Goals for Today

• Discuss distributing data and workload for increased performance in a DBMS

• Reason about DBMS concepts in parallel and distributed setting
  – How to achieve distributed ACID transactions
  – Data consistency across copies

• Understand why some of the disadvantages of distributed databases have led to some relaxation of consistency

Some Parallelism Terminology

• Throughput
  – Amount of work done per unit time

• Latency (response time)
  – Time to complete one unit work

• Speed-Up
  – For given amount of work, more resources means proportionally less time.

• Scale-Up
  – If resources increased in proportion to increase in work size, time is constant.

Logistics

• Final exam timing posted in “important dates”
  – Final (seniors)  take-home  05-04 at 2pm
  – Final (non-seniors)  take-home  05-10 at 5pm

• Lab 5 due next Wednesday
• Problem set 11 due next Thursday

• Last day of Mudd classes is next Friday!
  – For us, next Thursday

Resource contention impacts scale-up/speed-up
Parallel DBMS Architectures

- Multiple processors (CPUs) can do work in parallel
  - How do they communicate about what work to do?
  - Three main parallel DB architectures:
    - Shared Memory (aka shared everything)
    - Shared Disk
    - Shared Nothing

Data Partitioning

- Partitioning (aka sharding aka horizontally fragmenting) a table:
  - Range
  - Hash
  - Round Robin

Different Types of DBMS Parallelism

- **Inter-query** parallelism: different queries run on different nodes
- **Intra-operator** “partitioned” parallelism
  - Multiple machines working together to execute an operator (e.g., scan, sort, join)
  - Machines work on disjoint partition of the data
- Parallelizing a relational operator: merge and split
  - Merge streams of output to serve as input to an operator
  - Split output of operator to be processed in parallel

Example: Parallelizing Aggregation

- For each aggregate function, need a decomposition:
  - \( \text{count}(S) = \sum \text{count}(s(i)) \), same for \( \text{sum()} \)
  - \( \text{avg}(S) = (\sum \text{sum}(s(i))) / \sum \text{count}(s(i)) \)
  - and so on...
- For group by aggregates:
  - Sub-aggregate groups at each node
  - Ship each sub-aggregate to a node in charge of that group for final aggregation

Jim Gray & Gordon Bell: VLDB 93 Parallel Database Systems Survey
Different Types of DBMS Parallelism

- **Inter-operator “pipelined” parallelism**
  - Each relational operator may run concurrently on a different machine
  - Output of first operator consumed on-the-fly as input to second operator

  ![Sequential Code](image)

  Could be limited by blocking operators such as sort or aggregation

Exercise 2

- (a) inter-query
- (b) intra-operator “partitioned” parallelism

Distributed DBMS (Shared Nothing)

- Data is stored at several sites (geo-distributed), each managed by a DBMS that can run independently

  Extends Physical and Logical Data Independence principles

- **Distributed Data Independence:**
  Users should not have to know where data is located
  - Note: catalog needs to keep track of where data is

- **Distributed Transaction Atomicity:**
  Users should be able to write Xacts accessing multiple sites just like local Xacts

Distributed Query Processing: Joins

- **Approach 1 -- Fetch as Needed:** Page NL, Sailors as outer (query submitted at London):
  - \( D \) is cost to read/write page; \( S \) is cost to ship page
  - **Cost:** \( 500 \, D + 500 \times 1000 \, (D+S) = 500,500 \, D + 500,000 \, S \)
  - If query not submitted at London, must add cost of shipping result to query site

- **Approach 2 -- Ship to One Site:** Ship Reserves to London
  - **Cost:** \( 1000 \, (D+S) + 500 \, D + 500 \times 1000 \, D = 501,500 \, D + 1000 \, S \)

Could also consider other single-site join methods

LONDON
- **Sailors**
  - 500 pages

PARIS
- **Reserves**
  - 1000 pages

JOIN on sid
Semijoin

• Why ship all of Reserves to London?
  — Some of these tuples may not even end up being joined, wasted communication cost
  — Idea: only ship the Reserves tuples that will match Sailors tuples

  ![Diagram showing Sailors and Reserves with a semijoin operation]

• Bottom line: Tradeoff the cost of computing and shipping projection for cost of shipping full Reserves relation.

• Note: Especially useful if there is a selection on Sailors, and then join selectivity is also high

Refinement: Bloomjoin

• Idea: rather than shipping the join column, ship a more compact data structure that captures (almost) the same info
  — Bloom filter bit vector
  — Bit-vector cheaper to ship, almost as effective (false positives possible)

• Hash Sailors.sid values into range [0,k-1]
  — If tuple hashes to slot i, set bit i to 1
• Hash Reserves tuples into same range [0,k-1]
  — Discard tuples that hash to 0 in Sailors bit vector

  ![Diagram showing Sailors and Reserves with a Bloomjoin operation]

Query Optimization

• New considerations for cost-based approach
  — Communication costs
  — New distributed join methods

• Also optimizing for query response time?
  — Might want to consider a larger space of plans

  ![Query optimization tree diagram]

Exercise 3

• Concerns: correctness, deadlock, performance
• Some options:
  — All lock requests go through a central location
  — Lock requests distributed; request is sent to the site that holds the data that is desired
Distributed Transactions

- With data at distributed sites, a transaction may operate on data at multiple sites
  - xact broken down into sub-xacts that execute at each site
- Example: read and update inventory at four sites with horizontal partitions of the data
  - T: \( R(A), R(B), R(C), R(D), W(A), W(B), W(C), W(D) \), commit
- How do we guarantee atomicity??
  - Need a commit protocol (type of consensus protocol)
  - A log is maintained at each site, as in a centralized DBMS, and commit protocol actions are additionally logged

Two-Phase Commit (2PC)

- Site at which xact originates is coordinator; other sites at which it executes are subordinates
- Suppose xact wants to commit (normal execution):

2PC: Steps

- When a Xact wants to commit:
  1. Coordinator sends prepare msg to each subordinate.
  2. Subordinate force-writes an no or ready log record and then sends a no or yes msg to coordinator.
  3. If coordinator gets unanimous yes votes, force-writes a commit log record and sends commit msg to all subs. Else, force-writes abort log rec, and sends abort msg.
  4. Subordinates force-write abort/commit log rec based on msg they get, then send ack msg to coordinator.
  5. Coordinator writes end log rec after getting all acks.

Site and Link Failures 😞

- If coordinator detects a subordinate failed, e.g., after timeout

Exercise 4: when the failed subordinate site wakes up, can it tell if a global transaction committed or aborted?

- If coordinator for Xact T fails, subordinates who have voted yes cannot decide whether to commit or abort T until coordinator recovers
  - Xact T is blocked
Exercise 4

• Site that wakes up should interpret its log
  — (a) T committed! it should REDO actions for T
  — (b) T aborted! it should UNDO actions for T
  — (c) Can’t tell... Needs to consult coordinator about what coordinator decided
  — (d) T aborted! it should write abort, and UNDO
    • Since it never even wrote ready, coordinator couldn’t possibly have decided to commit (since it would have waited for the site to say yes)

(We’ll finish the following slides next time)

Data Replication

• Replication: keep copies of data at different sites

• Benefits
  — Gives increased availability
  — Faster query evaluation

• Flavors
  — Synchronous (eager) vs. Asynchronous (lazy)
    • Vary in how current copies are (i.e., how consistent they are)

• Can be used in addition to data partitioning
  — Full replication: copy of all data at every site (vs. partial)

  **Locking:** on primary copy or fully distributed

Updating Distributed Data

• Synchronous (Eager) Replication: set of copies of a modified relation must be updated **before the modifying xact commits**
  — Exclusive locks on all copies modified
  — Users/apps do not need to know data location(s)

• Asynchronous (Lazy) Replication: Copies of a modified relation are only periodically updated
  — Different copies may get out of sync in the meantime
  — Users/applications must be aware of data location(s)
Synchronous Replication: Majority

- Majority voting technique guarantees consistency:
  - Xact must write a majority of copies to modify an object
  - Each copy has version number for object
  - Xact must read enough copies to be sure of seeing at least one most recent copy

- Example: 6 copies of data
  - Written: 4 nodes
  - Read: 3 nodes

Cost of Synchronous Replication

- Before an update Xact can commit, it must obtain locks on all copies (assuming ROWA)
  - Sends lock requests to remote sites, and while waiting for the response, holds on to other locks!
  - If sites or links fail, Xact cannot commit until they are back up

Asynchronous Replication

- Modifying xact can commit before all copies have been changed
  - Users/apps must be aware of which copy they are reading, and that copies may be out-of-sync for short periods of time

- Two approaches for replication:
  - Primary Site (aka master)
  - Peer-to-Peer (aka multi-master or update-anywhere)
  - Difference lies in how many copies are "updatable"

Primary Site Replication

- Exactly one copy of a relation partition is designated the primary or master copy.
  - Replicas at other sites cannot be directly updated

- How are changes to the primary copy propagated to the secondary copies?
  - One approach: log shipping

So the alternative of asynchronous replication is becoming widely used
Peer-to-Peer Replication

- More than one of the copies of an object can be a master
- Changes to a copy must be propagated to other copies
- If two copies are updated in a conflicting manner, this must be resolved
  - E.g., Last write wins? Combine updates somehow?