Recap

• CAP Theorem?
  – Consistency, Availability, Partition Tolerance
  – Pick two

• Eventual consistency?
  – Availability and partition tolerance over consistency

• Lazy replication?
  – Replicate lazily in the background

• Gossiping?
  – Contact random targets, infect, and repeat in the next round

Amazon Dynamo

• Distributed key-value storage
  – Only accessible with the primary key
  – put(key, value) & get(key)

• Used for many Amazon services (“applications”)
  – Shopping cart, best seller lists, customer preferences, product catalog, etc.
  – Now in AWS as well (DynamoDB) (if interested, read http://www.allthingsdistributed.com/2012/01/amazon-dynamodb.html)

• With other Google systems (GFS & Bigtable), Dynamo marks one of the first non-relational storage systems (a.k.a. NoSQL)

Overview of Key Design Techniques

• Gossiping for membership and failure detection
  – Eventually-consistent membership

• Consistent hashing for node & key distribution
  – Similar to Chord
  – But there’s no ring-based routing; everyone knows everyone else

• Object versioning for eventually-consistent data objects
  – A vector clock associated with each object

• Quorums for partition/failure tolerance
  – “Sloppy” quorum similar to the available copies replication strategy

• Merkle tree for resynchronization after failures/partitions
  – (This was not covered in class)

Membership

• Nodes are organized as a ring just like Chord using consistent hashing

• But everyone knows everyone else.

• Node join/leave
  – Manually done
  – An operator uses a console to add/delete a node
  – Reason: it’s a well-maintained system; nodes come back pretty quickly and don’t depart permanently most of the time

• Membership change propagation
  – Each node maintains its own view of the membership & the history of the membership changes
  – Propagated using gossiping (every second, pick random targets)

• Eventually-consistent membership protocol

Amazon Dynamo

• A synthesis of techniques we discuss in class
  – Well, not all but mostly
  – Very good example of developing a principled distributed system
  – Comprehensive picture of what it means to design a distributed storage system

• Main motivation: shopping cart service
  – 3 million checkouts in a single day
  – Hundreds of thousands of concurrent active sessions

• Properties (in the CAP theorem sense)
  – Eventual consistency
  – Partition tolerance
  – Availability (“always-on” experience)
**Failure Detection**

- Does not use a separate protocol; each request serves as a ping
  - Dynamo has enough requests at any moment anyway
- If a node doesn’t respond to a request, it is considered to be failed.

**Node & Key Distribution**

- Original consistent hashing
- Load becomes uneven

**Node & Key Distribution**

- Consistent hashing with “virtual nodes” for better load balancing
- Start with a static number of virtual nodes uniformly distributed over the ring

**Node & Key Distribution**

- One node joins and gets all virtual nodes

**Node & Key Distribution**

- One more node joins and gets 1/2

**Node & Key Distribution**

- One more node joins and gets 1/3 (roughly) from the other two
Replication

- N: # of replicas, configurable
- The first is stored regularly with consistent hashing
- N-1 replicas are stored in the N-1 (physical) successor nodes (called preference list)

Object Versioning

- Writes should succeed all the time
  - E.g., “Add to Cart”
- Used to reconcile inconsistent data due to network partitioning/failures
- Each object has a vector clock
  - E.g., D1 \( [S_x, 1], [S_y, 1] \): Object D1 has written once by server \( S_x \) and \( S_y \).
  - Each node keeps all versions until the data becomes consistent
- Causally concurrent versions: inconsistency
- If inconsistent, reconcile later.
  - E.g., deleted items might reappear in the shopping cart.

Object Versioning Experience

- Over a 24-hour period
  - 99.94% of requests saw exactly one version
  - 0.00057% saw 2 versions
  - 0.00047% saw 3 versions
  - 0.00009% saw 4 versions
  - Usually triggered by many concurrent requests issued busy robots, not human clients
Quorums

- Parameters
  - N replicas
  - R readers
  - W writers
- Static quorum approach: R + W > N
- Typical Dynamo configuration: (N, R, W) == (3, 2, 2)
- But it depends
  - High performance read (e.g., write-once, read-many): R=1, W=N
  - Low R & W might lead to more inconsistency
- Dealing with failures
  - Another node in the preference list handles the requests temporarily
  - Delivers the replicas to the original node upon recovery

Replica Synchronization

- Key ranges are replicated.
- Say, a node fails and recovers, a node needs to quickly determine whether it needs to resynchronize or not.
  - Transferring entire (key, value) pairs for comparison is not an option
- Merkel trees
  - Leaves are hashes of values of individual keys
  - Parents are hashes of (immediate) children
  - Comparison of parents at the same level tells the difference in children
  - Does not require transferring entire (key, value) pairs

Summary

- Amazon Dynamo
  - Distributed key-value storage with eventual consistency
- Techniques
  - Gossiping for membership and failure detection
  - Consistent hashing for node & key distribution
  - Object versioning for eventually-consistent data objects
  - Quorums for partition/failure tolerance
  - Merkel tree for resynchronization after failures/partitions
- Very good example of developing a principled distributed system

Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).