Database Architectures

CPS352: Database Systems

Simon Miner
Gordon College
Last Revised: 4/15/15

Agenda

- Check-in
- Parallelism and Distributed Databases
- Technology Research Project
- Introduction to NoSQL
- Homework 6

Check-in

Parallelism

We Need More Power!

- Parallelism brought on by the success of the client-server model
  - Servers need to support more clients with more demanding operations
- Alternative to acquiring bigger faster more expensive hardware
- Bottlenecks which can be parallelized
  - CPU
  - Disk

More Speed for More Stuff!

- Speed-up – make individual transactions process faster
  - Multiple CPUs cooperate to complete a single (expensive) transaction
- Scale-up – handle more work in the same amount of time
  - Batch scale-up – increase the size of transactions (as database grows)
    - CPUs cooperate to complete (larger) transactions
  - Transaction scale-up – increase the volume of transactions
    - Each CPU handles its own transaction, but more can be processed at the same time

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

- 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.

- Partitioning techniques (number of disks = *n*):
  - **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
  - **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
  - **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

- Skew – non-uniform distribution of database records

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

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
  - For locally stored data – don't need to go to a central store
  - Multiple sites can potentially work on the same query in parallel
- 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

Fragmentation

- Splitting a table up between sites
  - Also called *sharding*
- Horizontal fragmentation
- Vertical Fragmentation
- Fragmentation in both directions

Horizontal Fragmentation

- Store different records (rows) at distinct sites
  - Records most pertinent to each site (i.e. store, plant, branch)
- 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
  - Updates to multiple copies of data
    - Update strategy must ensure that an inconsistent replica is not used to update other copies, but rather is itself restored to a consistent state

Choosing whether to Fragment and/or Replicate

- Use replication for small relations needed at multiple sites
- Use fragmentation for large relations when multiple sites need to access a static set of column
- Use centralization for large relations when there is no fixed set of columns which multiple sites need access to
  - In this case, communication costs would be higher for fragmentation
    - 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

Names of Data Items

- Criteria – Each data item in a distributed system should be
  - Uniquely named
  - Efficient to find
  - Easy to relocate
  - Each site should be able to create new items autonomously
- Approaches
  - Centralized naming server
    - Keeps item names unique, easy to find, easy to move (via lookup)
    - Names cannot be created locally -- high communication cost to get new names
      - What happens if the naming server goes down?
  - Incorporate site ID into names
    - Meets criteria, but at the cost of location transparency
  - Maintain a set of aliases at each site mapping local to actual names
    - i.e. customer => site17.customer

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
  - 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
  - Send local site only those book tuples relevant to the query
    - Semijoin -- book_info |X checkout
    - Data reducing operations at local and central sites

Datatabases

...of the Bible

Where's that Epistle?

Colossians 4:15-18

Modifying Distributed Data can be Complicated.

- Challenges related to updating data in a distributed system
  - 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
- How to do this
  - 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

Concurrency Control with Distributed Systems

- How to ensure serializable transactions in a distributed system?
- Locks – need to lock an item at multiple sites before accessing it
- Centralized lock manager – all locks obtained from this lock manager on one site
  - Transaction needing to lock several replicas at once can get all of its locks in a single message
  - Single source for dealing with deadlock
  - Local transactions involving locking incur communication overhead
  - Locking manager becomes a bottleneck and single point of failure

Distributed Locking

- Each site manages the locks of items stored there
  - Local transactions stay local, no single point of failure
- Disadvantages
  - More message overhead – need to send lock request, receive lock granted, and unlock message in addition to the data involved
  - Deadlock detection gets harder
  - Further complications to updating replicated data
    - How many replica locks are needed to do an update (all of them? Most of them?)
    - Primary copy method helps with this, as only primary copy needs to be locked

Timestamps for Distributed Concurrency Control

- Must ensure consistency and uniqueness of timestamps across sites
  - Combine locally generated timestamp and site id into a transaction's global timestamp
- Need to ensure that all sites' clocks are always synchronized with one another
  - If any site receives a request from a transaction originating elsewhere…
  - And that transaction's timestamp is greater than the current site's timestamp clock
  - Advance the local timestamp clock to one greater than the transaction timestamp

Technology Research Project

NoSQL

Pros and Cons of Relational Databases

- Advantages
  - Data persistence
  - Concurrency – ACID, transactions, etc.
  - Integration across multiple applications
  - (Mostly) Standard Model – tables and SQL
- Disadvantages
  - Impedance mismatch
  - Integration databases vs. application databases
  - Not designed for clustering

Impedance Mismatch

- Different representations of data when it is in the RDBMS vs. in memory
  - In-memory data structures use lists, dictionaries, nested and hierarchical data structures
  - Relational database only stores atomic values
    - No lists or nested records
  - Translating between these representations can be costly and confusing
    - Limits the productivity of application developers
- Object-relational mapping (ORM) can help with this
  - Abstraction can lead to neglect of query performance tuning

Impedance Mismatch Example

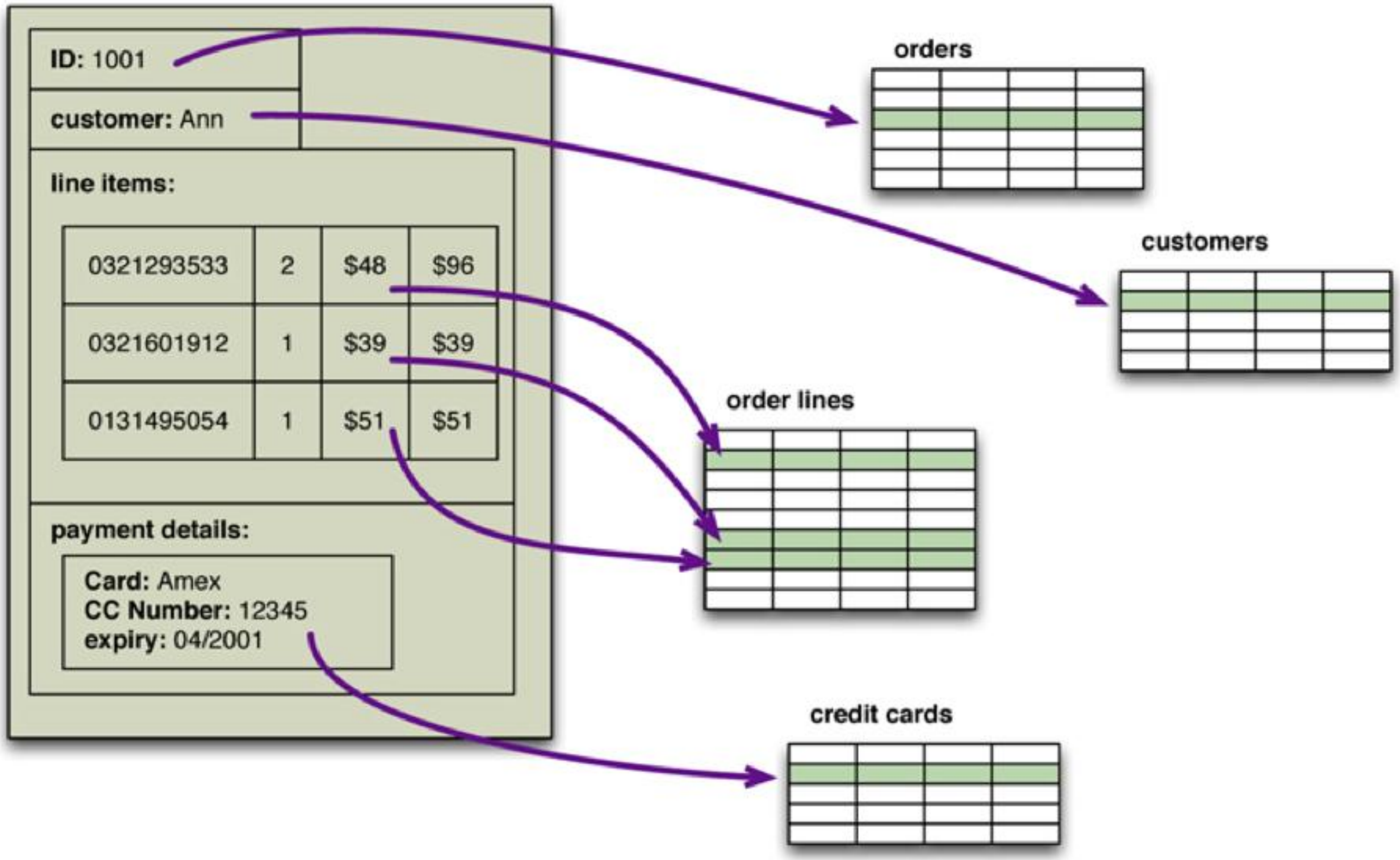

Figure 1.1. An order, which looks like a single aggregate structure in the UI, is split into many rows from many tables in a relational database

Integration vs. Application Databases

- Integration databases support multiple applications
  - Can be problematic if the applications have very different needs and are maintained by separate teams
    - Who maintains the database?
- SQL can be limiting as the only shared layer
  - Web services have become a more flexible alternative
- Application databases are simpler to deal with
  - Application is the only thing using the database
    - No connections from external sources
  - Security and flexibility decrease in priority

The Need for Clusters

- The Internet created the need to store and process huge amounts of data
  - Relational databases can scale "up" (bigger machine) , but not "out" (many machines) as well
    - Disk subsystem remains a single point of failure
    - Distributing/fragmenting/sharding data is complicated
    - High licensing costs for many database machines and CPUs
- Large web companies began developing their own alternative technologies to deal with these issues
  - Google's BigTable and Amazon's Dynamo
  - Issues addressed by these solutions have become relevant to smaller companies wanting to capture and analyze lots of data

The Emergence of NoSQL

- Ironically, the term “NoSQL” was first used as a name for an open source relational database released in the late 1990’s

- Term as it is used today was a hastily-chosen Twitter hash tag for a conference meet-up on the topic in 2009

- No official general definition for *NoSQL*, but common characteristics include:
  - Does not use the relational model (mostly)
  - Generally open source projects
  - Driven by the need to run on clusters
  - Built for the need to run 21^{st} century web properties
  - Schema-less

- More of a movement than a technology
  - Relational databases are not going away
  - *Polyglot persistence* – use the type of data store most appropriate for the situation

Homework 6