Multi-Client Syncing Strategies

Todd Kennedy

> whoami

Todd Kennedy
@whale_eat_squid
CTO, Scripto
Beard grower

We have a problem

We want to be able to let multiple people edit the same document ...but merge conflicts are bad

Luckily there are solutions

- Operational Transform (Google Docs, Wave, Etherpad)
- Differential Synchronization (O.G. Google Docs, Gedit)
- Conflict-Free Replicated Data Types (RIAK, Soundcloud)

Differential Synchronization

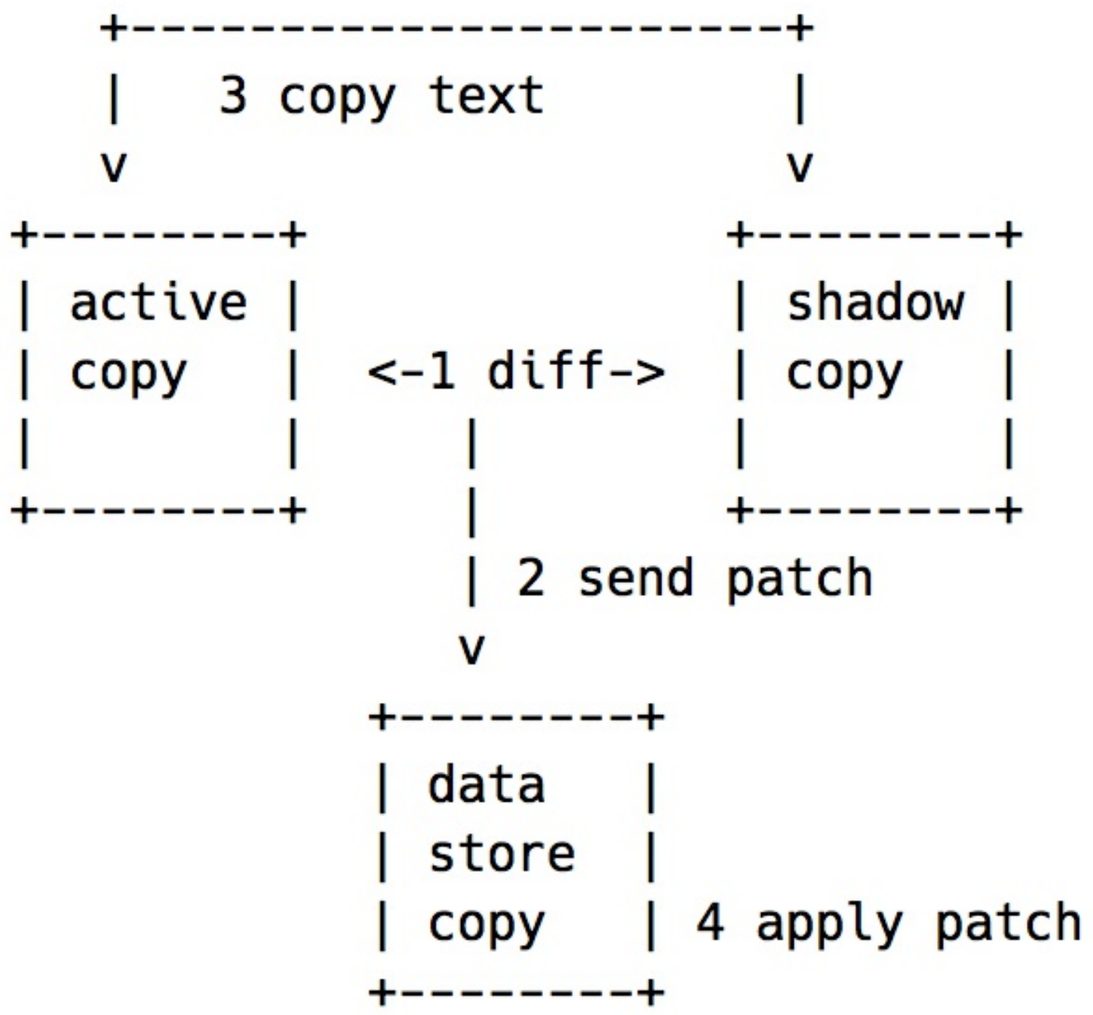
Neil Fraiser at Google in 2009 (white paper)

- Original concept for google docs
- Uses a character based diff to traffic changes

A basic example

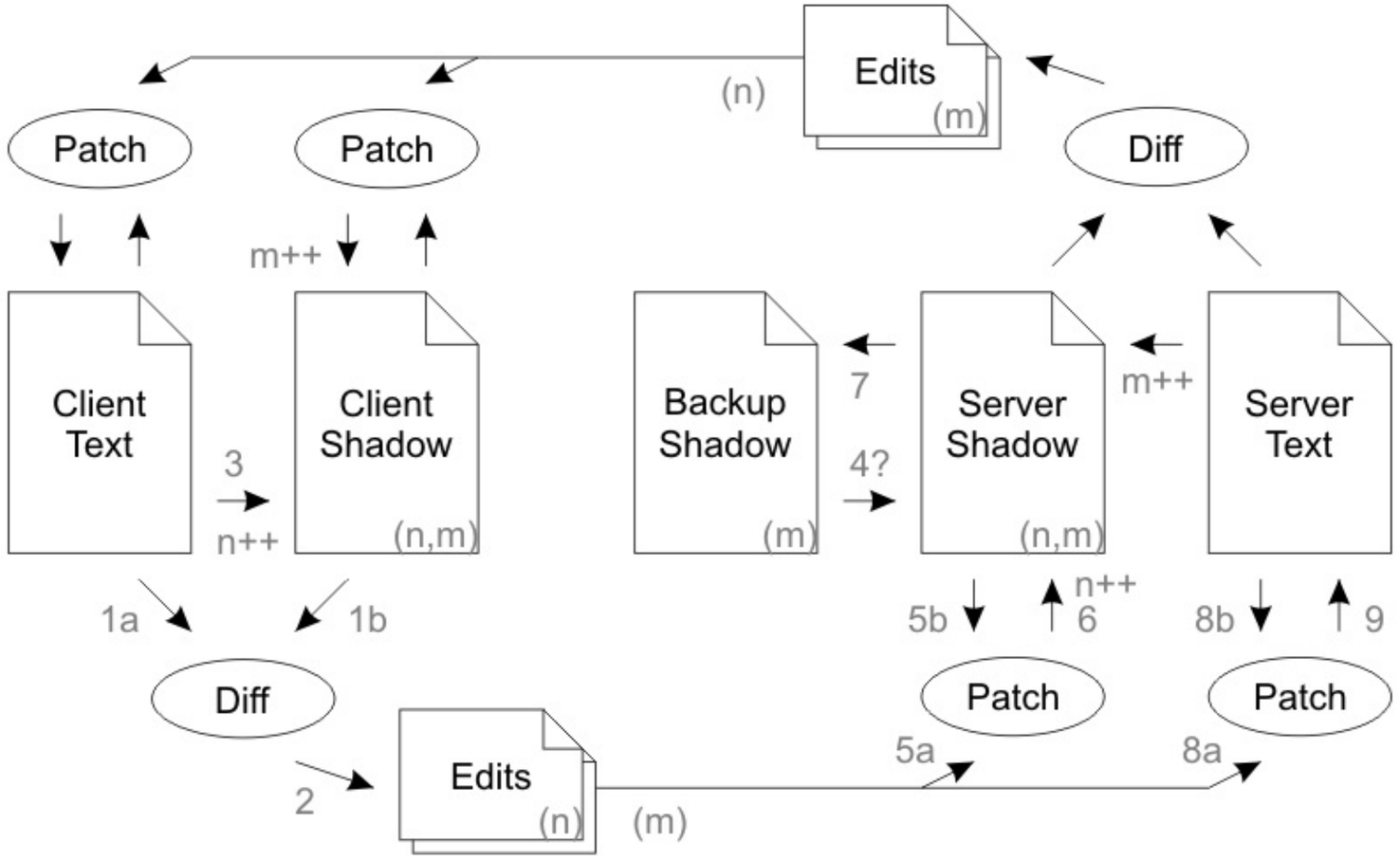
In a basic, non-networked setup, there are two copies of the text that may be edited at anytime: the copy you're actively working on and the copy stored in your datastore.

1. Each operation in the active copy is diffed against a *shadow* copy, creating a diff
2. This diff is handled to the datastore
3. The current version of the active copy becomes the *shadow* copy
4. The diff is applied as a patch against the datastore


3 copy text
active copy
<-1 diff->
shadow copy
2 send patch
data store copy
4 apply patch

Simple, huh?

Keeping muliple remote clients in sync requires 5 copies PER user



Whats good?

- Much simpler than OT & CRDT (for various definitions of "simplier")
- Allows for out of order application of changes
- Can work without central server

Whats bad?

- Scaling is complex & memory intensive
- Diff-Match-Patch is hard for structured data
- Can't track user performing edit in-band

Conflict-Free Replicated Data Types

Two types of CRDTs

Commutative Replicated Data Types

- Operation-based
- Commutative but not idempotent
- Ops can arrive in any order, but must only arrive once

Convergent Replicated Data Types

- State-based
- Requires sending a lot data over wire (all state)
- Requires merge to be commutative, associative and idempotent

WOOT (WithOut Operational Transform

A CRDT-based method for document editing

Whats Good

- Does not require a central server
- Less complex than OT (debateable!)

Whats Bad

- Can't delete data. Seriously, only hide it

Operational Transform

Developed at MCTC in Austin, TX 1989 & Xerox Parc in 1995 & Google in mid 2000s

Serialization and broadcast of specific operations performed on a shared document of equal length, with respect to the document cursor

Basic operations

- insertCharacters
- deleteCharacters
- retain

Example

Lets change "I like seattle" to "I like Seattle"

- `retain(7)`
- `deleteCharacters('s')`
- `insertCharacters('S')`
- `retain(6)`

So.... operations

How do we use them though?

ENTER TRANSFORM

The transform method is the heart of OT — it can apply operations on top of a document without requiring locking and resolving conflicts in a 'sane' fashion

Transform applies changesets to documents of the same length

All the characters in the retain, insert & delete operations must add up to the length of the current document the transform is being applied to

A better example

Two users editing a document that is the characters `Ta`

- User 1 inserts `o`
- User 2 inserts `p`

Or, in transforms:

```
retain(2), insertCharacters('o')
```

```
retain(2), insertCharacters('p')
```

The document has changed in the client and the server, but to two different states.

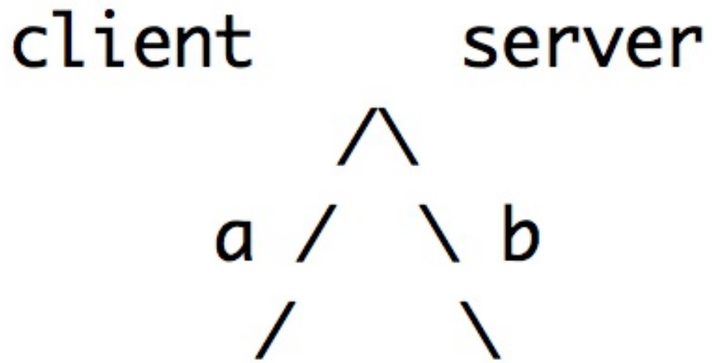
State A adds `o` to the document. State B adds `p`

Now we need to reconcile the two states so that the unified document is in agreement again

Putting both changesets into the `transform` method returns two new changesets that can be applied to the current document state respectively

```
const [a2, b2] = transform(a, b)
```

...but only because they're based on the same HEAD revision

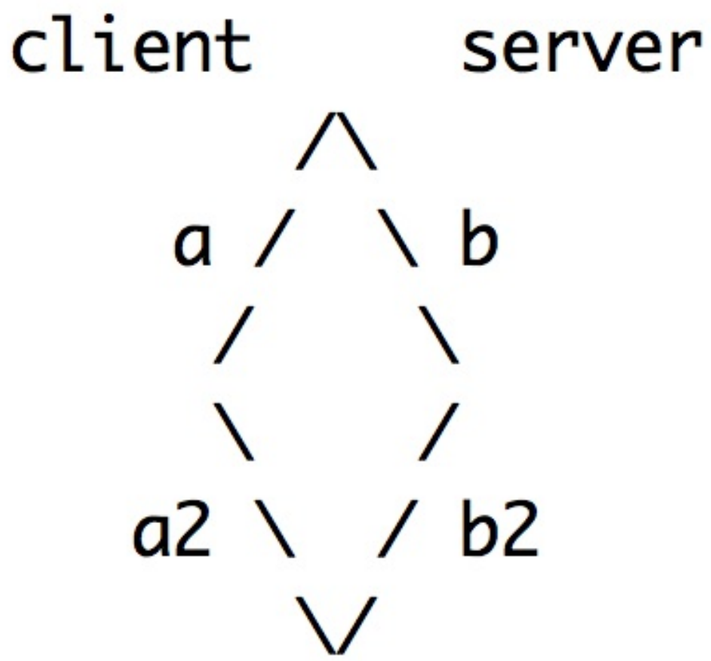


`transform` returns an `a2` that looks like:

```
retain(3), insertCharacters('o')
```

Now we can apply to document state A and `b2` to B and achieve singularity!

By doing that to the document (which is now `Tap`) and we end up with `Tapo`!

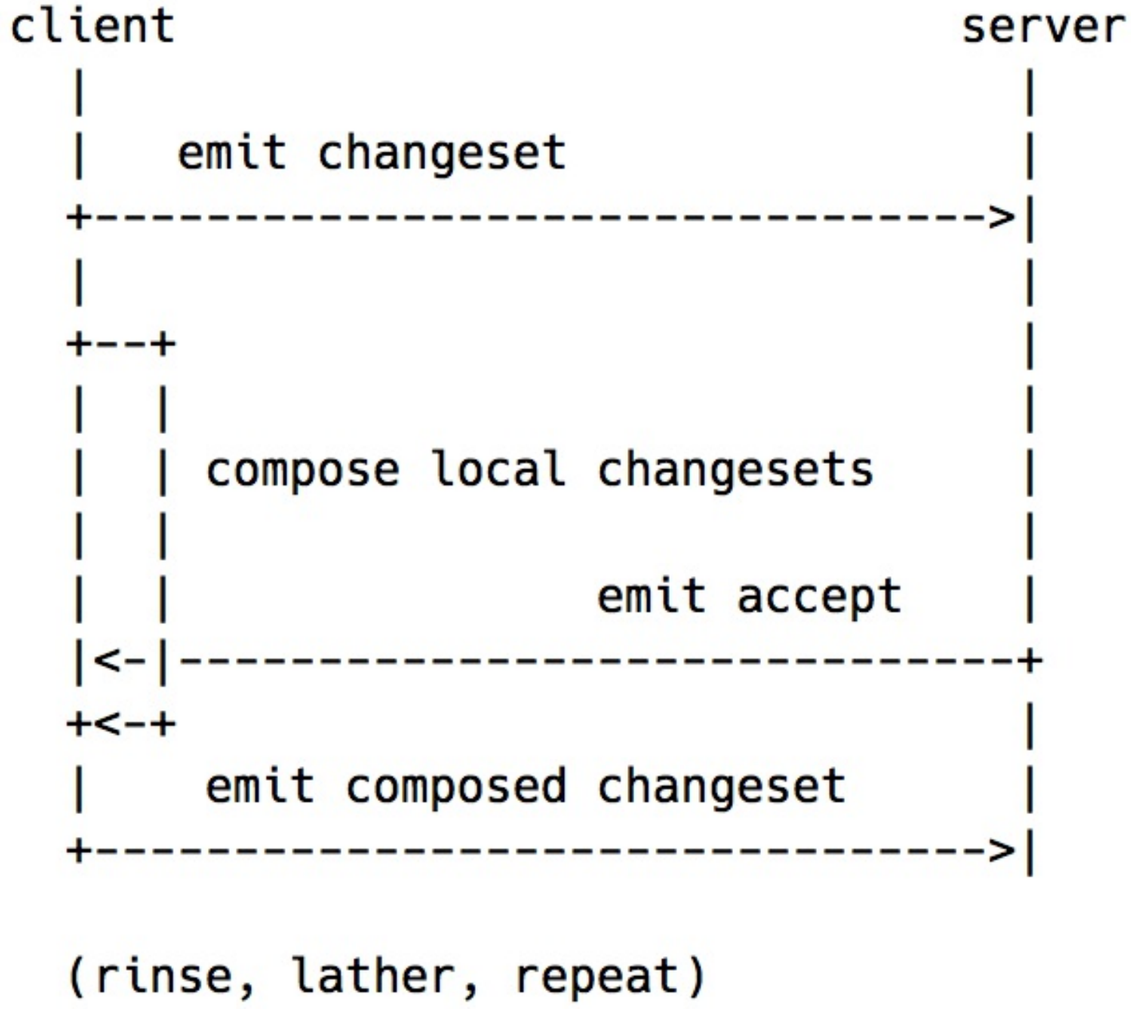


Huh? Tapo isn’t a word

No, but it's a conflict-free resolution to the issue — better than git telling you that your head is detacted and you need a three-way merge!

In a more complex scenario you'll be dealing with a lot more changesets with the same parent revision that will conflict. Most OT systems resolve this with a first-to-the-server strategy...

...since the server mediates the changesets between the clients

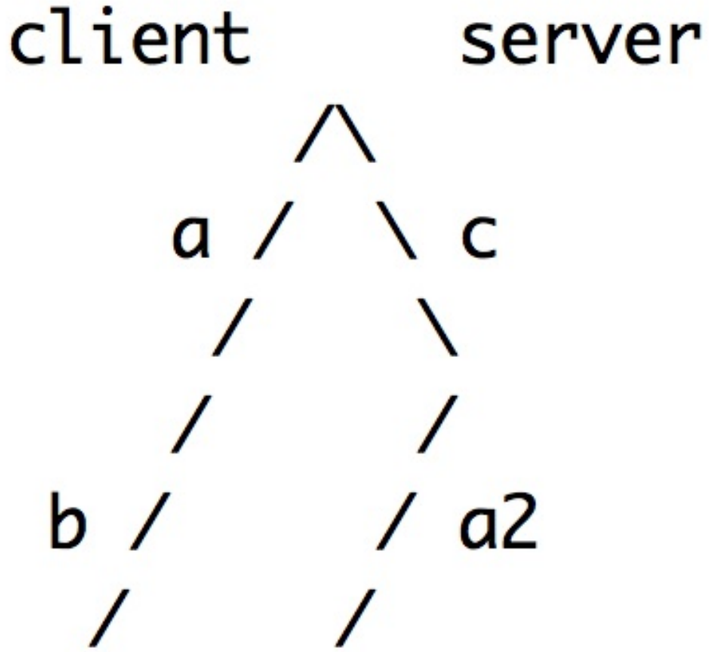


When the server accepts a commit message it

- assigns it a unique identifier (usually either a monotonically increasing integer or a SHA1 hash of the current document state).
- sends a `accept` message to the originating client
- broadcasts the change as to the rest of the connected clients

In reality...

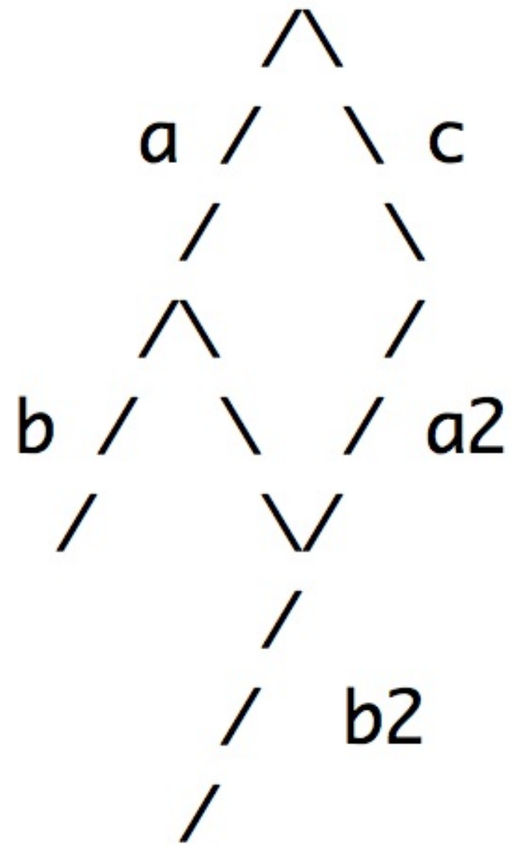
This is a way more likely scenario to encounter: the server and client are diverged by more than one state



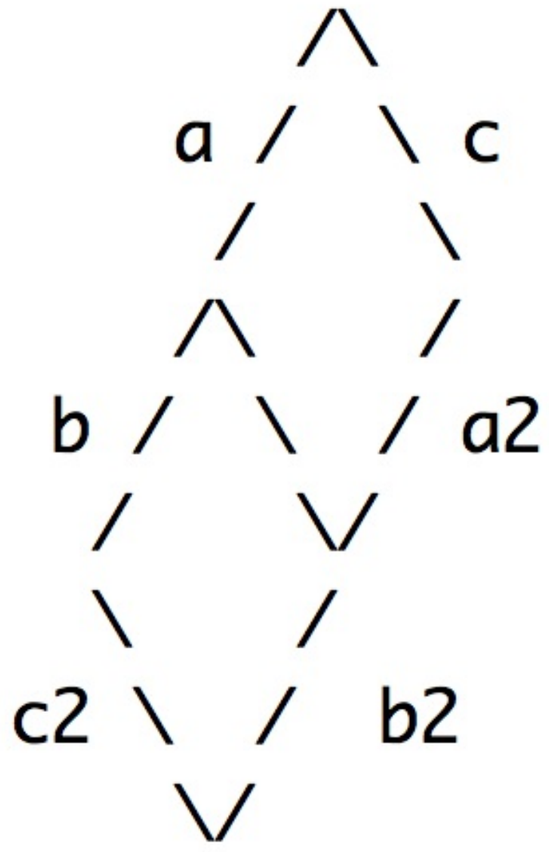
Thankfully the `transform` method allows us to resolve for this state as well.

In the simple example we discarded state `b2` since the client was disinterested in it and only sent `a2` to the server. Here, we need to use that to generate a new "bridge" transform.

By transforming `b` and `a2` we can derive `b2`



And keeping with that, we can also transform `b2` and `a2` against `c` to get `c2` which we can apply to this document. This "stepping" application can be applied on any number of changesets to derive any intermediate state so long as one shared revision exists.



That seems kind of laborious

It is! Not only that but it's Big O is `O(n log n)`!

This complexity makes it difficult to support large numbers of clients performing operations on the same document.

Lets just compose ourselves

Wave's improvement on this process is the `compose` function which is `O(n)`.

Composes takes changesets performed on the same document and combines them into one changeset.

So instead of transforming `c` against `b2` and `a2` we can compose the latter into `ab2` and just `transform(ab2, c)`

Thank you!

Resources

- Concurrency Control in Groupware Systems
- High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System
- Understanding and Applying Operational Transform
- Google Wave Operational Transform
- Neil Fraser's Google Tech Talk on Differential Sync
- WithOut Operational Transform
- WOOT for JavaScript and Scala
- Operational Transform JS Library
- Differential Synchronization