Flexible Update Propagation for Weakly Consistent Replication

Karin Petersen, Mike K. Spreitzer, Douglas B. Terry, Marvin M. Theimer and Alan J. Demers
Anti-entropy

- Entropy = disorder
- Anti-entropy = bringing two replicas up-to-date
- Allow arbitrary pairwise communication
- Question: what updates to propagate in what order?
Design goals

- Arbitrary communication topologies
- Operation over low-bandwidth networks
- Incremental progress
- Eventual consistency
- Efficient storage management
- Propagation through transportable media
- Lightweight replica creation and retirement
- Arbitrary policy choices
Basic setup

- Each replica/server maintains
  - Database
  - Write log
- Clients read or write from replicas
- Anti-entropy
  - one-way operation between two replicas
  - through propagation of writes
  - write propagation obeys accept-order
Accept order

- Each write carries an accept stamp = (Lamport clock, replica-id)
- Accept stamps define a partial order over all writes by a single server
- Prefix Property: If R has write $W_i$ that was initially accepted by server X, it has all writes X accepted before $W_i$
Version vectors

- Prefix property enables compact representation of a replica’s position
- Each replica $R$ maintains version vector $R.V$ such that $R.V(X)$ is largest accept-stamp of any write accepted by $X$ and known to $R$
- Replicas use VVs to bring each other up-to-date
Anti-entropy protocol

anti-entropy(S,R) {
    Get R.V from receiving server R
    \# now send all the writes unknown to R
    w = first write in S.write-log
    WHILE (w) DO
        IF R.V(w.server-id) < w.accept-stamp THEN
            \# w is new for R
            SendWrite(R, w)
            w = next write in S.write-log
        END
    END
}

Figure 1. Basic anti-entropy executed at server S to update receiving server R
Write stability

• When to apply a write to database and discard from log? What if long-lost replica shows up?

• Need a primary to commit writes
  • assigns commit sequence number (CSN) to writes

• New partial order enforced by (CSN, accept-stamp) in that order
Propagating committed writes

anti-entropy(S,R) {
    Get R.V and R.CSN from receiving server R
    #first send all the committed writes that R does not know about
    IF R.CSN < S.CSN THEN
        w = first committed write that R does not know about
        WHILE (w) DO
            IF w.accept-stamp <= R.V(w.server-id) THEN
                #R has the write, but does not know it is committed
                SendCommitNotification(R, w.accept-stamp, w.server-id, w.CSN)
            ELSE
                SendWrite(R, w)
            END
        w = next committed write in S.write-log
    END
    w = first tentative write
    #now send all the tentative writes
    WHILE (w) DO
        IF R.V(w.server-id) < w.accept-stamp THEN
            SendWrite(R, w)
            w = next write in S.write-log
        END
    }
}

Figure 2. Anti-entropy with support for committed writes (run at server S to update R)
Write log truncation

- Replica S maintains a version vector S.O representing omitted prefix of write log
- S maintains CSN for S.O
- If S.OSN > R.CSN, then S has discarded committed writes R is missing
- What to do?
Full database transfer

\[
\text{anti-entropy}(S,R) \{
\text{Request } R.V \text{ and } R.CSN \text{ from receiving server } R
\text{#check if } R\text{'s write-log does not include all the necessary writes to only send writes or}
\text{# commit notifications}
\text{IF } (S.OSN > R.CSN) \text{ THEN}
\text{  # Execute a full database transfer}
\text{  Roll back } S\text{'s database to the state corresponding to } S.O
\text{  SendDatabase}(R, S.DB)
\text{  SendVector}(R, S.O) \# \text{this will be } R\text{'s new } R.O \text{ vector}
\text{  SendCSN}(R, S.OSN) \# \text{R's new } R.OSN \text{ will now be } S.OSN
\text{END}
\text{# now same algorithm as in Figure 2, send anything that } R \text{ does not yet know about}
\text{IF } R.CSN < S.CSN \text{ THEN}
\text{  } w = \text{first committed write that } R \text{ does not yet know about}
\text{  WHILE } (w) \text{ DO}
\text{    IF } w.\text{accept-stamp} \leq R.V(w.\text{server-id}) \text{ THEN}
\text{      SendCommitNotification}(R, w.\text{accept-stamp}, w.\text{server-id}, w.CSN)
\text{    ELSE}
\text{      SendWrite}(R, w)
\text{    END}
\text{  } w = \text{next committed write in } S\text{.write-log}
\text{END}
\text{  } w = \text{first tentative write in } S\text{.write-log}
\text{  WHILE } (w) \text{ DO}
\text{    IF } R.V(w.\text{server-id}) < w.\text{accept-stamp} \text{ THEN}
\text{      SendWrite}(R, w)
\text{    } w = \text{next write in } S\text{.write-log}
\text{  END}
\text{END}
\}
\]

\text{Figure 3. Anti-entropy with support for write-log truncation (run at server } S \text{ to update server } R)
Consistency

- Causally consistent prefix at any time
- Total order enforced by primary
  - eventual consistency
- Session guarantees, e.g., *read your writes, monotonic reads/writes, writes follow reads* depend on causal prefix property
Replica management

• Need mechanism to
  • assign unique id to a replica
  • determine replica creation/retirement
• Use writes to create/retire!
  • maintains causal prefix property
Replica management

- $S_i$ creates itself by sending creation write to any $S_k$ as <$inf, T_{k,i}, S_k>$, where $T_{k,i}$ is accept stamp assigned by $S_k$
- <$T_{k,i}, S_k>$ becomes $S_i$’s id, and $T_{k,i} + 1$ its initial accept stamp
- Creation/retirement propagated just like regular writes
## Features enabled

<table>
<thead>
<tr>
<th>Feature</th>
<th>One-way Peer-to-Peer</th>
<th>Operation-based</th>
<th>Partial Propagation Order</th>
<th>Causal Propagation Order</th>
<th>Stable Log Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbitrary Communication Topologies</td>
<td>✦</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arbitrary Policy Choices</td>
<td>✦</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low-bandwidth Networks</td>
<td></td>
<td>✦</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Incremental Progress</td>
<td>✦</td>
<td>✦</td>
<td>✦</td>
<td>✦</td>
<td>✦***</td>
</tr>
<tr>
<td>Eventual Consistency</td>
<td></td>
<td></td>
<td></td>
<td>✦</td>
<td>✦**</td>
</tr>
<tr>
<td>Aggressive Storage Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✦</td>
</tr>
<tr>
<td>Use of Transportable Media</td>
<td>✦</td>
<td></td>
<td></td>
<td>✦</td>
<td></td>
</tr>
<tr>
<td>Light-weight Dynamic Replica Sets</td>
<td>✦</td>
<td>✦</td>
<td></td>
<td>✦</td>
<td></td>
</tr>
<tr>
<td>Per Update Conflict Management</td>
<td></td>
<td></td>
<td></td>
<td>✦</td>
<td></td>
</tr>
<tr>
<td>Session Guarantees</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✦</td>
</tr>
</tbody>
</table>
Performance

Figure 5. Anti-entropy execution as a function of the number of writes propagated (each write corresponds to one mail message)
Figure 6. Anti-entropy execution time breakdown for the propagation of 100 writes
(standard deviations on all total times are within 2.2% of the reported numbers)
Performance

Figure 8. Anti-entropy execution time for 100 writes as a function of the number of replicas