A Note on Assignment 2

• You will need to handle concurrent snapshots.
  • Concurrent snapshots should not interfere with each other.
  • Capture state (state = number of tokens) at the time of marker reception.

• Start snapshot at server by calling StartSnapshot on the server object.

• Always update your local state (state = number of tokens).
  • Snapshotting process should not affect operation of system.
Some context...

Dynamo and Bayou both offer **high availability** and **weak consistency**

Most traditional databases offer **strong consistency** and **low availability**
   Not suitable for modern applications with super high demands

What are some example applications of each?
   - Flight ticket booking (HA)
   - Amazon shopping carts (HA)
   - Offline edits (HA)
   - Billing services (SC)
   - Bank accounts (SC)

*Both are desirable properties, but why can’t we achieve both in a system?*
CAP Theorem

During network partitions, impossible to achieve both C and A
CAP Theorem
Availability is important

- Tens of millions of customers at peak times
- Tens of millions of shopping cart requests, 3 million checkouts per day
- Hundreds of thousands of concurrently active sessions
- Strict Service-Level Agreements (SLAs) translate to business value
Dynamo

Fully decentralized, highly available key-value store

Always writeable, resolve conflicts during reads

API for clients to specify requirements (99.9th percentile)

Departure from RDBMS: simpler functionality, fewer guarantees, runs on commodity hardware (low-end, broadly compatible, non-specialized machines)
Techniques for achieving availability

*Consistent hashing* for partitioning key space

*Vector clocks* for reconciling conflicts during reads

*Sloppy quorums* for handling temporary failures

*Anti-entropy using Merkle trees* for syncing key-value pairs

*Gossip-based protocol* for membership notifications
Techniques for achieving availability

*Consistent hashing* for partitioning key space

*Vector clocks* for reconciling conflicts during reads

*Sloppy quorums* for handling temporary failures

*Anti-entropy using Merkle trees* for syncing key-value pairs

*Gossip-based protocol* for membership notifications
Consistent Hashing

Assign each node a random position on the ring

Node owns the preceding key range

For fault tolerance, replicate each key at N successor nodes in the ring

Virtual nodes: each physical node gets assigned multiple nodes on the ring (e.g. B, D, F)
Consistent Hashing

Desirable properties?

Uniform distribution of load

Minimum object movements when nodes join or leave the ring

Number of virtual nodes can be adjusted for device heterogeneity
Techniques for achieving availability

**Consistent hashing** for partitioning key space

**Vector clocks** for reconciling conflicts during reads

**Sloppy quorums** for handling temporary failures

**Anti-entropy using Merkle trees** for syncing key-value pairs

**Gossip-based protocol** for membership notifications
Conflict resolution

Two machines write different values to the same key

*Vector clocks*: list of (node, count) pairs where count is incremented on write

If one vector clock subsumes another, discard older value

Else, return all conflicting values to client
Context contains vector clocks

Dynamo client API is simple:

get(key) (value, context)

put(key, value, context)

Common pattern: put after get
<table>
<thead>
<tr>
<th>Product</th>
<th>Seller</th>
<th>Stock Status</th>
<th>Prime</th>
<th>Gift Option</th>
<th>Delete</th>
<th>Save for later</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat-Opoly by LatefortheSky</td>
<td>Prime</td>
<td>In Stock</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fancy Feast Wet Cat Food, Grilled, Seafood</td>
<td>Purina Fancy Feast</td>
<td>In stock. Usually ships within 3 to 4 days.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feast Variety Pack, 3-Ounce Can, Pack of 24 by Furrhaven Pet</td>
<td>Furrhaven Pet</td>
<td>In Stock</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conflict resolution

Two machines write different values to the same key

Vector clocks: list of (node, count) pairs where count is incremented on write

If one vector clock subsumes another, discard older value

Else, return all conflicting values to client
Techniques for achieving availability

*Consistent hashing* for partitioning key space

*Vector clocks* for reconciling conflicts during reads

*Sloppy quorums* for handling temporary failures

*Anti-entropy using Merkle trees* for syncing key-value pairs

*Gossip-based protocol* for membership notifications
Sloppy Quorums

Write to N nodes, return success when W < N nodes respond

Read from N nodes, return value(s) from R < N nodes

Typically, W+R > N means at least one writer and one reader overlap, so values are consistent

*Sloppy* here means skip nodes that have failed, such that even if W+R > N, the readers and writers may not overlap = *not consistent!*
Sloppy Quorums

Example:

Typical values are $N = 3$, $W = R = 2$

Nodes C and D have failed, so key $k$ is written to E and F instead

Nodes C and D recover, and now client tries to read from C and D = stale value
Hinted Handoff

“Hint” refers to the node the data originally belongs to.

Example:

Nodes E and F remember they are writing on behalf of C and D.

As soon as C and D recovers, E and F transfer their values for $k$ to C and D.
Sloppy Quorums

Write to N nodes, return success when W < N nodes respond

Read from N nodes, return value(s) from R < N nodes

Typically, W+R > N means at least one writer and one reader overlap, so values are consistent

*Sloppy* here means skip nodes that have failed, such that even if W+R > N, the readers and writers may not overlap = *not consistent!*
Techniques for achieving availability

*Consistent hashing* for partitioning key space

*Vector clocks* for reconciling conflicts during reads

*Sloppy quorums* for handling temporary failures

*Anti-entropy using Merkle trees* for syncing key-value pairs

*Gossip-based protocol* for membership notifications
Anti-entropy using Merkle trees

Goal: minimize durability loss from above techniques

Nodes responsible for the same key spaces exchange Merkle trees

Find differences quickly while exchanging little information
Techniques for achieving availability

*Consistent hashing* for partitioning key space

*Vector clocks* for reconciling conflicts during reads

*Sloppy quorums* for handling temporary failures

*Anti-entropy using Merkle trees* for syncing key-value pairs

*Gossip-based protocol* for membership notifications
Membership notification

Gossip-based protocol to propagate membership changes

Each node learns the key spaces handled by all other nodes

**Result:** zero-hop distributed hash table (DHT)

*Clearly not infinitely scalable,* but storage requirement not a problem in practice
Bayou

What is it?
- Weakly consistent, replicated storage system

Goals:
- Maximize availability, *support offline collaboration*
- Minimize network communication
- Agree on all values (eventually)
Bayou Writes

Legend
Commit Timestamp:Write Timestamp:Write Server

A
Versions
P: 0
A: 0
B: 0

B
Versions
P: 0
A: 0
B: 0

Primary
Versions
P: 0
A: 0
B: 0

W(X, 4) Client 1
Bayou Writes

Legend
Commit Timestamp:Write Timestamp:Write Server

Client 1

Primary Versions
∞:1:P \text{ w}(x,4)
P: 1
A: 0
B: 0

A Versions
P: 0
A: 0
B: 0

B Versions
P: 0
A: 0
B: 0
Bayou Writes

Legend
Commit Timestamp: Write Timestamp: Write Server

Primary Versions

∞:1:P \(w(X, 4)\)
P: 1
A: 0
B: 0

Client 1
\(W(Y, 8)\)

Client 2
\(W(X, 3)\)
Bayou Writes

Legend
Commit Timestamp: Write Timestamp: Write Server

Client 1
W(Z, 8)

Client 2
W(Y, 4)

A

Versions

∞: 7: A w(X, 3)

P: 0
A: 7
B: 0

Primary Versions

∞: 1: P w(X, 4)
∞: 7: P w(Y, 8)

P: 7
A: 0
B: 0

B

Versions

P: 0
A: 0
B: 0
Bayou Writes

Legend
Commit Timestamp: Write Timestamp: Write Server

Primary

<table>
<thead>
<tr>
<th>P: 7</th>
<th>W(X,4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 0</td>
<td></td>
</tr>
<tr>
<td>B: 0</td>
<td></td>
</tr>
</tbody>
</table>

Versions

<table>
<thead>
<tr>
<th>∞:1:P</th>
<th>W(X,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 12</td>
<td></td>
</tr>
<tr>
<td>B: 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>∞:12:P</th>
<th>W(Y,8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P: 0</td>
<td></td>
</tr>
<tr>
<td>A: 0</td>
<td></td>
</tr>
<tr>
<td>B: 0</td>
<td></td>
</tr>
</tbody>
</table>

A

<table>
<thead>
<tr>
<th>∞:7:A</th>
<th>W(X,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P: 0</td>
<td></td>
</tr>
<tr>
<td>A: 12</td>
<td></td>
</tr>
<tr>
<td>B: 0</td>
<td></td>
</tr>
</tbody>
</table>

B

<table>
<thead>
<tr>
<th>∞:5:B</th>
<th>W(Z,8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P: 0</td>
<td></td>
</tr>
<tr>
<td>A: 0</td>
<td></td>
</tr>
<tr>
<td>B: 5</td>
<td></td>
</tr>
</tbody>
</table>
Bayou Anti-Entropy

Anti-Entropy Session
A & B
Bayou Anti-Entropy

P

\[ \begin{align*}
\infty: 1: P & \quad \text{W(X,4)} \\
\infty: 7: P & \quad \text{W(Y,8)} \\
\end{align*} \]

P: 7
A: 0
B: 0

A

\[ \begin{align*}
\infty: 5: B & \quad \text{W(Z,8)} \\
\infty: 7: A & \quad \text{W(X,3)} \\
\infty: 12: A & \quad \text{W(Y,4)} \\
\end{align*} \]

P: 0
A: 12
B: 5

B

\[ \begin{align*}
\infty: 5: B & \quad \text{W(Z,8)} \\
\infty: 7: A & \quad \text{W(X,3)} \\
\infty: 12: A & \quad \text{W(Y,4)} \\
\end{align*} \]

P: 0
A: 12
B: 5
Bayou Commit

Primary commits its entries

\[
\begin{align*}
&\text{P: 7} \\
&\text{A: 0} \\
&\text{B: 0} \\
&\text{Versions} \\
&1:1:P \quad W(X,4) \\
&2:7:P \quad W(Y,8) \\
&\infty:5:B \quad W(Z,8) \\
&\infty:7:A \quad W(X,3) \\
&\infty:12:A \quad W(Y,4) \\
&\text{P: 0} \\
&\text{A: 12} \\
&\text{B: 5} \\
&\text{Versions} \\
&\infty:5:B \quad W(Z,8) \\
&\infty:7:A \quad W(X,3) \\
&\infty:12:A \quad W(Y,4) \\
&\text{P: 0} \\
&\text{A: 12} \\
&\text{B: 5}
\end{align*}
\]
Bayou Write

Write after anti-entropy session
Write timestamp = max(clock, max(TS)+1)
Bayou Anti-Entropy

Anti-Entropy Session
P & B

A

Versions

P: 0
A: 12
B: 5

1:1: P  W(X,4)
2:7: P  W(Y,8)

∞:5: B  W(Z,8)
∞:7: A  W(X,3)
∞:12: A  W(Y,4)

B

Versions

P: 0
A: 12
B: 13

1:1: P  W(X,4)
2:7: P  W(Y,8)

P: 7
A: 0
B: 0

∞:5: B  W(Z,8)
∞:7: A  W(X,3)
∞:12: A  W(Y,4)
∞:13: B  D(Y)
Anti-Entropy Session
P & B
Primary respects causality
Bayou Commit

Primary commits: Its entries
Bayou

After a number of commits and anti-entropy sessions (without further writes)
Bayou and Dynamo similarities

- Anti-entropy to achieve eventual consistency
- Exchange vector clocks to determine order of operations
- Expose conflict resolution to application
- High availability!