques: Hash Partitioning

- Assume we have *n* disks
- **Hash partitioning**: Choose one or more partitioning attribute(s) and apply a hashing function to their values that produces a value within the range of 0…n – 1 disks
  - Good for sequential or point queries based on partition attribute(s)
  - Range queries are expensive

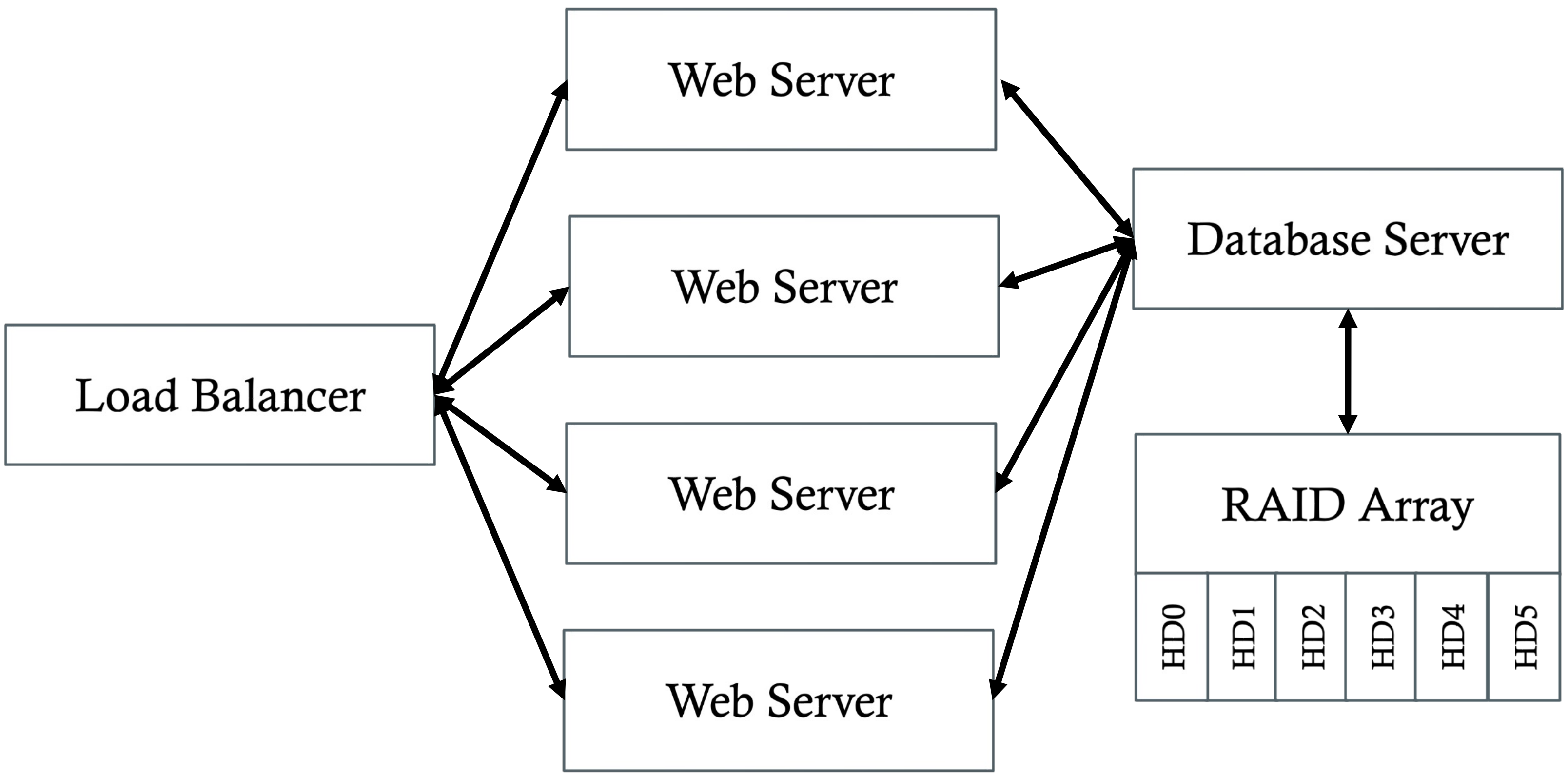
Partitioning Techniques: Range Partitioning

- Assume we have *n* disks
- **Range partitioning:** Choose a partitioning attribute, and divide its values into ranges, tuples that match a given range go in the corresponding partition
  - Clusters data by partition value (i.e. by date range)
  - Good for sequential access and point queries on partitioning attribute
  - Supports range queries on partitioning attribute

Potential Problems

- Skew: non-uniform distribution of database records
  - Hash partitioning: bad hash function (not uniform or random)
  - Range partitioning: lots of records going in the same partition (web traffic/orders stored in a date-partitioned table, more during the shopping season)

Parallelism Example

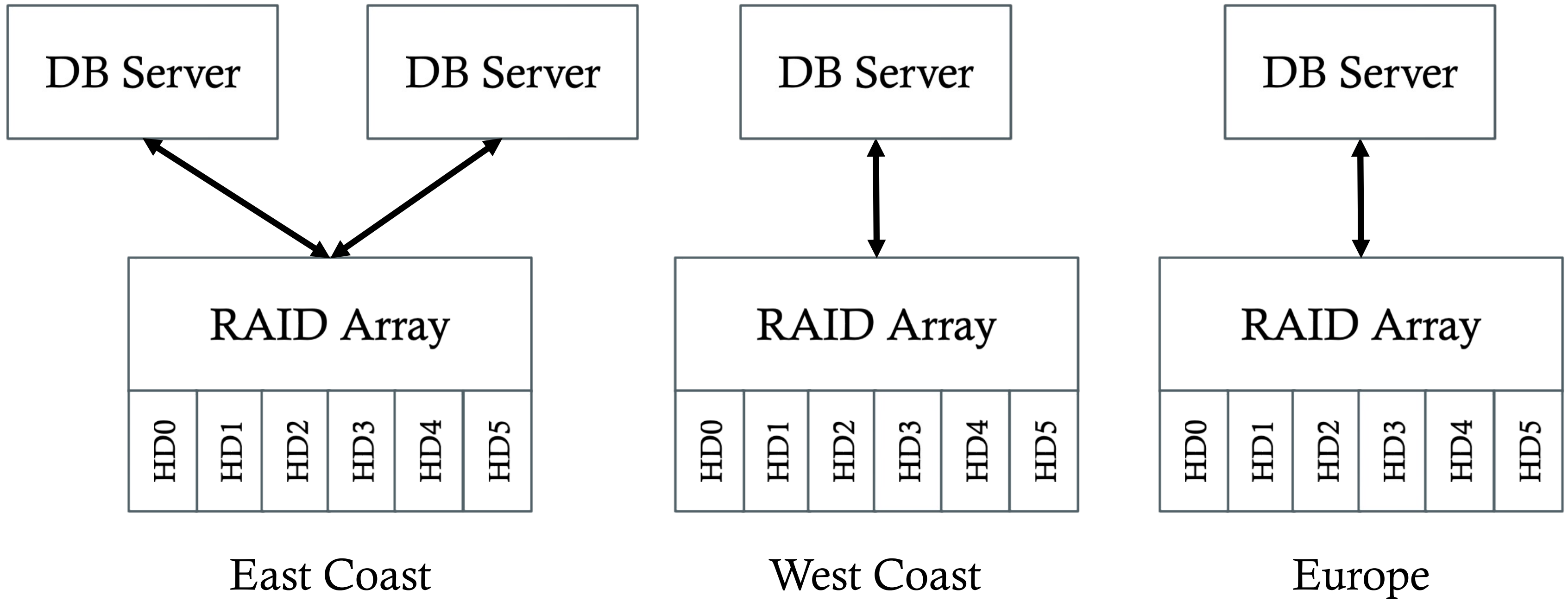


Distributed Databases

One Database, Multiple Locations

- Distributed database is stored on several computers located at multiple physical sites
- Types of distributed database
  - Homogeneous: all systems run the same brand of DBMS software on the same OS and hardware
    - Coordination is easier in this setup
  - Heterogeneous: system run different DBMS on potentially different OS and hardware

Distributed Database Examples



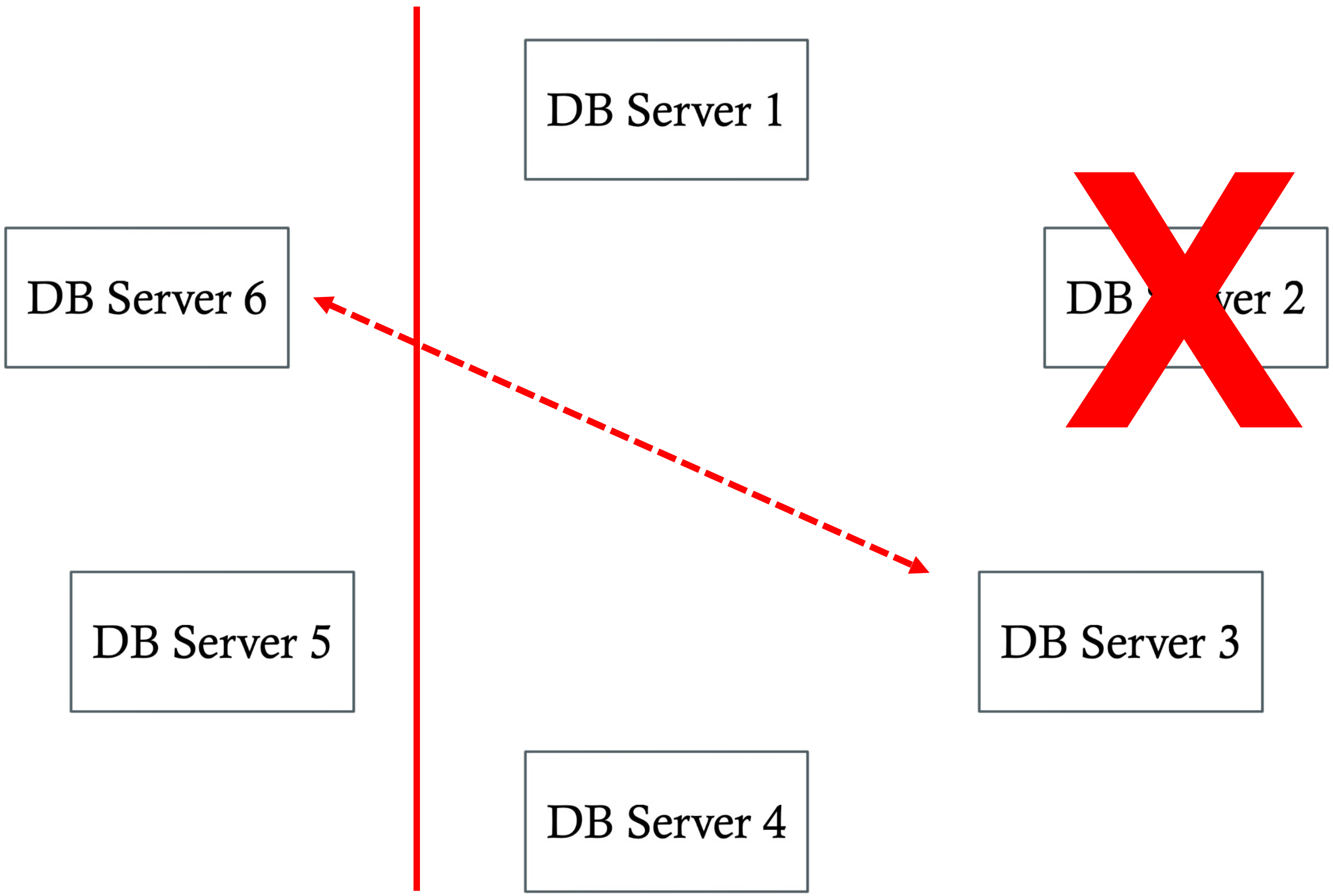
Advantages of Distributed Systems

- Sharing of data generated at different sites
- Local control and autonomy at each site
- Reliability and availability
  - If one site fails, there may be a performance reduction and some data may become unavailable, but processing can continue
  - Contrast with a failure of a centralized system
- Potentially faster query response times
  - Geographically closer data can be accessed with less latency
- Incremental system maintenance and upgrades

Disadvantages of Distributed Systems

- Cost and time required to communicate between sites
  - Operations involving multiple sites are slower because data must be transferred between them
- Increased complexity
- Difficult to debug

Distributed System Challenges



Distributed System Concepts

- Fragmentation
- Replication
- Consensus protocols:
  - Two-Phase Commit
  - Raft

Fragmentation

- Splitting a table up between sites (AKA, *sharding*)
  - Horizontal fragmentation
  - Vertical Fragmentation
  - Fragmentation in both directions
- Mostly applicable to larger organizations
  - Requires more hardware
  - More challenging to manage

Horizontal Fragmentation

- Store different records (rows) at distinct sites
  - Records most pertinent to each site (e.g., US users, European users, etc.)
- Specified by relational algebra selection operation
- Entire table can be reconstructed by a union of records at all sites
- Queries to local rows are inexpensive, but queries involving remote records have high communication cost

Vertical fragmentation

- Store different columns at distinct sites
  - Give access only to data that is needed at site
  - Restrict access to sensitive or unnecessary data at sites
  - Selectively replicate portions of a table
    - Replicate columns frequently used at remote sites for quicker access
- Specified by projection operation
- Entire table can be reconstructed by a natural join on the fragments
  - Requires (primary) key to be present in each fragment
    - Or some system-generated row id (not used by end users)

Fragmentation Example

<table>
<tr><th></th><th>General Personnel Information</th><th>Salary Information</th><th>Job History Information</th></tr>
<tr><td>Eastern Division</td><td>Eastern Division Employees -

Stored at Eastern Division office</td><td>Eastern Division Employees -

Stored at Corporate HQ</td><td rowspan="3">All Employees

Stored at Corporate HQ</td></tr>
<tr><td>Central Division / Corporate HQ</td><td colspan="2">Central Division Employees</td></tr>
<tr><td>Western Division</td><td>Western Division Employees -

Stored at Western Division office</td><td>Western Division Employees -

Stored at Corporate HQ</td></tr>
</table>

Replication

- Storing the same data at different locations
  - Improves performance: local access to replicated data is more efficient than working with a remote copy
  - Improves availability: if the local copy fails, the system may still be able to use a remote copy
- Can be combined with fragmentation
- Issues from data redundancy
  - Requires extra storage
  - Copies must be kept in sync

Choosing whether to Fragment, Replicate, and/or Centralize

- Use **replication** for small relations needed at multiple sites
- Use **fragmentation** for large relations when data is associated with particular sites
- Use **centralization** for large relations when data is not associated with particular sites
  - In this case, communication costs would be higher for fragmentation, as queries would have to access numerous remote sites instead of just the central site

Data Transparency

- Degree to which a user is unaware of how and where data is stored in distributed system
- Types of data transparency:
  - Fragmentation transparency
  - Replication transparency
  - Location transparency
- Advantages
  - Allows data to be moved without user needing to know
  - Allows query planner to determine the most efficient way to get data
  - Allows access of replicated data from another site if local copy is unavailable

Querying Distributed Data

- Queries and transactions can be either
  - **Local:** all data is stored at current site
  - **Global:** it needs data from one or more remote sites
    - Transaction might originate locally and need data from elsewhere
    - Transaction might originate elsewhere, and need data stored locally
- Planning strategies for global queries is difficult
  - Minimize data transferred between sites
  - Use statistical information to assist

Global Query Strategies

- Execute *data reducing operations* before transferring data between sites
  - Produce results smaller than starting data
  - Selection, projection, intersection, aggregation (count, sum, etc.)
  - Sometimes natural and theta join, union
- Execute *data expanding operations* after transferring data between sites
  - Produce results larger than starting data
  - Cartesian join, natural and theta join (sometimes)
- Semijoin -- |X
  - r_1 |X r_2 = π_{R1} (r_1 |X| r_2)
  - Transfer only those tuples in r_1 which match in the natural join with r_2 between sites

Global Query Library Example

- Given
  - checkout relation stored locally
  - (Large) book_info relation (call_no, title, etc.) stored centrally
- Find details (including book titles) of all local checkouts that have just gone overdue
- Strategies
  A. Copy entire book_info relation to the local site and do the join there
     - Not optimal – copying a very large relation for only a few matching tuples
  B. Send local site only those book tuples relevant to the query
     - Semijoin -- book_info |X checkout
     - Data reducing operations at local and central sites

The Challenge of Modifying Distributed Data

- Ensure that updates to data stored at multiple sites get committed or rolled back on each site
  - Avoid one site committing an update and another aborting it
- Ensure that replicated data is consistently updated on all replicas
  - Updates to different replicas do not occur at the same time
  - Avoid inconsistencies arising from data read from a replica that has not been updated yet
- Partial failure: one or more sites down due to hardware, software, or communication link failure
  - What happens when this failure occurs in the middle of an update operation?
  - How to deal with corrupted or lost messages?

Two-Phase Commit Protocol (2PC)

- Ensure that either all updates commit or none commit
  - Here, “updates” = changes to data (inserts, updates, deletes, etc.)
- One site (usually the site originating the update) acts as the coordinator
- Each site completes work on the transaction, becomes partially committed, and notifies the coordinator
- Once coordinate receives completion messages from all sites, it can begin the commit protocol
  - If coordinator receives a failure message from one or more sites, it instructs all sites to abort the transaction
  - If the coordinator does not receive any message from a site in a reasonable amount of time, it instructs all sites to abort the transaction
    - Site or communication link might have failed during the transaction

2PC Phase 1: Obtaining a Decision

- Coordinator writes a <prepare T> entry to its log and forces all log entries to stable storage
- Coordinator sends a prepare-to-commit message to all participating sites
- Ideally, each site writes a <ready T> entry to its log, forces all log entries to stable storage, and sends a ready message to the coordinator
  - If a site needs to abort the transaction, it writes a <no T> entry to its log, forces all entries to stable storage, and sends an abort message to the coordinator
  - Once a site sends a ready message to the coordinator, it gives up its right to abort the transaction
    - It must commit if/when the coordinator instructs it to

2PC Phase 2: Recording the Decision

- Coordinator waits for each site to respond to the prepare-to-commit message
- If any site responds negatively or fails to respond, coordinator writes an <abort T> entry to its log and sends an abort message to all sites
- If all responses are positive, coordinator writes a <commit T> entry to its log and sends a commit message to all sites
- At this point, the coordinator's decision is final
  - 2PC protocol will work to carry it out even if a site fails
- As each site receives the coordinator's message, it either commits or aborts the transaction, makes an appropriate log entry, and sends an acknowledge message back to the coordinator
- Once the coordinator receives acknowledge messages from all sites, it writes a <complete T> entry to its log
- If a site fails to send an acknowledge message, the coordinator may resend its message to it
  - Ultimately, the site is responsible to find and carry out the coordinator's decision

2PC: If a Remote Site or Communication Link Fails…

- …before sending its ready message, the transaction will fail
  - When the site comes back up, it may send its ready message, but the coordinator will ignore this
  - Coordinator will send periodic abort messages to site so that it will eventually acknowledge the failure and return to a consistent state
  - Same scenario as above if ready message is lost in transit
- …after the coordinator receives the ready message
  - The site must figure out what happened to the transaction once it recovers (via a message from coordinator or asking some other site) and take appropriate action
- …after the site receives the coordinator's final decision
  - The site will know what to do after it recovers (from commit or abort entry in its log)
  - Takes appropriate action and sends an acknowledgement message to the coordinator

2PC: If the Coordinator Fails…

- …before it sends a final decision
  - Sites that already sent ready messages have to wait for coordinator to recover before deciding what to do with the transaction
    - Can lead to *blocking* – locked data items unavailable until coordinator recovers
  - Sites that have not sent ready message can time out and abort the transaction
- …after sending a final decision to at least one site, it will figure out what to do after it recovers based on its log
  - <start T> but no <prepare T> → abort transaction
  - <prepare T> but no <commit T> → find out status of sites or abort transaction
  - <abort T> or <commit T>, but no <complete T> → restart sending of commit/abort messages and waiting for acknowledgements
- Sites may be able to find out what to do from each other when the coordinator is down

Updating Replicated Data

- All replicas of a given data item must be kept synchronized when updates occur
- Approach: simultaneous updates of all replicas for each transaction
  - Ensures consistency across replicas
  - Slows down update transactions and breaks replication transparency
  - What happens if a replica is unreachable during an update?

Primary Copy

- Designate a *primary* copy of the data at some site
  - Reads can happen on any replica, but updates happen on primary copy first
  - Primary copy's site sends updates to replica sites
    - Immediately after each update or periodically (if *eventual consistency* is OK)
    - Resending updates periodically to sites that are down
- Secondary copies might be a little out-of-date, so critical reads should go to the primary copy
- What happens when the site with the primary copy fails?
  - Data becomes unavailable for update until the primary copy site is recovered
  - Or, a secondary copy can become a temporary primary copy
    - Could lead to inconsistencies when trying to reactivate the real primary copy

Further Reading

- What is Distributed SQL? An Evolution of the Database
  - CockroachDB (open source, developed by former Google engineers)
  - Google F1 (proprietary) and Spanner (proprietary, offered by Google Cloud)
  - Amazon Aurora (proprietary, offered by AWS)
- Distributed functionality for traditional RDBMS's:
  - Features of standard PostgreSQL and Citus for PostgreSQL
  - Features of standard MySQL and MySQL Cluster