Dynamo Recap

● Consistent hashing
  ◆ 1-hop DHT enabled by gossip

● Execute reads and writes using sloppy quorums
  ◆ Coordinated by first available successor of key
  ◆ Read from/write to N successors
  ◆ Declare success if R reads or W writes succeed

● Rethink vector clocks to identify causality
  ◆ Vector of (coordinator, count) pairs
Dealing with Failures

- **Permanent failures:**
  - Identified and marked manually
  - Discovered via gossip

- **Temporary failures:**
  - Handled by sloppy quorums
  - What if less than $W$ of $N$ replicas are up?
  - How does a node catch up once it recovers?
Hinted Handoff

- Suppose coordinator doesn’t receive \( W \) replies from \( N \) successors
  - Could return failure
  - But, want to maximize availability

- Coordinator \textit{tries writing to subsequent nodes} (beyond \( N \) successors of key)
- Coordinator \textit{informs recipient of intended node}
Hinted Handoff: Example

- Say $N = 3$, $W = 3$
- Hinted Handoff:
  - B writes to itself, D and E
  - E points to node C
- When C is available again
  - E forwards the replicated data back to C
Removing threats to durability

- Hinted handoff node **may crash before it replicates data** to appropriate node
  - Need to ensure that each key-value pair is replicated N times

- Solution: **Replica synchronization**
  - Nodes nearby on ring periodically **gossip**
    » **Compare** the (k, v) pairs they hold
    » **Copy** any missing keys the other has

- How to efficiently sync between two nodes?
How to sync, quickly?

- B tells A: highest timestamp for every node
  - e.g., “X 30, Y 40” ← Version vector
- What if every node stores only (key, value) map?
- How to efficiently sync?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>⟨U₁, 10, X⟩</td>
<td>⟨U₁, 10, X⟩</td>
</tr>
<tr>
<td>⟨U₂, 20, Y⟩</td>
<td>⟨U₂, 20, Y⟩</td>
</tr>
<tr>
<td>⟨U₃, 30, X⟩</td>
<td>⟨U₃, 30, X⟩</td>
</tr>
<tr>
<td>⟨U₄, 40, X⟩</td>
<td>⟨U₅, 40, Y⟩</td>
</tr>
</tbody>
</table>
Efficient synchronization with Merkle trees

- Merkle tree hierarchically summarizes the key-value pairs a node holds

- Exchange and compare hash nodes from root downwards, pruning when hashes match

A’s values:
- $[0, 2^{128})$
- $[0, 2^{127})$
- $[2^{127}, 2^{128})$

B’s values:
- $[0, 2^{128})$
- $[0, 2^{127})$
- $[2^{127}, 2^{128})$
Load Balancing

What needs to happen to bootstrap N72?

Copy \((k,v)\) pairs from N90 to N72
Initializing New Server

kv-store map[string]string

InitServer(nodeID int64) {
    s = get_successor(hash(ID))

    // send successor your ID and ask
    // for keys that now belong to you
    call(s, "Transfer", ID, &kv-store)
}
Handing Off to New Server

\[\text{kv-store} \text{ map}[\text{string}]\text{string}\]

\text{Transfer}(\text{ID} \text{ int64}, \text{ shard map}[\text{str}][\text{str}]) \{ \\
\text{for key, val := range kv-store} \{ \\
\text{if hash(key) < hash(ID)} \{ \\
\text{shard[key] = value} \\
\text{delete(kv-store, key)} \\
\} \\
\}
\}

October 24, 2019 EECS 491 – Lecture 15
Revisiting Consistent Hashing

- What needs to happen to bootstrap N72?

Root cause of problems:

Consistent hashing determines both partitioning and placement

Node addition/removal affects both how data is partitioned and where partitions are placed

- Worse with virtual nodes

- Need to recompute Merkle trees
Decoupling Partitioning and Placement

- Partition hash space statically into equal sized shards
- Place shard on first N virtual nodes after its end
- Need to identify only which shards to hand off
- Maintain Merkle tree per shard
Dynamo Summary

● Scalability and low latency:
  ◆ 1-hop DHT enabled by inter-node gossip

● High availability:
  ◆ Sloppy quorums, hinted handoff

● Eventual consistency:
  ◆ Vector clocks, Merkle trees

● Load balancing:
  ◆ Decoupling partitioning and placement of data
Impact of Dynamo

- **NoSQL systems**
  - Eventual consistency popularized by Dynamo
  - Better scalability than strongly consistent systems

- Frustration over application development complexity to offer intuitive user experience

- Recent trend: **Scalable strong consistency**
  - Example: Google’s Spanner
Project 3

- You’ll reuse Paxos and PaxosRSM in Project 4
  - Ensure no KVPaxos-specific logic in either

- Things to watch out for in KVPaxos/PaxosRSM
  - Min() is not lowest undecided instance
  - seq <= Max() could be undecided
  - Max() can differ across peers

- Make sure to garbage collect KVPaxos state for detecting duplicate requests
Impact of Partitioning State

● Why consistent hashing or DHT?
  ◆ Enable load balancing across partitions
  ◆ Accommodate state too big for one server

● What if operation touches multiple partitions?

● Examples:
  ◆ Add meeting to calendars of two participants
  ◆ Transfer money from one account to another
  ◆ What about looking up balance of two accounts?

● Need distributed transactions
Executing Transactions

- For atomicity, execute one transaction at a time
- Problem?

- Desired property: **Serializability**
  - Despite concurrent execution, externally visible effects equivalent to some serial order of execution

- **Strict serializability** = Serializability + linearizability
Example of Serializability

- Concurrent execution of transactions:
  - T1: Transfer $10 from Alice to Bob
  - T2: Read balance in Alice’s and Bob’s accounts
  - Initial balance of $100 in both accounts

- Permissible outputs for T2?
  - (Alice: $100, Bob: $100) or (Alice: $90, Bob: $110)

- Invalid outputs for T2:
  - (Alice: $90, Bob: $100) or (Alice: $100: Bob: $110)