FAQs

• Snapshots and locks
• Vector Clock

Example 1: Designing locks for **read**

- **Example files**
  - `/home/cs_bookclub/Robin_Hood`
  - `/home/cs_bookclub/Killing_a_Mockingbird`

- **Requirement**: "While I am reading `/home/cs_bookclub/Killing_a_Mockingbird`, others can read the same book."

- **Read operation should obtain read locks on**:
  - `/home`
  - `/home/cs_bookclub`
  - `/home/cs_bookclub/Killing_a_Mockingbird`

- Any number of read operations can access `/home/cs_bookclub/Killing_a_Mockingbird` concurrently.

- If a user tries to append a text to `/home/cs_bookclub/Killing_a_Mockingbird`, read locks on `/home`, and `/home/cs_bookclub` will be required → allowed.

- Therefore, the write/append operation should wait until all of the reading locks are released.

Example 2: Designing locks for **write**

- **Example files**
  - `/home/cs_bookclub/Robin_Hood`
  - `/home/cs_bookclub/Killing_a_Mockingbird`

- **Requirement**: "While I am writing `/home/cs_bookclub/The_Black_Cat`, others cannot write a file with the same name."

- **Write operation should obtain read locks on**:
  - `/home`
  - `/home/cs_bookclub`

- **And write lock on**:
  - `/home/cs_bookclub/The_Black_Cat`

- No read operation to `/home/cs_bookclub/The_Black_Cat` will be allowed.

- No write operation to `/home/cs_bookclub/The_Black_Cat` will be allowed.

- Other files under `/home/cs_bookclub` will be allowed to read.

Example 3: Designing locks for **snapshot**

- **Example files**
  - `/home/cs_bookclub/Robin_Hood`
  - `/home/cs_bookclub/Killing_a_Mockingbird`

- **Requirement**: "While I am snapshotting `/home/cs_bookclub`, there should be no change in this directory."

- **Read locks on**:
  - `/home`
  - `/home/cs_bookclub` → This will not work. New file can be added under this directory. File can be modified under this directory as well.

- **Write lock on**:
  - `/home/cs_bookclub` → This will prevent any modification under this directory.

Example 4: Designing locks for **snapshot**

- **Example files**
  - `/home/cs_bookclub/Robin_Hood`
  - `/home/cs_bookclub/Killing_a_Mockingbird`

- **Requirement**: "While I am snapshotting `/home/cs_bookclub/Robin_Hood`, there should be no change in this file."

- **Read locks on**

- **Write lock on**
Example 4: Designing locks for snapshot. -- Answers

- Example files
  - /home/cs_bookclub/Robin_Hood
  - /home/cs_bookclub/Killing_a_Mockingbird

- Requirement: “While I am snapshotting /home/mybooks/ Robin_Hood, there is no change in this file.”
  - Read locks on: (preventing deletion of these directory)
    - /home
    - /home/cs_bookclub
  - Write lock on:
    - /home/cs_bookclub/Robin_Hood ➔ This will prevent any modification in this file
    - /home/cs_bookclub/Robin_Hood ➔ Read lock will also work for preventing possible modification

Today’s topics

- No SQL storage: Dynamo

Definition of the vector clocks

- VC(x) denotes the vector clock of event x
- VC(x), denotes the component of that clock for process z

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

- x → y denotes that event x happens before event y
- If x → y, then VC(x) < VC(y)

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
- Setting \( R \) and \( W \) for the given replication factor of \( N \)
  - \( R + W > N \)
  - \( W > N/2 \)
- The latency of a get (or put) operation is dictated by the slowest one 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
  - Here, \( R \) is the read quorum
- If multiple versions of the data are collected
  - Return all the versions it deems to be causally unrelated
- The reconciled version superseding the current versions is written back

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

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

The data will be sent to the node D
If C is temporarily down
This data contains a hint in its metadata
-- node where it was supposed to be stored
After the recovery, D will send data to C
Then, it will remove the information.

Part 2. Large scale data storage system

NoSQL Storage: 1. 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

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
- Block chain

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

Example

- We are trying to build a Merkle Tree with 16 blocks
  - Replication factor is 3, each storage node will maintain 3 Merkle Trees for each of replication based on the data block's original node identifier
  - Suppose that storage server nodes, ra, rb, and rc store replications of the same data blocks
  - How many hash values does each Merkle tree contain? (including the root hash)
  - What will be the maximum number of comparison(s) of hash values to detect the complete set of corrupted data blocks? (including the comparison of root hashes)

Example - answers

- We are trying to build a Merkle Tree with 16 blocks
  - Replication factor is 3, each storage node will maintain 3 Merkle Trees for each of replication based on the data block's original node identifier
  - Suppose that storage server nodes, ra, rb, and rc store replications of the same data blocks
  - How many hash values does each Merkle tree contain? (including the root hash)- 31 hash values
  - If there is a mismatch between root hashes of ra and rb, what will be the maximum number of comparison(s) of hash values to detect the complete set of corrupted data blocks? (including the comparison of root hashes)- 31 comparisons

Part 2. Large scale data storage system

NoSQL Storage: 1. 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

Identifier “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
  - 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
  - A does not know the existence of B
  - Logical partitioning

External Discovery

- Addresses the logical partitioning
- 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

Failure Detection

- Attempts to
  - Avoid communication 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

Part 2. Large scale data storage system

NoSQL Storage: 2. Column Family Stores
Google’s Big Table

This material is built based on,


Column-family storage

- Optimized for the data
  - Sparse columns and no schema
- Aggregate-oriented storage
  - Most data interaction is done with the same aggregate
  - Aggregate
    - A collection of data that we interact with as a unit
- Stores groups of columns (column families) together
get('1234','Profile:name')