Dynamo

Tom Anderson

Outline

Last two weeks: external consistency

- Chubby: coordination service
- BigTable: scalable storage of structured data
- GFS: large-scale storage for bulk data
- Spanner: Multi-key, multi-data center NoSQL

Today: eventual consistency

- Dynamo: Eventually consistent NoSQL

Motivation: Fast Available Writes

- Shopping cart: always allow customer to buy
- Write availability with external consistency?
  - Delay writes whenever quorum is down
- Write availability across data centers?
  - what if network access is partitioned?
- Performance/throughput
  - External consistency requires (logically) single copy
  - Either control of the copy pings around network
  - Or all updates must be streamed to the single copy
- Multi key operations even worse
  - need to coordinate updates across keys

Possible Solutions

- Snapshot reads: allow reads on consistent but slightly stale version of data
  - Improves write performance by decoupling reads
  - Example: GFS returns consistent prefix of file
  - Example: Spanner snapshot reads
- Commutativity: if operations can be redesigned to yield same result regardless of order
  - Example: UNIX file descriptor is “next unused slot”
  - Could be: “any unused slot”

Possible Solutions

- Post-hoc resolution
  - use logs, version vectors to detect when reconciliation is needed
  - Application-specific merge of different versions
- Git, Dynamo, …

Dynamo Goals

Dynamo design:

Replication within, across a small number of data centers

Limited scale (100s): every service uses a Dynamo copy

Goals:

- Improve 99.9th percentile of delay
- Handle constant server failures
- Handle "data centers being destroyed by tornadoes”
- Data "always writeable"

Implications

- Availability => replicas
- Always writeable => allow writes to bypass replicas when down or partitioned
- Always writeable => no paxos, no primary, no leases
- Multiple data centers => partitions likely
- Always writeable + replicas + partitions => conflicting versions

Eventual Consistency

Eventual consistency among versions

- Accept writes at any replica
- Allow divergent replicas
- Allow reads to see stale data
- Allow reads to see multiple versions
- Anti-entropy: store multiple versions at each replica
- Resolve conflicts when failures go away

Eventual Consistency Downsides

- There can be several “latest” versions
- Read can yield any (or all) versions
- Application must merge and resolve conflicts
- No atomic operations
  - No test-and-set
  - No de-friend and dis

Dynamo API: Simple Key Value

- get(k) -> set of (value, "context”)
  - context is version info
- put(k, v, context)
  - context indicates which versions this put supersedes

Where Should Data Be Placed?

Goals:

- Balance load, including as servers join/leave
- Replication for fault tolerance
- Find keys, including when there are failures
- Encourage put/get to see each other
- Avoid conflicting versions spread over many servers

Consistent Hashing

- Node ID assigned at random
  - Virtual nodes for better load balancing
  - All node IDs known to all clients
- Key ID = hash(key)
- Coordinator: successor of key
  - clients send puts/gets to coordinator
  - join/leave only affects neighbors
- Replicas at successors of coordinator
  - Coordinator forwards puts/gets to replicas

Consistent Hashing
Multiple Data Center Version

- Clients know all server IDs, locations
- Hash(key) determines "preference list"
- Ex: first successor in each data center
  - Vs. first k successors in one data center
- Clients go directly to closest replicas
- Anti-entropy pushes version reconciliation

Node Unreachable

When a node is unreachable, what should we do?

- if really dead, need to make new copies to maintain fault-tolerance
- if really dead, want to avoid repeated waiting
- if just temporary, wasteful to make new copies

Sloppy Quorum

- Do not block waiting for unreachable nodes
- Want get to see most recent put (with high probability)
- Quorum: R + W > N
  - Don't wait for all N
  - R and W will (normally) overlap
- N is first N *reachable* nodes in preference list
- Each node pings to keep estimate of up/down
- "sloppy" quorum -- nodes may disagree on who is reachable

Coordinator/Client

Coordinator or client handling put/get:

- send put/get to first N reachable nodes, in parallel
- put: wait for W replies
- get: wait for R replies

With no failures

- get will see all recent versions

With failures

- writes completely quickly
- reads eventually see?

Failure Corner Cases

What if a put() leaves data far down the ring?

After failures repaired, new data is beyond N

- server remembers a "hint" about where data belongs
- forwards once real home is reachable

Also periodic "merkle tree" sync of whole DB

Multiple Versions

How can multiple versions arise?

- Maybe a node missed the latest write due to network problem
- So it has old data, should be superseded

How can conflicting versions arise?

Network partition => different updates sent to different servers

- Example: Shopping basket with item X
- Partition 1 removes X, yielding ""
- Partition 2 adds Y, yielding "X Y"

Neither copy is newer than the other -- they conflict

- After partition heals, client read gets both versions, because a quorum read (may!) see both

Detecting conflicts: Version Vectors

Example versions at servers a, b:

[a:1]

[a:1,b:2]

Version vector indicates one supersedes the other

Dynamo automatically drops [a:1]

Another Example

[a:1]

[a:1,b:2]

[a:2]

Client must merge

Concurrency and Versions

What happens if two clients concurrently write?

- e.g. to increment a counter
- Each does read-modify-write
- So they both see the same initial version

Will the two versions have conflicting version vectors?

Version Vector Size

Dynamo deletes least-recently-updated entry if version vector has > 10 elements

Impact of deleting a VV entry?

- won't realize one version subsumes another
- put@b: [b:4]
- put@a: [a:3, b:4]
- forget b:4: [a:3]
- now if you sync with [b:4], merge is required

Hopefully never happens

Is Merge Always Possible?

- Suppose we're keeping a counter, x
- x=10, then partition
- incremented by 5 to x=15 in both partitions
- After heal, client sees two versions, both x=15
- What's the correct merge result?

Tail Latency

Does replication help limit 99.9th percentile delay?

Bad news:

- Some replicas may be at distant data centers
- consulting multiple nodes for get/put means at least one will be slow

Good news:

- Dynamo only waits for W or R out of N
- cuts off tail of delay distribution

Flexible N-R-W

What do you get by varying N-R-W?

- 3-2-2 : reasonably fast R/W, reasonably durable
- 3-3-1 : fast W, slow R, not very durable
- 3-1-3 : fast R, slow W, durable
- 3-3-3 : ???
- 3-1-1 : ???