Groupware Systems for fun and profit

CRDT, OT, Quorum-commit etc.

Max Klymyshyn

July 2, 2017

https://github.com/joymax/odessajs2017

Groupware or Collaborative System

synchronization in asynchronous environments

Real-time groupware systems are multi-user systems where the actions of one user **must quickly be propagated to the other users**.

Examples of Groupware Systems:

- Any kind of chats with clients on **multiple devices**
- Real-time group editing functionality (google docs etc.)
- Real-time games
- Conferences software
- Notes, contacts, reminders, calendars
- Shopping carts, group ordering

Why?

It matters for JavaScript and especially for client-side developers:

- **backend-free systems** – It's is possible to create and maintain completely back-end less **p2p architecture** using WebRTC, δ**-mutation** and different types of sharding.
- consistent behavior for devices with same service (gmail, notes etc.)
- network-partition tolerance

Synchronization in asynchronous environment

and different types of distributed replication approaches

Here is list of fundamental problems within asynchronous setting:

- **divergence** *Final document contents are different on all sites*
- **causality-violations** *Execution order is different eq* $o_1 \rightarrow o_3$
- **intention-violoation** *the actual effect is different from intended effect eq* $o_1 || o_2$

Replication and Consistency

and different types of distributed replication approaches

Basically there's only two hight-level approaches for replications: **Pessimistic** and **Optimistic** replications.

Pessimistic Replication

Or single-copy serializability: The idea is to prohibit access to a replica unless it is provably up to date. Techniques: two-phase commit, quorum-consensus, same-order delivery.

Optimistic Replication or Eventual Consistency

Updates sent asynchronously to other replicas; every replica eventually applies all updates, possibly in different orders

Naive approaches

to organize synchronization within groupware systems

- Locking
- Transactions
- Single Active Participant
- Dependency Detection
- Reversible Execution

Consistency Models

- **Strong Consistency** – after the update completes, any subsequent access will return the updated value
- **Weak Consistency** – the system does not guarantee that subsequent accesses will return the updated value
- **Eventual Consistency** – eventually all accesses will return the last updated value.
- **Strong Eventual Consistency** – special case of EC which adds the safety guarantee that any two nodes that have received the same (unordered) set of updates will be in the same state

Operation Transformation (OT) System Model

and definitions

OT

is a technology for supporting a range of collaboration functionalities in advanced collaborative software systems.

- Definitions
  - Processes or sites are participants of the collaborative system
  - Commands or operations are operations provided by system *(for example insert text, delete text etc.)*
  - Required properties and Transformations Matrices

Transformation Property

Operations must bee commutative:

$$op_1 \odot op_2 = op_2 \odot op_1$$

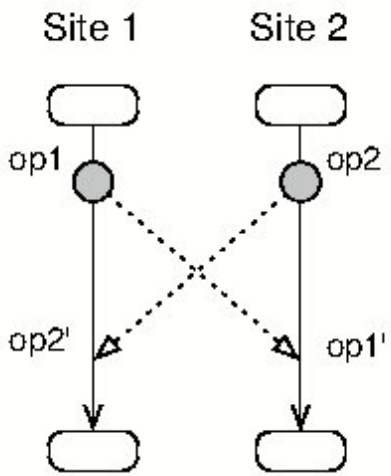


Figure: order of operations for different sites

Transformation Matrix

formal definition

$$o_j^{'} = T(o_j, o_i, p_j, p_i)$$

$$o_i^{'} = T(o_i, o_j, p_i, p_j)$$

then $\mathbf{T}$ is such that following is satisfied:

$$o_j^{'} \odot o_i^{'} = o_i^{'} \odot o_j^{'}$$

Transformation Function picture

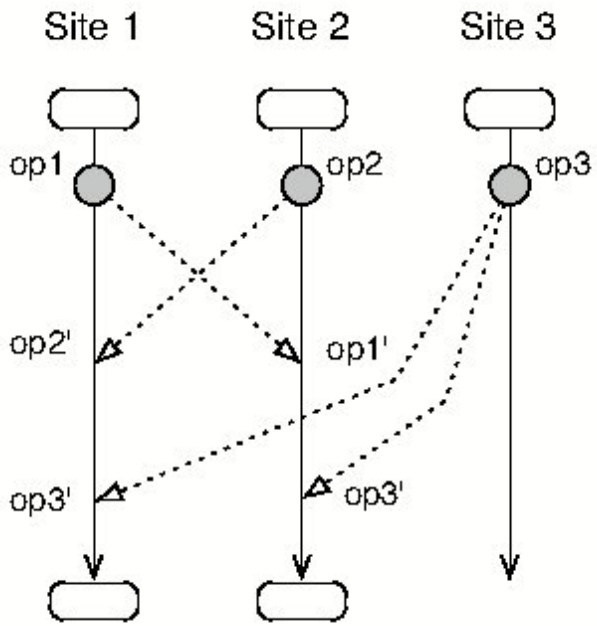


Figure: Transformation Function

OT Summary: Intention-violation

- Famous algos: GROVE, dOPT, Jupiter, adOPTed, REDUCE, GOT, GOTO
- Proving correctness of transformation functions is a complex task
- Impossible to establish proofs without computer within reasonable time for complex TF
- Following Oster et al. "Proving correctness of transformation functions in collaborative editing systems" paper only OT implementation of Tombstone Transformation Functions is correct

Who is using OT?

- Google Docs
- ex-Google Wave
- A lot of other experiments

OT Ressel et al. model

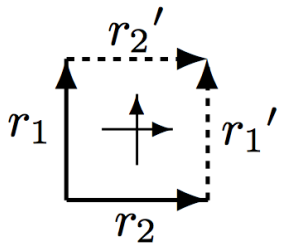
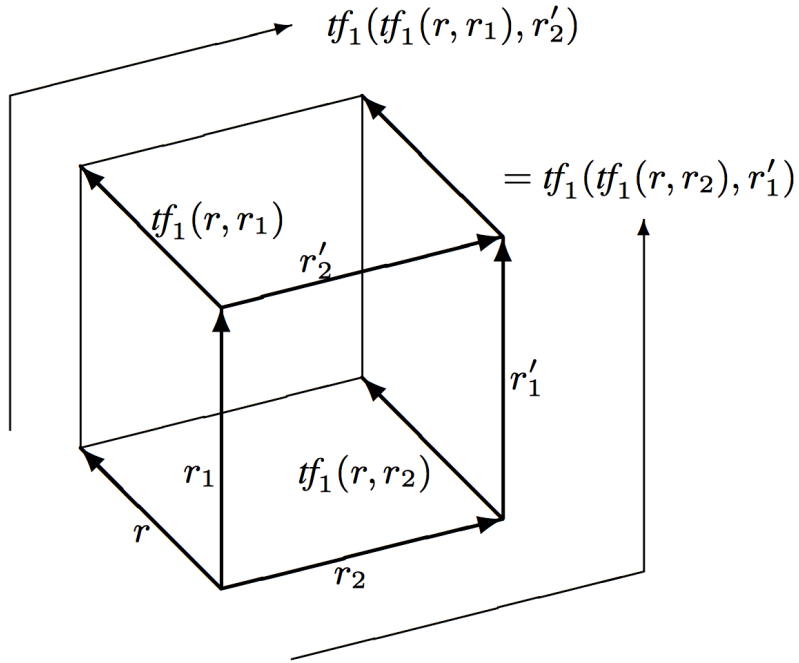


Figure: L-Transformation:
$r_1 \odot r_2' \equiv r_2 \odot r_1'$



Conflict-free replicated data types (CRDT)

CRDT is

data type for the system of processes interconnected by an asynchronous network where replicas can be updated independently and concurrently without coordination between the replicas.

This kind of replication is very handy for client-side developers because:

- Does not require active back-end system participation
- Quality leap in UX for users with weak or blinking network connections
- Better experience for SPAs with long single-load sessions like SoundCloud or GMail
- Offline work

CRDTs: CvRDT, CmRDT

state-based and op-based replication

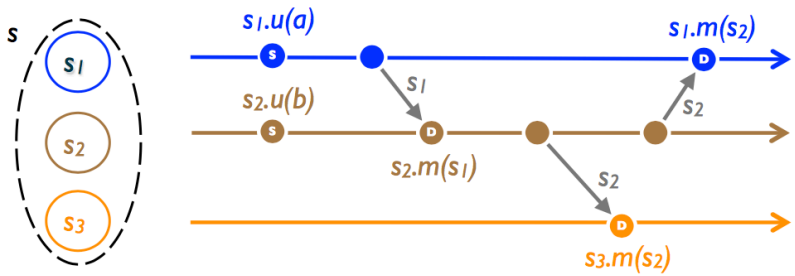


Figure: **State-based** Convergent Replicated Data Type or CvRDT

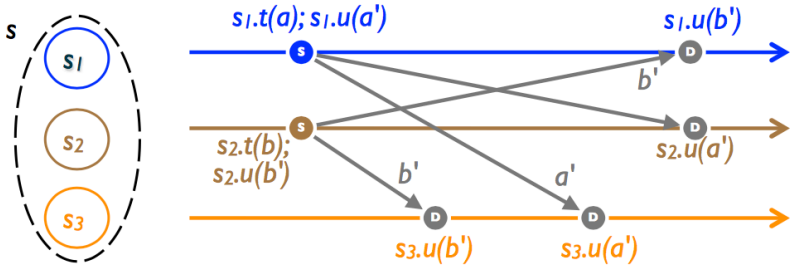


Figure: **Op-based** Commutative Replicated Data Type or CmRDT

CvRDT and CmRDT Properties

Required properties

In Abstract Algebra a CRDT is formally known as a Semilattice. All Semilattices must have the following properties:

- **commutativity** – $\forall x, y : x \odot y = y \odot x$
- **associativity** – nodes must come to the exactly same state regardless of the order they receive information:
  $x \odot (y \odot z) = (x \odot y) \odot z$
- **idempotency** – $x \odot x = x$

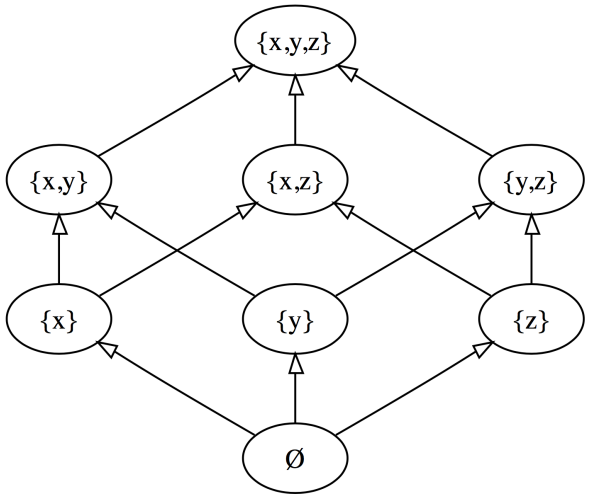


Figure: Least Upper Bound

CRDT: Types

Registers, Counters, Sets

- Register: LWW or Multi-Value (Dynamo or Couchdb-like)
- Counter (growth-only) and Counter w/decrementing
- G-Set – growth-only set
- 2P-Set – remove only once set (G-Set + Tombstones set)
- LWW-Element-Set – vector clocks
- OR-Set – unique-tagged elements and list of tags within Tombstones set
- Treedoc, Logoot etc.

CRDT: GCounter

Increment only counter

```
class GCounter {
  constructor(n) { this.n = n; }

  increment() { return new GCounter(this.n + 1); }

  query() { return this.n; }

  merge(counter) {
    return new GCounter(
      Math.max(counter.query(), this.n));
  }
}
```

CRDT: PNCounter

Increment and decrement counter

```
class PNCounter {
  constructor(p, n) { this.n = n; this.p = p}

  increment() { return new PNCounter(
    this.p.increment(), this.n); }

  decrement() { return new PNCounter(
    this.p, this.n.increment()); }

  query() { return this.p.query() - this.n.query(); }

  merge(pncounter) {
    return new PNCounter(this.p.merge(pncounter.p),
                         this.n.merge(pncounter.n));
  }
}
```

CRDT: PNCounter Test

Increment and decrement counter

```
function examples_PNCounter() {
  let log = (op, c1, c2) => console.log(
    "[OP] " + op + ": c1=" + c1.query() + ", c2=" + c2.query());
  let pn_counter1 = new PNCounter(
    new GCounter(0), new GCounter(0));
  let pn_counter2 = new PNCounter(
    new GCounter(0), new GCounter(0));

  pn_counter1 = pn_counter1.increment();
  pn_counter1 = pn_counter1.increment();
  log("[BEFORE MERGE] pncounter1++", pn_counter1, pn_counter2);
  pn_counter2 = pn_counter2.merge(pn_counter1);
  log("[AFTER MERGE] pncounter1++", pn_counter1, pn_counter2);
  pn_counter2 = pn_counter2.decrement();
  log("[BEFORE MERGE] pncounter2--", pn_counter1, pn_counter2);
  pn_counter1 = pn_counter1.increment();
  log("[BEFORE MERGE] pncounter1++", pn_counter1, pn_counter2);
  pn_counter1 = pn_counter1.merge(pn_counter2);
  pn_counter2 = pn_counter2.merge(pn_counter1);
  log("[AFTER MERGE]", pn_counter1, pn_counter2);
}
```

CRDT: PNCounter example output

```
[OP] [BEFORE MERGE] pncounter1++: c1=2, c2=0
[OP] [AFTER MERGE] pncounter1++: c1=2, c2=2
[OP] [BEFORE MERGE] pncounter2--: c1=2, c2=1
[OP] [BEFORE MERGE] pncounter1++: c1=3, c2=1
[OP] [AFTER MERGE]: c1=2, c2=2
```

CRDT: OR-Set

Observed-Remove Set

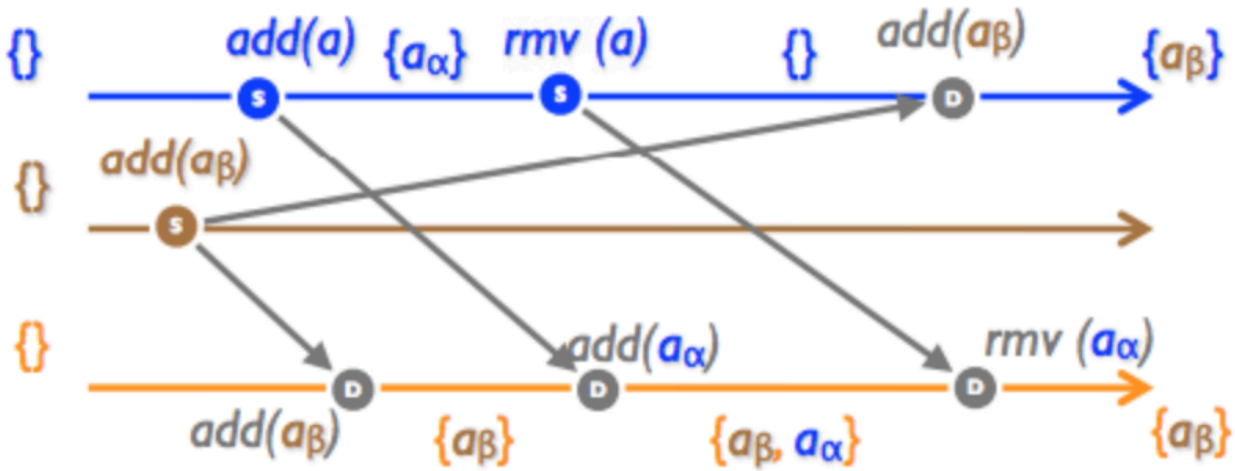


Figure 14: Observed-Remove Set (op-based)

Existing Tools

- ShareDB (and fall of Google Wave)
- Swarm (and forever-in-pre-alpha tool)
- OT.js (Operational Transformation for JS)
- CRDT – github.com/dominictarr/crdt
- replikativ.io – p2p distributed system framework
- other CRDT implementations

Who is using CRDTs?

- Facebook
- TomTom
- League of Legends
- SoundCloud
- Bet265
- RIAK Distributed Database

Drawbacks of CRDT

you can't pour two buckets of shit into one bucket

However, a major drawback in current state-based CRDTs is **the communication overhead of shipping the entire state**, which can get very large in size. For instance, the size of a counter CRDT (a vector of integer counters, one per replica) *increases with the number of replicas*; whereas in a grow-only Set, the state size depends on the set size, that grows as more operations are invoked.

δ-mutation

Everything at a cost

- "state fragments" merged using desired semantics
- δ-mutators, defined in such a way to return a delta-state, typically, with a much smaller size than the full state.

delta-enabled-crdts

References

[1] Marc Shapiro, Nuno Pregui ca, Carlos Baquero, Marek Zawirski. *A comprehensive study of Convergent and Commutative Replicated Data Types.*

[2] Yasushi Saito, Marc Shapiro *Replication: Optimistic Approaches*

[3] Mihai Let, Nuno Pregui, Marc Shapiro *CRDTs: Consistency without concurrency control∗*

[4] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels *Dynamo: Amazon's Highly Available Key-value Store*

[5] G´erald Oster, Pascal Urso, Pascal Molli, Abdessamad Imine. *Real time group editors without Operational transformation.*

[6] Gérald Oster, Pascal Urso†, Pascal Molli, Abdessamad Imine *Proving correctness of transformation functions in collaborative editing systems*

[1] Paulo S´ergio Almeida, Ali Shoker, and Carlos Baquero *Efficient State-based CRDTs by Delta-Mutation*

[2] A. Bieniusa, M. Zawirski, N. Preguiça, M. Shapiro, Carlos Baquero, Valter Balegas, Sérgio Duarte. *An Optimized Conflict-free Replicated Set.*

[3] C.A. Ellis, S.J. Gibbs *Concurrency Control in Groupware Systems*

[4] Pascal Molli, Ge´rald Oster *Using the Transformatinal Approach to Build a Safe and Generic Data Synchronizer*

[5] Pascal Molli, Ge´rald Oster *Using the Transformatinal Approach to Build a Safe and Generic Data Synchronizer*

[6] Paulo S´ergio Almeida, Ali Shoker, and Carlos Baquero *Delta State Replicated Data Types*, 2016

Thanks

@maxmaxmaxmax

https://github.com/joymax/odessajs2017