PART 2.
DATA STORAGE AND FLOW MANAGEMENT

Instructor: Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs435

Today’s topics
- FAQs
- Dynamo

FAQs
- Help session for PA3
  - April 8th 11:00AM - noon CSB130
  - Please check the assignment page for the link

Key Value Stores: Dynamo
(1) Partitioning
(2) High Availability for writes
(3) Handling temporary failures
(4) Recovering from permanent failures
(5) Membership and failure detection

Scalable Key location in Chord
- Let \( m \) be the number of bits in the key/node identifiers
- Each node \( n \), maintains,
  - A routing table with (at most) \( m \) entries
  - Called the finger table
- The \( i \)th entry in the table at node \( n \), contains the identity of the first node, \( y \),
  - Succeeds \( n \) by at least \( 2^i \) on the identifier circle
  - I.e., \( x = \text{successor}(n+2^i) \), where \( i \)thion (and all arithmetic is modulo \( 2^m \))

Chord
Definition of variables for node \( n \), using \( m \)-bit identifiers

- \( \text{finger}[i].\ start = (n + 2^i) \mod 2^m \), \( 1 \leq i \leq m \)
- \( \text{finger}[i].\ interval = [\text{finger}[i].\ start, \text{finger}[i+1].\ start] \)
- \( \text{finger}[i].\ start = \text{finger}[i].\ interval < \text{finger}[i+1].\ start \)
- \( \text{finger}[i].\ node = \text{first node that is greater than or equal to } n.\text{finger}[i].\ start \)

Finger tables

- Each node stores information about only a small number of other nodes
  - A node’s finger table generally (for a large cluster) does not contain enough information to determine the successor of an arbitrary key \( k \)
- What happens when a node \( n \) does not know the successor of a key \( k \)?
  - If \( n \) finds a node whose ID is closer than its own to \( k \), that node will know more about the identifier circle in the region of \( k \) than \( n \) does.

Lookup process (1/3)

- \( n \) searches its finger table for the node \( j \)
  - Whose ID most immediately precedes \( k \)
  - Ask \( j \) for the node it knows whose ID is closest to \( k \)

1. Go clockwise
2. Never overshoot

Lookup process (2/3)

- \( n \) searches its finger table for the node \( j \)
  - Whose ID most immediately precedes \( k \)
  - Ask \( j \) for the node it knows whose ID is closest to \( k \)

1. Go clockwise
2. Never overshoot

Lookup process (3/3)
Lookup process: example 1

0. Request comes into node (machine) 1 to find the successor of id 4. Check the node 1 itself first.
1. Node 3 wants to find the successor of identifier 4.
2. Identifier 4 belongs to [3,5).
3. Pass to succ: 3.
4. Check the node 3 itself.
5. Successor of 4 is 0.

What if node 0 is overloaded?

4. Node 3 asks node 0 to find successor of 1.
5. This node stores id 0: succ is 0.

Dynamo’s partitioning

- Inspired by Consistent Hashing and Chord
- Repeated reconciliation of the membership change
- Partitioning and placement information are propagated via the gossip-based protocol
- Direct forwarding of read/write operations are possible

Key Value Stores: Dynamo

1. Partitioning
2. High Availability for writes
3. Handling temporary failures
4. Recovering from permanent failures
5. Membership and failure detection

Replication (1/3)

- Dynamo replicates its data on multiple hosts
  - Each data item is replicated at $R$ hosts, where $R$ is a parameter configured "per-instance"
- Each key $k$ is assigned to a coordinator node
  - The coordinator is managing the replication of the data items that fall within its range
  - Stores at the ($R+1$) clockwise successor nodes in the ring
  - Each node is responsible for the region of the ring between it and its $R$th predecessor
- More availability for the “read” operations
Replication (2/3)

Machine D will store the keys, \([A,B), [B,C),\) and \([C,D)\)

Replication (3/3)

- Preference list
  - The list of nodes that is responsible for storing a particular key
- Node failures
  - Preference list contains more than \(R\) nodes
- Virtual nodes can reduce actual number of machines in \(R\) nodes
  - The preference list for a key is constructed only by distinct physical nodes

Data Versioning

- Dynamo provides eventual consistency
  - Allows for updates to be propagated to all replicas asynchronously
  - A `put` call may return to its caller before the update has been applied to all the replicas
  - More availability for the “write” operations

“Add to Cart” example

- The shopping cart application requires that an “Add to Cart” operation can never be forgotten or rejected
  - If the most recent cart is not available and user makes changes to an old version of the cart
    - Still the change should be meaningful and preserved
  - “add to cart” and “delete item from cart” should be translated into `put` operation to Dynamo
  - The divergent versions are reconciled later

Maintaining vector clock

- Dynamo treats the result of each modification as a new and immutable version of the data
  - Multiple version of data can be in the system
- Version branching
  - Due to the failure(s) in node(s), there are conflicting versions of an object
- Merging
  - Collapses multiple branches of data evolution back into one
  - Semantic reconciliation
  - e.g. merging different versions of shopping cart and preserving all of the items those client put into the cart

Vector clocks

- Used to capture causality between different versions of same object
  - Two versions of object are on parallel branches or have a causal ordering
- Vector clock
  - A list of (node, counter) pairs
  - One vector clock is associated with every version of every object
Definition of the vector clocks

- $VC(x)$ denotes the vector clock of event $x$
- $VC(x)_z$ denotes the component of that clock for process $z$
- $x \rightarrow y$ denotes that event $x$ happens before event $y$
- If $x \rightarrow y$, then $VC(x) < VC(y)$

\[ VC(x) < VC(y) \iff \forall z [ VC(x)_z \leq VC(y)_z ] \land \exists z' [ VC(x)_z < VC(y)_{z'} ] \]

Examples

- $VC(D1)=(Sx,3, Sy,2, Sz,2, Sq,2)$
- $VC(D2)=(Sx,3, Sy,2, Sz,2, Sq,1)$
- $VC(D3)=(Sx,3, Sy,2, Sz,2, Sq,1)$
- $VC(D4)=(Sx,3, Sy,3, Sz,2, Sq,1)$

- Is $VC(D1) > VC(D2)$?  YES
- Is $VC(D3) > VC(D2)$?  NO
- Is $VC(D4) > VC(D2)$?  YES

Examples [Answers]

- $VC(D1)=(Sx,3, Sy,2, Sz,2, Sq,2)$
- $VC(D2)=(Sx,3, Sy,2, Sz,2, Sq,1)$
- $VC(D3)=(Sx,3, Sy,2, Sz,2, Sq,1)$
- $VC(D4)=(Sx,3, Sy,3, Sz,2, Sq,1)$

- Is $VC(D1) > VC(D2)$?  YES
- Is $VC(D3) > VC(D2)$?  NO
- Is $VC(D4) > VC(D2)$?  YES

Properties of the vector clocks

- If $VC(a) < VC(b)$, then $a \rightarrow b$

- Antisymmetry:
  - If $VC(a) < VC(b)$ then NOT $VC(b) < VC(a)$

- Transitivity
  - If $VC(a) < VC(b)$ and $VC(b) < VC(c)$, then $VC(a) < VC(c)$

Execution of get and put operations

- Users can send the operations to any storage node in Dynamo

- Coordinator
  - A node handling a read or write operation
  - The top $N$ nodes in the preference list

- Client can select a coordinator
  - Route its request through a generic load balancer
  - Use a partition-aware client library
  - Directly access the coordinators
Using quorum-like system

- **R**
  - Minimum number of nodes that must participate in a successful read operation

- **W**
  - Minimum number of nodes that must participate in a successful write operation

- General rule of setting R and W for the degree of replication N
  - \( R + W > N \)
  - \( W > N/2 \)

- The latency of a get (or put) operation is dictated by the slowest of the R (or W) replicas
  - R and W are configured to be less than N

**put request**

- Coordinator node
  1. Generates the vector clock
     -- For the new version
  2. Writes the new version locally
  3. Sends the new version to the N highest-ranked reachable nodes
     -- Along with the new vector clock

**get request**

- The coordinator requests all existing versions of data for that key from the N highest-ranked reachable nodes
  - In the preference list
  - Waits for R responses
  - If multiple versions of the data are collected
    - Returns all the versions it deems to be causally unrelated
  - The reconciled version superseding the current versions is written back

**Key Value Stores: Dynamo**

1. Partitioning
2. High Availability for writes
3. Handling temporary failures
4. Recovering from permanent failures
5. Membership and failure detection

**Sloppy quorum**

- All read and write operations are performed on the first N healthy nodes from the preference list
  - May not always be the first N nodes on the hashing ring

- **Hinted handoff**
  - If a node is temporarily unavailable, data is propagated to the next node in the ring
  - Metadata contains information about the originally intended node
  - Stored in a separate local database and scanned periodically

- Upon detecting that the original node is recovered,
  - A data delivery attempt will be made
  - Once the transfer succeeds, the data at the temporary node will be removed

**Example**

- If C is temporarily down
  - The data will be sent to the node D
  - After the recovery, D will send data to C
  - Node where it was supposed to be will be removed

  

This data contains a hint in the metadata -- node where it was supposed to be sent.

Then, it will remove the data.
What if $W$ is 1?

- Applications that need the highest level of availability can set $W$ as 1
  - Under Amazon’s model
- A write is accepted as long as a single node in the system has durably written the key to its local store
- A write request is rejected,
  - Only if all nodes in the system are unavailable

Key Value Stores: Dynamo

- Partitioning
- High Availability for writes
- Handling temporary failures
- Recovering from permanent failures
- Membership and failure detection

Anti-entropy protocol

- Replica synchronization protocol
- Hinted replica can be lost before they can be returned to the original replica node
- Detect inconsistencies between the replicas faster
- Minimize the amount of transferred data
- Dynamo uses Merkle tree

Merkle tree

- Hash tree where leaves are hashes of the values of individual keys
  - Parent nodes are hashes of their respective children
- Each branch of the tree can be checked independently
  - Without requiring nodes to download the entire tree or dataset
- If the hash values of the root of two trees are equal
  - The values of the leaf nodes in the tree are equal
  - No synchronization needed

Uses of Merkle tree

- Merkle trees can be used to verify any kind of data stored, handled and transferred
  - Used in a peer-to-peer network
- Trusted computing systems
  - Sun’s ZFS (Zeta File System)
- Google’s Wave protocol
- Git
- Cassandra and Dynamo
- BitTorrent protocol

How Merkle tree works
How Dynamo uses Merkle tree

- Each node maintains a separate Merkle tree for each key range
- Two nodes exchange the root of the Merkle tree
- Corresponding to the key ranges that they host in common
- Node performs tree traversal and determines if there is any difference
- Perform the appropriate synchronization action

Disadvantage
- When a new node joins or leaves
  - Tree needs to be recalculated

How Merkle tree works for Dynamo

1. Compare Top hashes
2. If top hash did not match
   Compare H5 vs. H13 and H6 vs. H14
3. If H6 and H14 did not match
   Compare H3 vs. H12 and H4 vs. H13

Key Value Stores: Dynamo

(1) Partitioning
(2) High Availability for writes
(3) Handling temporary failures
(4) Recovering from permanent failures
(5) Membership and failure detection

Ring Membership

- A node outage should not result in re-balancing of the partition assignment or repair of the unreachable replicas
  - A node outage is mostly temporary
- Gossip-based protocol
  - Propagates membership changes
  - Maintains an eventually consistent view of membership
- Each node contacts a peer every second
  - Random selection
  - Two nodes reconcile their persisted membership change history

Logical partitioning

- Almost concurrent addition of two new nodes
  - Node A joins the ring
  - Node B joins the ring
- A and B consider themselves members of the ring
  - Yet neither would be immediately aware of each other
  - Logical partitioning

External Discovery

- Seeds
  - Discovered via an external mechanism
  - Known to all nodes
  - Statically configured (or from a configuration service)
- Seed nodes will eventually reconcile their membership with all of the nodes
Communication failure

- Attempts to
  - Communicate with unreachable peers during a get or put operation
  - Transfer partitions and hinted replicas

- Detecting communication failures
  - When there is no response to an initiated communication

- Responding to communication failures
  - Sender will try alternate nodes that map to failed node’s partitions
  - Periodically retry failed node for recovery