Hash History: A Method for Reconciling Mutual Inconsistency in Optimistic Replication

Brent ByungHoon Kang, Robert Wilensky and John Kubiatowicz

CS Division, UC Berkeley
Background

- Optimistic Replication
  - Allow mutable replica to be inconsistent temporarily
    - in a controlled way
    - for high availability and performance
  - Tentative update support in OceanStore
  - Bayou, USENET, and Peer-to-Peer File System (e.g., Ivy, Pangaea, etc.)

- Need Mechanism for
  - Figuring out the ordering among updates
  - Extracting deltas to be exchanged during reconciliation
Previous Approaches: Version Vectors

- Widely used in reconciling replicas
  - In most weakly consistent replication systems
  - Bayou, Ficus, Coda, Ivy, Pangaea ... etc.
- Complexity of management grows
  - As new replica site added or deleted
  - Need to assign unique id dynamically for newly added replica sites
- Doesn’t scale as number of replica site increases
  - Version vector needs one entry for each replica site
  - Size of vector grows in proportion to number of replica sites
Our Proposal: Hash History

- Each site keeps a record of the hash of each version
  - Capture dependency among versions as a directed graph of version hashes (i.e., hash history)
- The sites exchange the hash history in reconciling replicas
- The most recent common ancestral version can be found, if no version dominates
  - Useful hints in a subsequent diffing/merging
\[ H_{i,\text{site}} = \text{hash} (V_{i,\text{site}}) \]
\[ \text{\(H_{i,\text{site}} = \text{hash} (V_{i,\text{site}})\)} \]
\[ H_{i,\text{site}} = \text{hash}(V_{i,\text{site}}) \]
\[ H_{i,\text{site}} = \text{hash}(V_{i,\text{site}}) \]
Hash History with Hashtable

Hash History Graph

Hash History with Hashtable

<table>
<thead>
<tr>
<th>Child</th>
<th>Parents</th>
</tr>
</thead>
<tbody>
<tr>
<td>H_{0,A}</td>
<td>null</td>
</tr>
<tr>
<td>H_{1,A}</td>
<td>H_{0,A}</td>
</tr>
<tr>
<td>H_{2,B}</td>
<td>H_{0,A}</td>
</tr>
<tr>
<td>H_{3,C}</td>
<td>H_{0,A}</td>
</tr>
<tr>
<td>H_{4,A}</td>
<td>H_{1,A} : H_{2,B}</td>
</tr>
<tr>
<td>H_{5,C}</td>
<td>H_{4,A} : H_{3,C}</td>
</tr>
</tbody>
</table>

Latest: H_{5,C}
Hash History with Hashtable

<table>
<thead>
<tr>
<th>Child</th>
<th>Parents</th>
<th>delta</th>
</tr>
</thead>
<tbody>
<tr>
<td>H_{0,A}</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>H_{1,A}</td>
<td>H_{0,A}</td>
<td>d_{1}</td>
</tr>
<tr>
<td>H_{2,B}</td>
<td>H_{0,A}</td>
<td>d_{2}</td>
</tr>
<tr>
<td>H_{3,C}</td>
<td>H_{0,A}</td>
<td>d_{3}</td>
</tr>
<tr>
<td>H_{4,A}</td>
<td>H_{1,A} : H_{2,B}</td>
<td>m_{4}</td>
</tr>
<tr>
<td>H_{5,C}</td>
<td>H_{4,A} : H_{3,C}</td>
<td>m_{5}</td>
</tr>
</tbody>
</table>

Latest : H_{5,C}

(a) Hash History Graph

(b) Hash History with Hashtable
HH Properties

- Size of hash history is unbounded
  - Simple Aging
  - Sharable Archived Hash Histories
- Can capture equality case
  - When two different schedule of deltas produce the same output
  - Helps faster convergence
Why Less Conflict in HH than VV

- HH can convey equality information to the descendents while VV cannot
  - E.g., \( v_1 = \langle A:4,B:5,C:0,D:0,E:0,F:0 \rangle \)
  - \( v_2 = \langle A:5,B:4,C:0,D:0,E:0,F:0 \rangle \)
  - C merges then \( v_3 = \langle A:5,B:5,C:1,D:0,E:0,F:0 \rangle \)
  - E merges then \( v_4 = \langle A:5,B:5,C:0,D:0,E:1,F:0 \rangle \)
  - \( v_3 \) and \( v_4 \) could be the same but VV shows conflict!

- If \( v_3 \) and \( v_4 \) are considered equal, then
  - all descendents of \( v_4 \) will dominate \( v_3 \).

- If \( v_3 \) and \( v_4 \) are considered as in conflict,
  - all descendents of \( v_4 \), will be in conflict with \( v_3 \).
Experiment Goal

- Comparison with version vector result:
  - HH converges faster with a lower conflict rate than a version vector scheme
  - To what extent is this true in practice?

- Aging Policy:
  - the aging period for pruning hash history
  - vs. HH size
  - vs. the false conflict rate due to aging
    - when the pruned part of the hash history is required for determining the version dominance
Simulation Setup

- Event-driven simulator
  - Events are collected from CVS logs
  - Each user represents a replica site
  - Reads the event \(<\text{time, user, filename}>\)
  - After each event, the simulator
    - repeats the anti-entropy for 50% (or 25%) of the total number of sites.
    - E.g., if there are 20 sites so far, the anti-entropy is repeated for 10 times with 50% parameter after each event.
# CVS Trace Data
(from sourceforge.net)

<table>
<thead>
<tr>
<th></th>
<th>Dri</th>
<th>Freenet</th>
<th>Plogen</th>
</tr>
</thead>
<tbody>
<tr>
<td># of events</td>
<td>10137</td>
<td>2281</td>
<td>404</td>
</tr>
<tr>
<td># of users</td>
<td>21</td>
<td>64</td>
<td>39</td>
</tr>
<tr>
<td>inter-commit time AVG</td>
<td>101.3 min</td>
<td>237.8 min</td>
<td>225.4 min</td>
</tr>
<tr>
<td>MEDIAN</td>
<td>0.016 min</td>
<td>34.6 min</td>
<td>2.16 min</td>
</tr>
</tbody>
</table>
Dominance rate of VV and HH

![Graph showing the dominance rate of VV and HH over the number of anti-entropy cycles. The graph plots the ratio of results to cycle number against the number of anti-entropy cycles, with two lines representing VV and HH dominance rates.](image)
### Aging Period vs. HH Size

<table>
<thead>
<tr>
<th>Aging period (days)</th>
<th>HH size (# of entries) – dri</th>
<th>pcgen</th>
<th>freenet</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>146.3</td>
<td>159.1</td>
<td>61.5</td>
<td>122.3</td>
</tr>
<tr>
<td>64</td>
<td>413.9</td>
<td>443.9</td>
<td>147.5</td>
<td>335.1</td>
</tr>
<tr>
<td>128</td>
<td>551.5</td>
<td>591.7</td>
<td>612.8</td>
<td>585.3</td>
</tr>
</tbody>
</table>
Conclusion

- Simple to maintain
  - No complexity in site addition/deletion
  - No need to assign unique id dynamically for newly added replica sites
- Scalable to thousands of sites
  - HH grows in proportion to number of update instances not number of sites
- Faster Convergence
  - HH can capture and propagate equality information
- HH growth can be controlled effectively by
  - using aging policy or sharing archived hash history
Future Work

- Security aspect of HH
  - Self-verifiable
  - Can detect mal-functioning site
- More information
  - Hash History Approach for Reconciling Mutual Inconsistency in Optimistic Replication, B. Kang, R. Wilensky and J. Kubiatowicz, The 23rd International Conference on Distributed Computing Systems (ICDCS), 2003, Providence, Rhode Island USA
  - http://www.cs.berkeley.edu/~hoon/hashhistory
\[\sigma_{\text{site value}} = \text{sign}_{\text{site}}(\text{value of site's local counter})\]
\[ H_i = \text{hash} (V_i, \text{site}) \]
\[ \alpha_{\text{site}}^k = \text{hash}(\alpha^k \text{’s parents} \parallel H_k) \]
$H_i = \text{hash}(V_i, \text{site})$