CS 555: Distributed Systems [Dynamo]

Shrideep Pallickara
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

- How can W=1 ever be fault tolerant?
  - Are write-always systems eventually consistent?

Topics covered in this lecture

- Amazon's Dynamo
  - Assumptions & Requirements
  - Design Choices
  - System Architecture
  - Partitioning Algorithm
  - Replication
  - Versioning
  - Experiences

DYNAMO: AMAZON'S HIGHLY AVAILABLE KEY-VALUE STORE

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, Werner Vogels: *Dynamo: Amazon's Highly Available Key-value Store*. SOSP 2007: 205-220

Storage technologies at Amazon

- Simple Storage Service (S3)
- SimpleDB
  - Distributed database
  - Written in *Erlang*
- Dynamo

Dynamo: Highlights

- Completely decentralized system
- Provides a **key-value** store
- Underlying technology for several core services
- **Scales** to extreme peak loads
  - Holiday shopping period
  - No downtimes
  - 3 million checkouts per day

Data from 2007

But why not use an RDBMS?

- Would lead to **inefficiencies**
- Scaling and availability issues
  - ACID Guarantees
  - Difficult to scale-out
  - Difficult to have **smart partitioning**
- Most services only need *primary-key* access
  - No need for complex querying and management
- $$$$$?

Many services in Amazon only need primary-key access to the data store

- Best seller lists
- Shopping carts
- Customer preferences
- Session management
- Product catalog

Techniques used by Dynamo

- Scalability and availability
  - Data partitioned and replicated
  - Consistent **hashing**
- Consistency among replicas
  - Decentralized, **quorum** protocol [**sloppy quorums**, **hinted handoffs**]
- **Gossip** protocols
  - Memberships
  - Failure detection

Dynamo: Primary research contributions

- How different distributed systems' techniques can be combined
- **Eventually consistent** storage can be used in
  - Production & highly-demanding settings

DYNAMO: ASSUMPTIONS & REQUIREMENTS

Dynamo: System Assumptions
Query Model

- read and `write` operations uniquely identified with **key**
- State stored as **binary object** (blob)
- Operations *do not span* multiple data items
  - No need for relational schema
- Target applications store **small objects**
  - Less than 1 MB

Dynamo: System assumptions

ACID {Atomicity, Consistency, Isolation, Durability}

- If data is stored with ACID properties?
  - Poor availability
- Trade-off *consistency* for *availability*
- *Isolation*?
  - Cannot access data modified during a transaction
    - That has **not yet completed**
  - **No** isolation guarantees in Dynamo

Dynamo: System Assumptions
Efficiency

- Must function on **commodity hardware**
- Stringent requirements
  - Latency and throughput
  - Service Level Agreements (SLAs)
- Tradeoff space:
  - Performance, cost efficiency, availability, durability

Clients and services agree on Service Level Agreements (SLAs)

- Example SLA: Provide response
  - Within 300 milliseconds
  - For 99.9% of the requests
  - Up to a peak client load: 500 requests/second
- Rendering page requests in Amazon?
  - Construct response from 150 service requests
  - *Each* service in the call chain must meet contract

DYNAMO: DESIGN CHOICES

Design choices: Why strong consistency is out

- When there is *uncertainty* about data correctness?
  - Data is made unavailable
  - Must be absolutely certain, data is correct
- Not possible to have the **A** in **CAP**

Design considerations: Eventual consistency

- Increase availability using **optimistic** replication
  - Concurrent, disconnected updates allowed
- Conflicting changes must be
  - Deleted
  - Resolved
- **Conflict resolution**
  - When?
  - Who?

Conflict resolution in traditional stores: Done during writes

- Read complexity is kept simple
- Writes may be **rejected** if data store cannot reach majority of the replicas
  - At the same time

Conflict resolution in Dynamo: When?

- Data store must be **always writeable**
  - Rejecting customer updates?
    - Poor customer experience
    - $$$$
- Shopping cart edits must be allowed
  - Even during network and server failures
- Complexity of *conflict resolution pushed to reads*

Conflict resolution in Dynamo: Who?

- Data store?
  - **Last write wins** for conflicting updates
- Application?
  - Aware of the **data schema**
  - Decide on most suitable conflict resolution
- E.g.: Application that maintains shopping carts?
  - **Merge** conflicting versions, and return unified cart

Dynamo: Other guiding design principles

- **Incremental scalability**
  - Scale out <u>one server at a time</u>
- **Symmetry**
  - Every node is a peer
- **Decentralized**
- **Heterogeneity** in infrastructure
  - No need to replace all nodes at same time
  - Add new nodes with higher capacity

DYNAMO SYSTEM ARCHITECTURE

System Interface

- Store objects with a *key*
  - get() and put()
- get(key)
  - Locates objects replicas associated with key
  - Returns single or list of objects
    - Conflicting versions along with **context**

Context encodes system metadata about object

- Includes information about object **version**
- `put(key, context, object)`
  - *Where* should replicas of `object` be placed?
    - Based on 128-bit MD5 hash applied on key
  - Based on key
- Context information stored with the object
  - Used to verify validity of `put` request

PARTITIONING ALGORITHM

A key requirement is that Dynamo must scale incrementally

- *Dynamically partition* data over a set of storage nodes
- Uses **consistent hashing**
  - DHT
  - Data item identified by key
    - Assigned to node responsible for MD5-hash(key)

Basic hashing scheme presents some challenges

- Random position assignment may lead to
  - Non-uniform data and load distribution
- Algorithm **oblivious** to heterogeneity of devices

Dynamo uses a variant of consistent hashing

- Introduces the notion of **virtual nodes**
- Virtual node looks like a real node
- Each node is responsible for more than 1 virtual nodes
  - A node is assigned **multiple positions** in the ring

Advantages of virtual nodes

- If a node becomes *unavailable*
  - **Load** handled by failed node, **dispersed** across remaining virtual nodes
- When node becomes available again
  - Accepts roughly the same amount of work from other nodes
- **Number of virtual nodes** are decided based on machine's capacity

DYNAMO REPLICATION

Dynamo replicates data on multiple hosts

- Each data item is replicated at **N** hosts
- Coordinator is *responsible* for nodes that fall in its range
- Additionally, a coordinator *replicates* key at **N**-1 clockwise **successor** nodes

What does this mean?

- Each node is responsible for region between
  - *Itself* and its Nth *predecessor*
- List of nodes responsible for a key
  - Preference list
- A node maintains a list of more than N to account for failures
  - Account for virtual nodes
    - Make sure your list contains *different* physical nodes

DYNAMO VERSIONING

Data versioning

- A `put()` may return *before* it is applied to all replicas
- If there are no failures
  - **Upper bound** on update propagation times
- If there are failures
  - Things take much longer

There are applications at Amazon that tolerate this

- Shopping carts
- `Add to Cart` can never be forgotten or rejected
- If most recent state of cart unavailable
  - Make changes to the *older* version
  - **Divergent** versions are reconciled later

Dynamo treats each modification as a new, immutable version of the data

- Multiple versions of data present at same time
- Often new versions **subsume** old data
  - *Syntactic* reconciliation
- When an automatic reconciliation is not possible
  - Clients have to do it
  - **Collapse** branches into one
  - Manage your shopping cart

Dynamo uses vector clocks to capture causality

- A vector clock for *each* version of the object
- Two versions of object being compared
  - If $VC_1 <= VC_2$ **for all** indices of the vector clock
    - O_1 occurred before O_2
  - Otherwise, changes are in conflict
    - Need reconciliation

A client must specify which version it is updating

- Pass context from an earlier read operation
  - Context contains vector clock information
- Requests with branches that cannot be reconciled?
  - Returns **all** objects with versioning info in **context**
  - Update done using this context *reconciles* and *collapses* all branches

Execution of get() and put() operations

- Read and write operations involve the first **N** healthy nodes
- During failures, nodes lower in priority are accessed

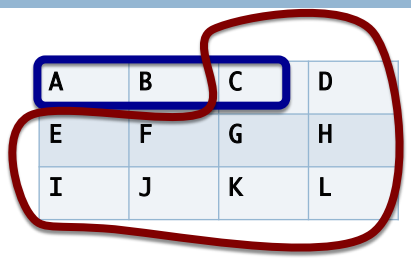
To maintain consistency, Dynamo uses a quorum protocol

- Uses configurable settings for replicas that must participate in
  - Reads
  - Writes

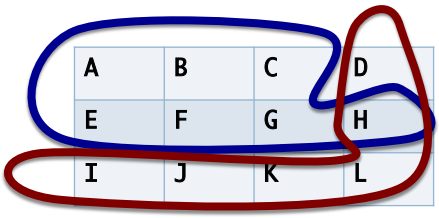
Quorum-based protocols: When there are N replicas

- Read quorum N_R
- To modify a file write-quorum N_W
- $N_R + N_W > N$
  - Prevent **read-write** conflict
- $N_W > N/2$
  - Prevent **write-write** conflict

Quorum-based protocols: Example



N_R=3 N_W=10



N_R=7 N_W=6

Read Quorum:
Write Quorum:

Write-write conflict
Concurrent writes to
{A, B, C, E, F, G} and {D, H, I, J, K, L}
will be accepted

Upon receiving a put() request for a key

- Coordinator generates a **vector clock** for new version
  - Sends new version to **N** highest-ranked reachable nodes
  - If at least N_W-1 nodes respond: write is successful!

External Discovery: During node adds

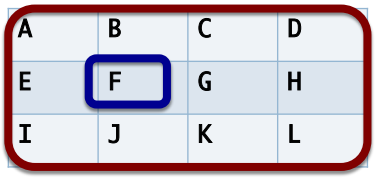
- When **A** and **B** join, it might be a while before they know each other's existence
  - **Logical partitioning**
- Use seed nodes that are known to all nodes
  - All nodes *reconcile* membership with seed

DYNAMO: EXPERIENCES

Popular reconciliation strategies

- Business logic specific
- Timestamp
  - Last write wins
- High performance read engine
  - High read rates
  - Small update rates
    - N_R=1 and N_W=N

Quorum-based protocols: Example 2



N_R=1 N_W=12

Read Quorum:
Write Quorum:

Common configuration of the quorum

- $N_R=2$
- $N_W=2$
- $N=3$

Balancing performance and durability

- Some services not happy with 300 ms SLA
  - Writes tend to be slower than reads
- To cope with this, nodes maintain **object buffer**
  - Main memory
  - *Periodically* written to storage

The contents of this slide-set are based on the following references

- Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, Werner Vogels: *Dynamo: Amazon's Highly Available Key-value Store*. SOSP 2007: 205-220