EECS 491
Introduction to Distributed Systems

Fall 2019

Harsha V. Madhyastha
Dynamo: Amazon’s Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

ABSTRACT
Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world. One of the lessons our organization has learned from operating Amazon’s platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service

- Added to SOSP “Hall of Fame”
- Rumored to be underpinning of Amazon S3’s architecture
Dynamo

- Setting:
  - Tens of millions of customers
  - Data spread across tens of thousands of servers

- Example use case: Store shopping carts

- Goals:
  - High availability
  - Low latency
Consistent Hashing in Dynamo

- Recall: Consistent hashing maps value for key to successor in hash space

- Replicate value for every key at N nodes
  - N clockwise successors of key

- Execution of writes
  - Write received by coordinator (successor of key)
  - Coordinator forwards to successors
Replication in Dynamo
Using Consistent Hashing

Client → Front-end
   
Front-end → Server
Front-end → Server
Front-end → Server
Consistent Hashing in Dynamo

What would it take to make this work?

Client -> Server -> Server -> Server -> 1-hop DHT
Gossip

- Once per second, each server **contacts a randomly chosen** other server
- Servers **exchange their lists of known servers**
  - Including virtual node IDs
- How long for everyone to discover server addition/removal?
- Won’t mismatch in view lead to consistency violations?
Sloppy quorums

- N replicas for every key
  - Higher durability with greater N
- Serving reads and writes:
  - Coordinator forwards request to N-1 successors
  - Waits for response from R or W replicas

- How to pick values of R and W ...
  - to maximize availability/minimize latency?
    » Low R and/or low W
  - to ensure read sees last committed write?
    » R+W > N
Latency/availability over consistency

N = 3, W = 1, R = 1

Put(k, y)

Get(k)

Client1

Client2
Consistency over latency/availability

N = 3, W = 2, R = 2

How to tell which of R copies read is latest version?

Client1

Put(k, y)

Client2

Get(k)
Vector clocks

- Store a vector clock with each key-value pair
- What we have discussed previously:
  - Vector with # of components = # of servers
  - Not scalable
- Dynamo’s adaptation of vector clocks:
  - List of (coordinator node, counter) pairs
  - Example: [(A, 1), (B, 3), ...]
Vector clocks in Dynamo

\(N = 3, W = 2, R = 2\)

\([A, 1]\) \(\rightarrow\) \((A, ([A], [B], 1)))\) \(\rightarrow\) \([A, 1], [B, 1]\))

\(\text{Client1} \xrightarrow{\text{Put(k, x)}} \text{Client2} \xrightarrow{\text{Get(k)}} \text{Client1} \xrightarrow{\text{Put(k, y)}} \text{Client2} \)

\(([A, 1], [B, 1]) > ([A, 1])\)
Project 3

- Focus first on correctness, then efficiency

- A peer may be acting as proposer, acceptor, and learner on same instance concurrently
  - Keep roles as separate as possible

- Okay to move on to part B if passing all handout test cases for part A

- Reminder about Honor Code
Vector clocks in Dynamo

N = 3, W = 2, R = 2

Client1

k: x

Put(k, x)

N = 3, W = 2, R = 2

Client2

k: y

Get(k)

Put(k, y)

k: x

Put(k, y)

([B, 1]) and ([A, 1]) are in conflict!

How to recognize that Put(k, x) → Put(k, y)?
Vector clocks in Dynamo

- How can vector clocks help in differentiating
  - Conflict?
  - Okay to garbage collect?

- When responding to a GET, Dynamo returns the vector clock for value returned

- Client includes vector clock in subsequent PUT
Automatic conflict resolution

\[ v_2 > v_1, \text{ so Dynamo automatically drops } v_1 \text{ at } C \]
App-specific conflict resolution

\[ v_1 = [(A,1)] \]

- Put handled by node A
- Put handled by node B
- Put handled by node C

\[ v_2 = [(A,1), (B,1)] \]
\[ v_3 = [(A,1), (C,1)] \]

Client reads \( v_2, v_3 \); writes with \( [(A,1), (B,1), (C,1)] \)

\[ v_4 = [(A,2), (B,1), (C,1)] \]

\( v_2 \parallel v_3 \), so client must perform reconciliation.
Dynamo’s client interface

- Client interface in projects:
  - Get(key) → value
  - Put(key, value)

- Get(key) → List of <value, context> pairs
  - Returns one value or multiple conflicting values
  - Context describes version(s) of value(s)

- Put(key, value, context)
  - Context indicates which versions this version supersedes or merges
Trimming version vectors

- Many nodes may process Puts to same key
  - Version vectors may grow arbitrarily long

- Dynamo’s clock truncation scheme
  - Dynamo stores time of modification with each version vector entry
  - When version vector > 10 nodes long, Dynamo drops node that least recently processed key

- Problems with truncation?
  - False concurrency
Impact of clock truncation

\[ v_1 [(A,1)] \]

put handled by node A

\[ v_2 [(A,1), (B,1)] \]

put handled by node B