Consistency without concurrency control in large, dynamic systems

Marc Shapiro, INRIA & LIP6
Nuno Preguiça, Universidade Nova de Lisboa
Mihai Leția, ENS Lyon

Two things
- no concurrency control
- large dynamic
Consistency without concurrency control

Object $x$, operation $f(x)$
- propose $f(x_1)$
- eventually replay $f(x_2), f(x_3), ...$

If $f || g$ commute: converges safely without concurrency control

Commutative Replicated Data Type (CRDT): Designed for commutative operations

Not same order at 1 and 2?
OK if
- concurrent $f$ and $g$ commute
Assuming causal delivery
A sequence CRDT

Treedoc = sequence of elements:
  • *insert-at-pos*, *delete*
  • Commutative when concurrent
  • Minimise overhead
  • Scalable

A commutative replicated data type for cooperative editing, ICDCS 2009

Focus today:
  • Garbage collection
  • *vs. scale*

I will just skim the surface of the Treedoc design
Refer to ICDCS paper for the details
Commutative updates

Naming tree: minimal, self-adjusting: logarithmic
TID: path = [0|1]*
Contents: infix order
*insert* adds leaf ⇒ non-destructive, TIDs don’t change
Delete: *tombstone*, TIDs don't change

Consistency without concurrency control in large, dynamic systems

Thanks to non-destructive updates, immutable TIDs: concurrent updates commute
Efficient: Data structures and TID lengths logarithmic *
if balanced*
Ignoring lots of details, e.g. concurrent inserts at same position (see paper)
Wikipedia GWB page: space overhead

Consistency without concurrency control in large, dynamic systems

GWB: most edited page
Edits translated into treedoc insert/deletes
- Tree unbalanced, long TIDs, lots of tombstones: not logarithmic
In this example rebalancing is not spectacular.
Imagine a deep unbalanced tree with lots of tombstones: large effect.

Why rebalance:
- Unbalanced tree costs time, space
- Long TIDs
- Tombstone overhead
Rebalance

Invalidates TIDs:
- Frame of reference = epoch
- Requires agreement
- Pervasive!
  - e.g. Vector Clocks

Consistency without concurrency control in large, dynamic systems

TID changed: R was $\epsilon$, now 10
Pervasive problem:
- asynchronous updates $\Rightarrow$ old data structures
- see cleaning up Vector Clocks

(Background colour indicates epoch)
Rebalance in large, dynamic systems

Rebalance requires consensus
Consensus requires small, stable membership
  • Large communities?!
  • Dynamic scenarios?!

Solution: two tiers
  • *Core*: rebalancing (and updates)
  • *Nebula*: updates (and rebalancing)
  • Migration protocol

Core: controls rebalancing
Group membership
Small, stable

**Rebalance:**
- Unanimous agreement (2-phase commit)
- All core sites in same epoch

Arbitrary membership
Large, dynamic
Communicate with sites in same epoch only
*Catch-up* to rebalance, join core epoch

Migrate core to nebula: just leave group
Core

Group membership
Small, stable

Rebalance:
• Unanimous agreement (2-phase commit)
• All core sites in same epoch

Nebula

Arbitrary membership
Large, dynamic

Communicate with sites in same epoch only

Catch-up to rebalance, join core epoch

Migrate from nebula to core: migrate to core epoch + join group
Catch-up protocol summary

- **Core**
  - Send old updates
  - Send rebalance

- **Nebula**
  - replay core's updates
  - Replay rebalance, ignoring nebula updates.
  - Transform nebula updates.
  - Send transformed updates.

Here is the basic insight to the migration protocol:
- Replay core's updates: N now in same state as C before rebalance
- Replay rebalance: *ignoring concurrent N updates*, has same effect as in C ==> same TIDs, same epoch
- Transform buffer: now ready to be replayed in C (in different order, but that's OK since they commute)
- N now in C state, can join the core or remain in nebula

Furthermore updates are idempotent (multiple catch-ups cause no harm)
Consistency without concurrency control in large, dynamic systems

*Catch-up protocol*

---

**Core Site**

- **del(1)**

  ![Core Site Diagram]

**Nebula Site**

- **ins(L, 00)**
- **ins('001)**

  ![Nebula Site Diagram]

---

**Table**

<table>
<thead>
<tr>
<th>del(1)</th>
<th>ins(L, 00)</th>
<th>ins('001)</th>
</tr>
</thead>
</table>

---

- del in old epoch
- ins + ins in
- old epoch
- rebalance starts new epoch
Consistency without concurrency control in large, dynamic systems

**Catch-up protocol**

- **Core Site**
  - `R`
  - `A`
  - `I`
  - `N`

- **Nebula Site**
  - `R`
  - `A`
  - `I`
  - `L`

### Operations

<table>
<thead>
<tr>
<th>Core Site</th>
<th>Nebula Site</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>del(1)</code></td>
<td><code>ins(L,00)</code></td>
</tr>
<tr>
<td><code>rebalance</code></td>
<td><code>ins('001)</code></td>
</tr>
<tr>
<td><code>ins(L,00)</code></td>
<td><code>ins('001)</code></td>
</tr>
</tbody>
</table>

- `del in old epoch`
- `old epoch`
- `rebalance starts new epoch`
Catch-up protocol

Core Site

N
I
R
I

Nebula Site

R
I
A
L
N
I

\[
\begin{array}{c|c|c}
\text{del}(I) & \text{ins}(L,00) \\
\text{rebalance} & \text{ins}(',001) \\
\end{array}
\]

Consistency without concurrency control in large, dynamic systems

white = old epoch
pink = new epoch
Consistency without concurrency control in large, dynamic systems

Core Site

Nebula Site

<table>
<thead>
<tr>
<th>del(1)</th>
<th>del(1)</th>
<th>ins(L,00)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rebalance</td>
<td>rebalance</td>
<td>ins('001)</td>
</tr>
</tbody>
</table>

del uttered in old epoch $\Rightarrow$ can send to S
now up to date with Core
send rebalance

S replays del;
S replays rebalance;
intervening ins move;
S now in new epoch
Consistency without concurrency control in large, dynamic systems

Core Site

N

I

L

R

ins(L, 000)

Nebula Site

I

N

R

ins('0001)

ins(L, 000)

ins('0001)

ins arguments transformed to new epoch

Core replays ins
Summary

CRDT:
- Convergence ensured
- Design for commutativity

GC cannot be ignored
- Requires commitment
- Pervasive issue

Large-scale commitment:
- Core / Nebula
- To synchronise: catch-up + migration

CRDT = non-locking synchronisation in weak memory model
Future work

More CRDTs
Understanding CRDTs: what invariants can be CRDTized
Approximations of CRDTs
Data types for consistent cloud computing without concurrency control