Bayou / Chord

Oct 6, 2022

[Adapted from Andrew Or and Ion Stoica’s original slides]
Context on Bayou: Disconnected Nodes [Stoica]

Early days: nodes always on when not crashed

○ Network bandwidth always plentiful.
○ Never needed to work on a disconnected node

Now: nodes detach then reconnect elsewhere

○ Even when attached, bandwidth is variable
○ Reconnection elsewhere means often talking to different replica
○ Work done on detached nodes
Bayou

- "[R]eplicated, [eventually] consistent storage system designed for ... portable machines with less-than-ideal network connectivity."

- System developed at PARC in the mid-90’s

- First coherent attempt to fully address the problem of disconnected operation
Bayou

What is it?
Weakly consistent, replicated storage system

Goals:
Maximize availability, support offline collaboration
Minimize network communication
Agree on all values (eventually)
Bayou Update Protocol: Review from Class

- Client sends update to a server
- Updates uniquely identified by:
  - `<Commit Sequence Number (CSN), Local Timestamp, Node-id`
- Updates are either committed or tentative
  - CSNs increase monotonically
  - Tentative updates have commit-stamp = $\infty$
- Only Primary server can commit updates
  - Allocates CSN in monotonically increasing order
  - CSN is different from time-stamp
Bayou Writes

Legend

Commit Sequence Number (CSN): Local Timestamp: Node-id

Primary Versions

P: 0  A: 0  B: 0

W(X, 4)  Client 1

A Versions

P: 0  A: 0  B: 0

B Versions

P: 0  A: 0  B: 0
Bayou Writes

Legend

Commit Sequence Number (CSN): Local Timestamp: Node-id

Client 1
Bayou Writes

Legend

Commit Sequence Number (CSN): Local Timestamp: Node-id

Client 1
W(Y, 8)

Client 2
W(X, 3)

Primary Versions

∞:1:P  W(X, 4)
P: 1
A: 0
B: 0

Client 1

Version A

P: 0
A: 0
B: 0

Version B

P: 0
A: 0
B: 0
Bayou Writes

Legend
Commit Sequence Number (CSN): Local Timestamp: Node-id

Client 1
W(Z, 8)

Client 2
W(Y, 4)

A
$\infty:7:A \ w(X,3)$

P: 0
A: 7
B: 0

Primary Versions

$\infty:1:P \ w(X,4)$

P: 7
A: 0
B: 0

$\infty:7:P \ w(Y,8)$

P: 0
A: 0
B: 0

B

Versions
Bayou Writes

Legend

Commit Sequence Number (CSN): Local Timestamp: Node-id

A  Versions
∞:7:A  w(X,3)  P: 0
∞:12:A  w(Y,4)  A: 12
            B: 0

B  Versions
∞:5:B  w(Z,8)  P: 0
            A: 0
            B: 5
Bayou Anti-Entropy (Sync)

Anti-entropy Session
A & B

A

Versions

\begin{align*}
\infty &: 7: A \quad W(X,3) \\
\infty &: 12: A \\
& \quad \text{W}(Y,4) \\
P &: 0 \\
A &: 12 \\
B &: 0
\end{align*}

B

Versions

\begin{align*}
\infty &: 5: B \\
& \quad \text{W}(Z,8) \\
P &: 0 \\
A &: 0 \\
B &: 5
\end{align*}

P

Versions

\begin{align*}
\infty &: 1: P \\
& \quad \text{W}(X, 4) \\
\infty &: 7: P \\
& \quad \text{W}(Y, 8) \\
P &: 7 \\
A &: 0 \\
B &: 0
\end{align*}
Bayou Anti-Entropy (Sync)
Bayou Commit

Primary commits its entries
Bayou Write

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

Anti-entropy Session
P & B

A

Versions

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

P: 0
A: 12
B: 5

P

Versions

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

P: 7
A: 0
B: 0

B

Versions

∞:5:B W(Z,8)
∞:7:A W(X,3)
∞:12:A W(Y,4)
∞:13:B D(Y)

P: 0
A: 12
B: 13

∞:5:B W(Z,8)
∞:7:A W(X,3)
∞:12:A W(Y,4)
∞:13:B D(Y)
Bayou Anti-Entropy

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), all nodes converge on same state.
Context for Chord: Key Value Stores

Amazon:
- Key: customerID
- Value: customer profile (e.g., buying history, credit card, ..)

Facebook, Twitter:
- Key: UserID
- Value: user profile (e.g., posting history, photos, friends, ..)

iCloud/iTunes:
- Key: Movie/song name
- Value: Movie, Song

Distributed file systems
- Key: Block ID
- Value: Block

Credit: Ion Stoica’s slide deck
Context for Chord: Key Value Stores

Key Value Store

Also called a Distributed Hash Table (DHT)
Main idea: partition set of key-values across many machines

Credit: Ion Stoica’s slide deck
Chord

- Chord: “a distributed lookup protocol” for a peer-to-peer distributed hash table [Stoica ’01]
- *Consistent hashing* for partitioning key space / lookup
Consistent Hashing

Assign each node a random position on the ring

Node owns the preceding key range

Virtual Nodes

Key K

Nodes B, C and D store keys in range (A,B) including K.
Achieving Efficiency: *finger tables*

Finger Table at 80

<table>
<thead>
<tr>
<th>i</th>
<th>ft[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>1</td>
<td>96</td>
</tr>
<tr>
<td>2</td>
<td>96</td>
</tr>
<tr>
<td>3</td>
<td>96</td>
</tr>
<tr>
<td>4</td>
<td>96</td>
</tr>
<tr>
<td>5</td>
<td>112</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
</tr>
</tbody>
</table>

Say $m=7$

\[
(80 + 2^6) \mod 2^7 = 16
\]

\[
i\text{th entry at peer with id } n \text{ is first peer with id } \geq n + 2^i \mod 2^m
\]

Credit: Ion Stoica’s slide deck