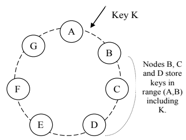
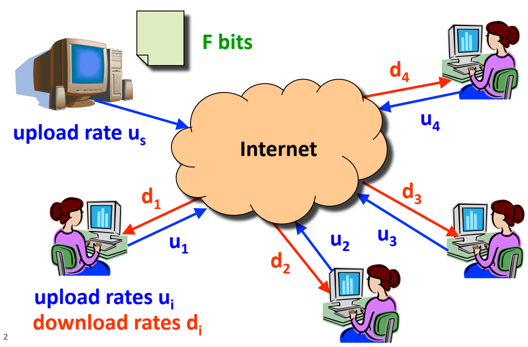


Peer-to-Peer in the Datacenter: Amazon Dynamo

Mike Freedman
COS 461: Computer Networks

http://www.cs.princeton.edu/courses/archive/spr14/cos461/

Last Lecture…



2

This Lecture…

3

Amazon's "Big Data" Problem

- Too many (paying) users!
  - Lots of data
- Performance matters
  - Higher latency = lower "conversion rate"
- Scalability: retaining performance when large

4

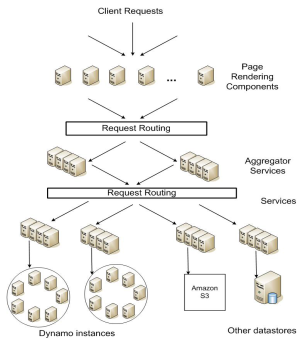
Tiered Service Structure

Stateless

Stateless

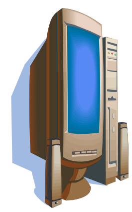
Stateless

All of the State

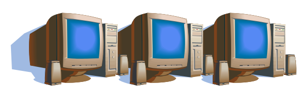


5

Horizontal or Vertical Scalability?



Vertical Scaling



Horizontal Scaling

6

Horizontal Scaling is Chaotic

- k = probability a machine fails in given period
- n = number of machines
- $1-(1-k)^n$ = probability of any failure in given period
- For 50K machines, with online time of 99.99966%:
  - 16% of the time, data center experiences failures
  - For 100K machines, 30% of the time!

7

Dynamo Requirements

- High Availability
  - Always respond quickly, even during failures
  - *Replication!*
- Incremental Scalability
  - Adding "nodes" should be seamless
- Comprehensible Conflict Resolution
  - High availability in above sense implies conflicts

8

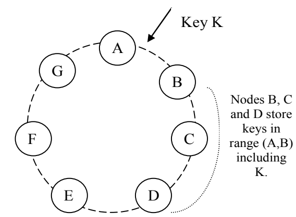
Dynamo Design

- Key-Value Store via DHT over data nodes
  – get(k) and put(k, v)
- Questions:
  – Replication of Data
  – Handling Requests in Replicated System
  – Temporary and Permanent Failures
  – Membership Changes

9

Data Partitioning and Data Replication

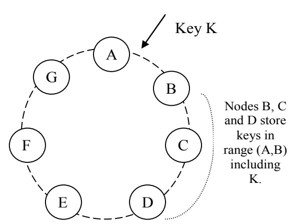
- Familiar?
- Nodes are virtual!
  – Heterogeneity
- Replication:
  – Coordinator Node
  – *N-1* successors also
  – Nodes keep preference list



10

Handling Requests

- Request coordinator consults replicas
  – How many?
- Forward to *N* replicas from preference list
  – *R* or *W* responses form a read/write quorum
- Any of *top N* in pref list can handle req
  – Load balancing & fault tolerance



11

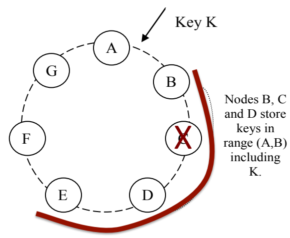
Detecting Failures

- *Purely* Local Decision
  – Node A may decide independently that B has failed
  – In response, requests go further in preference list
- A request hits an unsuspecting node
  – “temporary failure” handling occur

12

Handling Temporary Failures

- E is in replica set
  - Needs to receive replica
  - Hinted Handoff: replica contains "original" node
- When C comes back
  - E forwards the replica back to C



Add E to the replica set!

13

Managing Membership

- Peers randomly tell another their known membership history – "gossiping"
- Also called epidemic algorithm
  - Knowledge spreads like a disease through system
  - Great for ad hoc systems, self-configuration, etc.
  - Does this make sense in Amazon's environment?

14

Gossip could partition the ring

- Possible Logical Partitions
  - A and B choose to join ring at about same time: Unaware of one another, may take long time to converge to one another
- Solution:
  - Use *seed* nodes to reconcile membership views: Well-known peers that are contacted frequently

15

Why is Dynamo Different?

- So far, looks a lot like normal p2p
- Amazon wants to use this for application data!
- Lots of potential synchronization problems
- Uses versioning to provide *eventual consistency.*

16

Consistency Problems

- Shopping Cart Example:
  - Object is a history of "adds" and "removes"
  - *All adds* are important (trying to make money)

Client:	Expected Data at Server:
Put(k, [+1 Banana])	[+1 Banana]
Z = get(k)	
Put(k, Z + [+1 Banana])	[+1 Banana, +1 Banana]
Z = get(k)	
Put(k, Z + [-1 Banana])	[+1 Banana, +1 Banana, -1 Banana]

17

What if a failure occurs?

Client:	Data on Dynamo:
Put(k, [+1 Banana])	[+1 Banana] at A
	A Crashes
Z = get(k)	
Put(k, Z + [+1 Banana])	*B **not** in first Put's quorum*
	[+1 Banana] at B
Z = get(k)	
Put(k, Z + [-1 Banana])	[+1 Banana, -1 Banana] at B
	Node A Comes Online

At this point, Node A and B disagree about object state
- How is this resolved?
- Can we even tell a conflict exists?

18

"Time" is largely a human construct

- What about time-stamping objects?
  - Could authoritatively say whether object newer or older?
  - *But, all events are not necessarily witnessed*
- If system's notion of time corresponds to "real-time"…
  - New object always blasts away older versions
  - Even though those versions may have important updates (as in bananas example).
- Requires a new notion of time (causal in nature)
- Anyhow, real-time is impossible in any case

19

Causality

- Objects are causally related if value of one object depends on (or witnessed) the previous
- Conflicts can be detected when replicas contain causally independent objects for a given key
- Notion of time which captures causality?

20

Versioning

- Key Idea: Every PUT includes a version, indicating most recently witnessed version of updated object
- Problem: replicas may have diverged
  - No single authoritative version number (or "clock" number)
  - Notion of time must use a *partial ordering* of events

21

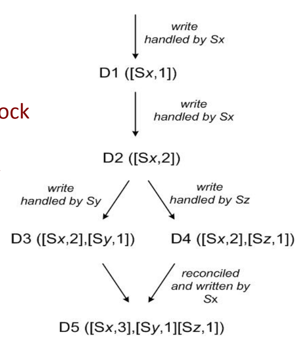
Vector Clocks

- Every replica has its own logical clock
  - Incremented before it sends a message
- Every message attached with *vector* version
  - Includes originator's clock
  - Highest seen logical clocks for each replica
- If M_1 is causally dependent on M_0:
  - Replica sending M_1 will have seen M_0
  - Replica will have seen clocks ≥ all clocks in M_0

22

Vector Clocks in Dynamo

- Vector clock per object
- get() returns obj's vector clock
- put() has most recent clock
  - Coordinator is "originator"
- Serious conflicts are resolved by app / client



23

Vector Clocks in Banana Example

Client:	Data on Dynamo:		
Put(k, [+1 Banana])	[+1]	v=[(A,1)]	at A
	A Crashes		
Z = get(k)			
Put(k, Z + [+1 Banana])	*B **not** in first Put's quorum*		
Z = get(k)	[+1]	v=[(B,1)]	at B
Put(k, Z + [-1 Banana])	[+1,-1]	v=[(B,2)]	at B
	A Comes Online		

[(A,1)] and [(B,2)] are a conflict!

24

Eventual Consistency

- Versioning, by itself, does not guarantee consistency
  - If you don't require a majority quorum, you need to periodically check that peers aren't in conflict
  - How often do you check that events are not in conflict?
- In Dynamo:
  - Nodes consult with one another using a tree hashing (Merkel tree) scheme
  - Quickly identify whether they hold different versions of particular objects and enter conflict resolution mode

25

NoSQL

- Notice that Eventual Consistency and Partial Orderings do not give you ACID!
- Rise of NoSQL (outside of academia)
  - Memcache
  - Cassandra
  - Redis
  - Big Table
  - MongoDB

26