Today's topics

- No SQL storage

This material is built based on,


Overview of Dynamo

- Partitions and replicates data using consistent hashing
- Tracks object version to provide consistency
- Uses quorum-like technique to ensure consistency among replicas
- Uses a decentralized synchronization protocol
  - Storage nodes can be added and removed from Dynamo without any manual partitioning or redistribution
- Gossip based distributed failure detection and membership protocol
System Assumptions (1/2)

- **Query model**
  - Simple read and write operations to a data item
  - Uniquely identified by a key
  - State is stored as binary objects (i.e. blobs) identified by unique keys
  - No operations span multiple data items
- **ACID Properties**
  - Dynamo targets applications that operate with weaker consistency if this results in high availability
  - Dynamo does not provide any isolation guarantees

System Assumptions (2/2)

- **Efficiency**
  - The system needs to function on a commodity hardware infrastructure
  - Stringent latency requirements

Summary of techniques

<table>
<thead>
<tr>
<th>Problem</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Partitioning</td>
<td>Consistent Hashing</td>
</tr>
<tr>
<td>(2) High Availability for writes</td>
<td>Vector clocks with reconciliation during reads</td>
</tr>
<tr>
<td>(3) Handling temporary failures</td>
<td>Slippy Quorum and hinted handoff</td>
</tr>
<tr>
<td>(4) Recovering from permanent failures</td>
<td>Anti-entropy using Merkle tree</td>
</tr>
<tr>
<td>(5) Membership and failure detection</td>
<td>Gossip-based membership protocol and failure detection</td>
</tr>
</tbody>
</table>

Partitioning algorithm

- Dynamically partitions the data over the set of nodes
- Distributes the load across multiple storage hosts
- Dynamo uses **consistent hashing**

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
**Non-consistent hashing vs. consistent hashing**

- When a hash table is resized
  - Non-consistent hashing algorithm requires to re-hash complete table
  - Consistent hashing algorithm requires only partial records of the table

**Consistent hashing (1/3)**

Consistent hash function assigns each node and key an m-bit identifier using a hashing function

Hashing value of IP address

**Consistent hashing (2/3)**

Consistent hashing assigns keys to nodes:
- Key \( k \) will be assigned to the first node whose identifier is equal to or follows \( k \) in the identifier space

**Consistent hashing (3/3)**

- If machine C leaves circle, \( \text{successor}(5) \) will point to A
- If machine N joins circle, \( \text{successor}(2) \) will point to N

**Scalable Key location**

- In consistent hashing:
  - Each node needs to be aware of its successor node on the circle
  - Queries can be passed around the circle via these successor pointers until it finds resource
- What is the disadvantage of this scheme?
  - It may require traversing all \( N \) nodes to find the appropriate mapping
Dynamo’s partitioning

- **Inspired by Consistent Hashing and Chord**
  - Zero-hop DHT
- When a node starts for the first time
  - Chooses its set of tokens (virtual nodes in the consistent hash space)
  - Maps nodes to their respective token sets
  - Stores both tokens and nodes onto disk
- Repeated reconciliation of the membership change
  - Partitioning and placement information are propagated via the gossip-based protocol
  - Token ranges handled by its peers
  - Direct forwarding of read/write operations are possible

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 $R$-th clockwise successor nodes in the ring
  - Each node is responsible for the region of the ring between it and its $R$-th predecessor

Replication (2/3)

- Preference list
  - The list of nodes that is responsible for storing a particular key
- If there are 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
“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
- "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), denotes the component of that clock for process z

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

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

Examples

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

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

- Setting \( R \) and \( W \) for the given replication factor of \( N \)
  - \( R = \lceil \frac{N}{2} \rceil \)
  - \( W > \frac{N}{2} \)

- 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**

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

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
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.

- Handed off:
  - 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, where it was supposed to be stored.
- After the recovery, D will send data to C, then remove the data.

What if \( W = 1 \)?

- 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.