Parallel Updates (Queris Continued)
Network Fails

Why not option 2?
- B doesn't have an up to date view of the data

Option 1 (consistency)
- A is restarted
  - Switch over to B as primary

Option 2 (availability)
- B reconnects as a follower

B fails
- Nothing happens
- B is restarted
  - B catches up to A's state
  - Yay!

A fails
- Everything fails
C

A

P

Theorem
- W(\(x=2\))
  - tell R "ready"
  - be told "ready"
  - \(R(x) \rightarrow 2\)
\[
\frac{w}{N} \leq \frac{1}{N} \quad \text{for} \quad r + w > N \quad \text{and} \quad N - w > 1
\]

N nodes

w #nodes the writer waits for
r #nodes the reader waits for
Parallel Data

Types of Parallelism

- Replication (Multiple copies of the same data)
  - Better throughput for read-only computations
  - Data safety

- Partitioning (Different data at different sites)
  - More space
  - Better throughput for writes
  - Sometimes better throughput for read-only computations

Challenges

- Replication
  - Reading the same value from each site.

- Partitioning
  - Transactions (Update A and B atomically)

Consensus

Getting everyone to agree on something

- Did a transaction commit?
- In which order were the transactions applied?
- What is the current value of object A?

Techniques

- Primary/Secondary (aka Leader/Follower, aka Master/Slave)
  - Pick one node as the primary
    - Deterministic property (lowest IP, etc...)
    - Additional consensus protocol for leader selection
  - Primary is the authoritative version
    - All writes go to the primary first.
    - Writes are replicated to the secondary(ies) if any exist.
    - Secondaries can handle (potentially stale) reads, but not writes

- 2-Phase Commit
  - Every time something happens, everyone communicates with everyone else.
  - All participants signal readiness to participate in consensus
  - A temporary, per-consensus task ‘leader’ signals all other participants to vote
  - All participants communicate their vote to the leader.
  - Leader tallies votes based on goal requirements
- k-Data stability requires k replicas to acknowledge
- Commit/Abort requires unanimous acknowledgement
- The leader notifies everyone of the vote result.

**Log Consensus**
- Sometimes possible. Nodes log messages in an agreed-upon order. Nodes agree to any message they receive in the correct, agreed-upon order.

**Failure Modes**

**Fail-Fast / Fail-Stop**
- Software/Hardware failure that causes the node to crash (although it can eventually be restarted)
- The node stops functioning outright — no signs of life at all

**Non-Fail-Stop**
- Software/Hardware failure that causes the node to behave incorrectly
- The node keeps responding, but does not respond according to the programmer's expectations

**Byzantine Faults**
- Software/Hardware failure that causes the node to behave as incorrectly as possible.
- The node responds in the most harmful way possible.

**Failures**

**What can fail?**
- The node itself
- The network connecting the nodes
- Part of the network connecting the nodes (partition)

**Does it matter which?**
- If the node crashes, it loses its local state and has to be restarted from scratch
- If the network fails... both nodes continue to be active but are unaware of each other's existence... but may be aware of the existence of other nodes.

**Can a node tell which is which?**
- No. If Nodes A and B are trying to reach consensus, and B stops responding, A has no clue why.
- So, what happens when the failure condition ends?

**Recovery in Primary/Secondary Replicas**

**Secondary Node Failure**
- No Harm. Secondary reboots and rejoins.

**Primary Node Failure**
- A secondary can rise to take its place... Repeat leader selection process
- Primary reboots as a secondary

**Network Failure**
- From the point of view of secondaries... identical to primary node failure.

**Partitions in Consensus**
Option 1: Assume Node Failure
- Maximize availability. Promote secondary to primary to ensure that there’s always a primary available.
- Creates risk of inconsistency, as there are now two primaries. Two authoritative versions of the data.

Option 2: Assume Connection Failure
- Ensure consistency. Wait for network (or primary node) to recover.
- Affects availability. Can’t do anything until the primary recovers.

CAP
- Consistency, Availability, Partition-Tolerance
- Pick any 2
- More precisely, pick a tradeoff between consistency and availability. How much of each are you willing to sacrifice.

Reader/Writer Stability

In a system with N nodes, you want to read the 'latest' version that everyone agrees on.

Failure mode:
- Receive Ack for write
- Successfully Read an earlier value

Naive:
- Write to N nodes, wait for everyone to acknowledge write.
- Read from N nodes, wait for everyone to agree on read.

Fault-Tolerant
- Write to N nodes, wait for w nodes to acknowledge write
- Read from N nodes, wait for r nodes to agree on read.
- If w+r > N, there must be one overlapping node. Guaranteed to be reading at least latest acked value.
- Can tolerate F failures if w + r - F > N