Bayou: Replication with Weak Inter-Node Connectivity

Brad Karp
UCL Computer Science

CS GZ03 / M030
26th October 2016
Context: Availability vs. Consistency

• NFS, Ivy, 2PC all had single points of failure; not available under failures
• Paxos allows view-change to elect primary, thus state machine replication
  – Strong consistency model: all operations in same order at all replicas, always appearance of single system-wide order for all operations
  – Strong reachability requirement: majority of nodes must be reachable by leader
• If reachability weaker, can we provide any consistency when we replicate?
Bayou: Calendar Application Case Study

• Today’s lecture:
  – Bayou’s office calendar application as case study in ordering and conflicts in a distributed system with poor connectivity

• Each calendar entry: room, time, and set of participants

• Want everyone to see same set of entries (eventually)
  – else, users may double-book room, avoid using unoccupied room, &c.
Traditional Calendar Application: One Central Server

- Ordering of users’ requests: only one copy, server picks order
- Conflict resolution: server checks for conflicts (i.e., “is this room already booked during this period?”) before accepting updates
  - Returns error to user if conflict; user decides what to do
What’s Wrong with Central Server?

• Want my calendar on my iPhone
  – i.e., each user wants database replicated on his PDA or laptop
  – No master copy

• iPhone has only intermittent connectivity
  – 3G/4G expensive when roaming, WiFi not everywhere; no connectivity on many flights
  – Bluetooth useful for direct contact with other calendar users’ PDAs, but very short range
Simple Proposal: Swap Complete DBs

- Suppose two users in Bluetooth range
- Each sends entire calendar DB to other, as with “classic” Palm or iPhone sync
- Possibly lots of network bandwidth
- What if conflict, i.e., two concurrent meetings?
  - iPhone sync just keeps both meetings!
  - Want to do better: automatic conflict resolution
Automatic Conflict Resolution

• Can’t just view DB items as bits—too little information to resolve conflicts!
  – “Both files have changed” can falsely conclude entire DBs conflict
  – “Distinct record in each DB changed” can falsely conclude no conflict

• Want to build intelligent DB app that knows how to resolve conflicts
  – More like users’ updates: read DB, think, change request to eliminate conflict
  – Must ensure all nodes resolve conflicts in same way to keep replicas consistent
Insight: Ordering of Updates

• Maintain ordered list of updates at each node
• Make sure every node holds same updates
• Make sure every node applies updates in same order
• Make sure updates are deterministic function of DB contents
• If we obey above, "sync" really just a simple merge of two ordered lists!
What’s in a Write?

• Each node’s ordered list of writes: write log

• Suppose calendar update takes form:
  – “10 AM meeting, Room 6.12, Mark and Brad”
  – Sufficient for our goal?

• Better: “1-hour meeting, Room 6.12, Mark and Brad, at 9, else 10, else 11”
  – Also include unique ID: <local-time-stamp, originating-node-ID>
What’s in a Write?

Instructions for write more than data to write
Write log really an “instruction” for calendar program
Want all nodes to execute same instructions in same order, eventually

• Better: “1-hour meeting, Room 6.12, Mark and Brad, at 9, else 10, else 11”
  – Also include unique ID: <local-time-stamp, originating-node-ID>
Write Log Example

- \(<701, A>\): Node A asks for meeting M1 to occur at 10 AM, else 11 AM
- \(<770, B>\): Node B asks for meeting M2 to occur at 10 AM, else 11 AM
- Let’s agree to sort by write ID (e.g., \(<701, A>\))
- As “writes” spread from node to node, nodes may initially apply updates in different orders
Write Log Example (2)

- Each newly seen write merged into log
- Log replayed
  - May cause calendar displayed to user to change!
  - i.e., all entries really “tentative,” nothing stable
- After everyone has seen all writes, everyone will agree (contain same state)
Global Time Synchronization Impossible

• Does this mean that globally ordering writes by local timestamps impossible?
• No—timestamps just allow agreement on order
  – Nodes may have wrong clocks
  – OK, so long as users don’t expect writes to reach calendar in real-time order made
Timestamps for Write Ordering: Limitations

- Ordering by write ID arbitrarily constrains order
  - Never know if some write from past hasn’t yet reached your node
  - So all entries in log must be **tentative forever**
  - And you must store entire log forever

- Problem: how can we allow committing a tentative entry?
  - So we can have meetings and trim logs
Criteria for Committing Writes

• For log entry X to be committed, everyone must agree on:
  – Total order of all previous committed entries
  – Fact that X is next in total order
  – Fact that all uncommitted entries are “after” X
How Bayou Agrees on Total Order of Committed Writes

- One node designated “primary replica”
- Primary marks each write it receives with permanent CSN (commit sequence number)
  - That write is committed
  - Complete timestamp is <CSN, local-TS, node-id>
- Nodes exchange CSNs
- CSNs define total order for committed writes
  - All nodes eventually agree on total order
  - Uncommitted writes come after all committed writes
Showing Users that Writes Have Committed

• Still not safe to show users that an appointment request has committed

• Entire log up to newly committed entry must be committed
  – else there might be earlier committed write a node doesn’t know about!
  – ...and upon learning about it, would have to re-run conflict resolution

• Result: committed write not stable unless node has seen all prior committed writes
Showing Users that Writes Have Committed

Bayou propagates writes between nodes to enforce this invariant
i.e., Bayou propagates writes in order
must be committed
– else there might be earlier committed write a node doesn’t know about!
– ...and upon learning about it, would have to re-run conflict resolution

• Result: committed write not stable unless node has seen all prior committed writes
Committed vs. Tentative Writes

• Can now show user if a write has committed
  – When node has seen every CSN up to that point, as guaranteed by propagation protocol

• Slow or disconnected node cannot prevent commits!
  – Primary replica allocates CSNs; global order of writes may not reflect real-time write times

• What about tentative writes, though—how do they behave, as seen by users?
Tentative Writes

- Two nodes may disagree on meaning of tentative (uncommitted) writes
  - Even if those two nodes have synced with each other!
  - Only CSNs from primary replica can resolve these disagreements permanently
Example: Disagreement on Tentative Writes
Example: Disagreement on Tentative Writes

\[ W <0, C> \]
Example: Disagreement on Tentative Writes

A   B   C

W <0, C>

logs

<0, C>
Example: Disagreement on Tentative Writes

\[ \text{time} \quad \begin{array}{ccc} \text{A} & \text{B} & \text{C} \\ W <0, C> & W <1, B> & W <0, C> \end{array} \]
Example: Disagreement on Tentative Writes

A  B  C

W <0, C>
W <1, B>
W <1, B>

logs
Example: Disagreement on Tentative Writes

- Time
  - A
  - B
  - C

Logs:
- <2, A>
- <1, B>
- W <0, C>
Example: Disagreement on Tentative Writes

- Time logs:
  - \(W <2, A>\) for A
  - \(W <1, B>\) for B
  - \(W <0, C>\) for C

- Logs:
  - A: <2, A>
  - B: <1, B>
  - C: <0, C>
Example: Disagreement on Tentative Writes

A

B

C

W <0, C>

W <1, B>

W <2, A>

sync (3)

<2, A>

<1, B>

<0, C>

logs

time
Example: Disagreement on Tentative Writes

\[ \text{time} \]

A \quad B \quad C

W <2, A> \\
W <1, B> \\
W <0, C>

\[ \text{sync (3)} \]

\[ \text{logs} \]

\begin{align*}
\text{A:} & \quad \text{<1, B>} \\
& \quad \text{<2, A>}
\end{align*}

\begin{align*}
\text{B:} & \quad \text{<1, B>} \\
& \quad \text{<2, A>}
\end{align*}

\begin{align*}
\text{C:} & \quad \text{<0, C>}
\end{align*}
Example: Disagreement on Tentative Writes

logs

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;1, B&gt;</td>
<td>&lt;1, B&gt;</td>
<td>&lt;0, C&gt;</td>
</tr>
<tr>
<td>&lt;2, A&gt;</td>
<td>&lt;2, A&gt;</td>
<td></td>
</tr>
</tbody>
</table>

time

W <2, A>

sync (3)

sync (4)
Example: Disagreement on Tentative Writes

```
<table>
<thead>
<tr>
<th>time</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>W &lt;0, C&gt;</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>W &lt;1, B&gt;</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>W &lt;2, A&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Logs:

```
A: 
B: <0, C> <1, B> <2, A> 
C: <0, C> <1, B> <2, A> 
```

Syncs:

- Sync (3): <0, C> <1, B> <2, A>
- Sync (4): <0, C> <1, B> <2, A>
Example: Disagreement on Tentative Writes

\[
\begin{array}{c|c|c|c}
\text{time} & \text{A} & \text{B} & \text{C} \\
0 & \text{W <0, C>} & \text{W <1, B>} & \text{W <0, C>} \\
1 & \text{W <1, B>} & \text{W <2, A>} & \\
2 & \text{W <2, A>} & & \\
\end{array}
\]

\[\text{sync (3)}\] \quad \text{sync (4)}

logs

\[
\begin{array}{c|c|c}
\text{logs} & \text{A} & \text{B} & \text{C} \\
1 & \text{<1, B>} & \text{<0, C>} & \text{<0, C>} \\
2 & \text{<2, A>} & \text{<1, B>} & \text{<1, B>} \\
\end{array}
\]
Trimming the Log

• When nodes receive new CSNs, can discard all committed log entries seen up to that point
  – Update protocol guarantees CSNs received in order

• Instead, keep copy of whole database as of highest CSN
  – By definition, official committed database
  – Everyone does (or will) agree on contents
  – Entries never need go through conflict resolution
Trimming the Log

• When nodes receive new CSNs, can discard all committed log entries seen up to that point

Result: no need to keep years of log data!

• Instead, keep copy of whole database as of highest CSN
  – By definition, official committed database
  – Everyone does (or will) agree on contents
  – Entries never need go through conflict resolution
Ordering of Commits by Primary Replica

• Can primary commit writes in any order it pleases?
  – Suppose user creates appointment, then decides to delete it, or change attendee list
  – What order must these ops take in CSN order?
    • Create first, then delete or modify
    • Must be true in every node’s view of tentative log entries, too!

• Total order of writes must preserve order of writes made at each node
  – Not necessarily order among different nodes’ writes
How Does Primary Replica Commit Each Node’s Writes in Order?

• Nodes don’t quite use real-time clocks for timestamps—use Lamport logical clocks
  – Anytime see message with later timestamp than current time, set clock to after that timestamp

• All nodes send updates in order

• So primary receives updates in per-node causal order, and commits them in that order
Syncing with Trimmed Logs

• Suppose nodes discard all writes in log with CSNs
  – Just keep copy of “stable” DB, reflecting discarded entries

• Cannot receive writes that conflict with DB
  – Only could be if write has CSN less than a discarded CSN
  – Already saw all writes with lower CSNs in right order—if see them again, can discard!
Syncing with Trimmed Logs (2)

- To propagate to node X
- If node X’s highest CSN less than mine:
  - Send X full stable DB
  - X uses that DB as starting point
  - X can discard all his CSN log entries
  - X can play his tentative writes into that DB
- If node X’s highest CSN greater than mine:
  - X can ignore my DB!
Bayou: Summary

• **Seems more useful than old Palm’s calendar!**
  – Often disconnected when making appointments
  – Automatic conflict resolution convenient

• **Not at all transparent to applications!**
  – Very strange programming practices
  – Writes are code, not just bits!
  – Check for conflicts, resolve conflicts

• **Doesn’t work for all apps**
  – Bank account may be OK
  – But hard to imagine for source code repository!