CS x55: Distributed Systems
[Consistency & Dynamo]

Shrideep Pallickara
Computer Science
Colorado State University

Computer Science Department

Colorado State University

1

Frequently asked questions from the previous class survey

- Why is the read quorum not > N/2?
- How does atomicity differ from isolation in ACID?

2

Topics covered in this lecture

- Eventually Consistent
- Entropy and Anti-entropy
- Amazon's Dynamo
  - Assumptions & Requirements
  - Design Choices
  - System Architecture
  - Partitioning Algorithm
  - Replication
  - Versioning
  - Experiences

EVENTUALLY CONSISTENT

Eventual consistency

- A form of **weak consistency**
- Storage system guarantees that if no new updates are made to the object?
  - **Eventually** all accesses will return last updated value
- If no failures occur, size of the inconsistency window is determined by:
  - Communication delays, system load, and number of replicas

Eventual consistency variations

- Causal consistency
- Read-your-writes consistency
- Session consistency
  - As long as session exists, system guarantees read-your-writes consistency
  - Guarantees *do not overlap* sessions
- Monotonic read consistency
- Monotonic write consistency

RDBMS implement replication in different modes

- **Synchronous**
  - Replica update is part of the transaction
- **Asynchronous**
  - Updates arrive at the backup in a delayed manner
    - **Log shipping**
  - If primary fails before the logs were shipped?
    - Reading from promoted backup will produce old, inconsistent values

Other RDBMS approaches to improve speed

- RDBMSs have also started to provide ability to read from backup
  - Classic case of eventual consistency
- Size of the inconsistency window in such a setting?
  - Periodicity of the log shipping

SERVER SIDE CONSISTENCY

Server-side consistency

- Based on how updates flow through the system
- **N**: Number of nodes that store replicas of data
- **W**: Number of replicas that need to acknowledge receipt of update before it completes
- **R**: Number of replicas that are contacted when data object is accessed through read operation

W+R > N?

- The write-set and read-set overlap
  - Possible to guarantee strong consistency
- Primary-backup RDBMS
  - With synchronous replication
    - N=2, W=2 and R =1
    - Client always reads a consistent answer
  - With asynchronous replication
    - N=2, W=1 and R=1
    - Consistency cannot be guaranteed

In distributed storage systems the number of replicas is higher than two

- Systems that focus on fault tolerance use **N**=3
  - With **W**=2 and **R**=2
- Systems that serve very high read loads
  - Replicate data beyond what is needed for fault tolerance
  - **N** can 10s to 100s of nodes
  - **R** will be set to 1
    - A single read will return the result
  - For consistency **W**=**N** for updates
    - Decreases the probability of write succeeding

For systems concerned about fault tolerance but not consistency

- **W=1**
  - Minimal durability
- Rely on lazy (epidemic) techniques to update other replicas

Configuring values of N, R and W

- Depends on the **common case**
- **Performance path** that needs to be optimized
- If **R**=1 and **N**=**W** ?
  - We optimize for the read case
- If **W**=1 and **R**=**N** ?
  - We optimize for a very fast write
  - Durability is not guaranteed
  - If **W** < (**N**/**2**+1) there is a possibility of conflicting writes when the write-sets do not overlap

Weak/eventual consistency

- Also arises when **W**+ **R** <= **N**
  - Possibility that the read and write set will not overlap
- If it's deliberate and not based on failure cases?
  - Hardly makes sense to set **R** to anything but 1

Weak/eventual consistency: Two common cases where R=1

- Massive replication for read scaling
- When data access is more complicated
  - In simple <key, value> systems easy to compare versions to determine latest written value
  - When set of objects are returned, reasoning gets more complicated

When partitions occur

- Some nodes cannot reach a set of other nodes
- With a classic majority quorum approach
  - Partition that has **W** nodes of the replica set continues to take updates
  - The other partition becomes unavailable

For some applications unavailability of partitions is unacceptable

- Important that clients, that reach a partition, can progress
- Merge operation is executed when partition heals
- Amazon shopping-cart?
  - **Write-always** system
  - Customer can continue to put items in the cart even when original cart lives on other partitions

ANTI-ENTROPY

COMPUTER SCIENCE DEPARTMENT
COLORADO STATE UNIVERSITY

19

Entropy

- Entropy is a property that represents the **measure of disorder** in the system
- In a distributed system, entropy represents a *degree of state divergence* between the nodes
- Since this property is undesired and its amount should be kept to a *minimum*, there are many techniques that help to deal with entropy

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT EVENTUAL CONSISTENCY & DYNAMO L36.20

20

Anti-entropy

- **Anti-entropy** is usually used to bring the nodes back up-to-date in case the primary delivery mechanism has failed
- The system can continue functioning correctly even if the coordinator fails at some point
  - Since the other nodes will continue spreading the information
- In other words, anti-entropy is used to **lower the convergence time bounds** in eventually consistent systems

Anti-entropy: How?

- To keep nodes in sync, anti-entropy triggers a **background or a foreground process** that compares and reconciles missing or conflicting records
- Background anti-entropy processes use auxiliary structures such as Merkle trees and update logs to identify divergence
- Foreground anti-entropy processes piggyback read or write requests: hinted handoff, read repairs, etc.

Hinted-handoff: An anti-entropy approach

- A **write-side repair** mechanism
- If the target node fails to acknowledge the write, the write coordinator or one of the replicas stores a special record, called a **hint**
- The **hint** is replayed to the target node as soon as it comes back up
  - Hinted writes *aren't counted* toward the replication factor
    - Since the data in the hint log isn't accessible for reads and is only used to help the lagging participants catch up

Sloppy-quorums

- With sloppy quorums, in case of replica failures, write operations can **use additional healthy nodes** from the node list
- And … these nodes do not have to be target replicas for the executed operations

Sloppy-quorums: An example

- Say we have a five-node cluster with nodes {**A, B, C, D, E**}, where {**A, B, C**} are replicas for the executed write operation, and node **B** is down
  1. **A**, being the coordinator for the query, picks node **D** to *satisfy the sloppy quorum* and maintain the desired availability and durability guarantees
  2. Now, data is replicated to {**A, D, C**}.
  3. However, the record at **D** will have a hint in its metadata, since the write was originally intended for **B**
  4. As soon as **B** recovers, **D** will attempt to *forward a hint* back to it
  5. Once the **hint is replayed** on **B**, it can be safely removed at **D** without reducing the total number of replicas

25

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

DYNAMO: AMAZON'S HIGHLY AVAILABLE KEY-VALUE STORE

26

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, and durability

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT EVENTUAL CONSISTENCY & DYNAMO L36.33

33

Clients and services agree on Service Level Agreements (SLAs)

- Example SLA: Provide response
  - Within 300 milliseconds
  - For 99.9% of the requests
- Rendering page requests in Amazon?
  - Construct response from 150 service requests
  - *Each* service in the call chain must meet contract

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT EVENTUAL CONSISTENCY & DYNAMO L36.34

34

DYNAMO: DESIGN CHOICES

COMPUTER SCIENCE DEPARTMENT COLORADO STATE UNIVERSITY

35

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**

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT EVENTUAL CONSISTENCY & DYNAMO L36.36

36

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 one server at a time
- **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

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT EVENTUAL CONSISTENCY & DYNAMO L36.41

41

DYNAMO SYSTEM ARCHITECTURE

COMPUTER SCIENCE DEPARTMENT COLORADO STATE UNIVERSITY

42

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**

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT EVENTUAL CONSISTENCY & DYNAMO L36.43

43

Context encodes system metadata about object

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

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT EVENTUAL CONSISTENCY & DYNAMO L36.44

44

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 <u>more than 1</u> 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 N^{th} *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

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
- Alex Petrov. Database Internals. O'Reilly Media. ISBN-13: 978-1492040347. 1st edition. 2019. [Chapter 12]
- Werner Vogels: Eventually Consistent. ACM Queue 6(6): 14-19 (2008)
- Martin Kleppmann. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. 1st Edition. O'Reilly Media. 2017. [Chapter 9]