Etherpad and EasySync Technical Manual

AppJet, Inc., with modifications by the Etherpad Foundation

December 9, 2018

Contents

1 Documents

- A document is a list of characters, or a string.
- A document can also be represented as a list of *changesets*.

2 Changesets

- A changeset represents a change to a document.
- A changeset can be applied to a document to produce a new document.
- When a document is represented as a list of changesets, it is assumed that the first changeset applies to the empty document, [].

3 Changeset representation

$$(\ell \to \ell')[c_1, c_2, c_3, ...]$$

where

ℓ is the length of the document before the change,

ℓ' is the length of the document after the change,

$[c_1, c_2, c_3, ...]$ is an array of ℓ' characters that described the document after the change.

Note that $\forall c_i : 0 \leq i \leq \ell'$ is either an integer or a character.

- Integers represent retained characters in the original document.
- Characters represent insertions.

4 Constraints on Changesets

- Changesets are canonical and therefor comparable. When represented in computer memory, we always use the same representation for the same changeset. If the memory representation of two changesets differ, they must be different changesets.
- Changesets are compact. Thus, if there are two ways to represent a changeset in computer memory, then we always use the representation that takes up the fewest bytes.

Later we will discuss optimizations to changeset representation (using "strips" and other such techniques). The two constraints must apply to any representation of changesets.

5 Notation

- We use the algebraic multiplication notation to represent changeset application.
- While changesets are defined as operations on documents, documents themselves are represented as a list of changesets, initially applying to the empty document.

Example $A = (0 \to 5)[\text{“}hello\text{”}]$ $B = (5 \to 11)[0 - 4, \text{“ }world\text{”}]$

We can write the document “hello world” as $A \cdot B$ or just AB. Note that the “initial document” can be made into the changeset $(0 \to N)[\text{“} < \textit{the document text} > \text{”}]$.

When A and B are changesets, we can also refer to (AB) as “the composition” of A and B. Changesets are closed under composition.

6 Composition of Changesets

For any two changesets A, B such that

$$A = (n_1 \to n_2)[\cdots]$$

$$B = (n_2 \to n_3)[\cdots]$$

it is clear that there is a third changeset $C = (n_1 \to n_3)[\cdots]$ such that applying C to a document X yields the same resulting document as does applying A and then B. In this case, we write $AB = C$.

Given the representation from Section 3, it is straightforward to compute the composition of two changesets.

7 Changeset Merging

Now we come to realtime document editing. Suppose two different users make two different changes to the same document at the same time. It is impossible to compose these changes. For example, if we have the document X of length n, we may have $A = (n \to n_a)[\ldots n_a\text{characters}]$, $B = (n \to n_b)[\ldots n_b\text{characters}]$ where $n \neq n_a \neq n_b$.

It is impossible to compute $(XA)B$ because B can only be applied to a document of length n, and (XA) has length n_a. Similarly, A cannot be applied to (XB) because (XB) has length n_b.

This is where *merging* comes in. Merging takes two changesets that apply to the same initial document (and that cannot be composed), and computes a single new changeset that preserves the intent of both changes. The merge of A and B is written as $m(A, B)$. For the Etherpad system to work, we require that $m(A, B) = m(B, A)$.

Aside from what we have said so far about merging, there are many different implementations that will lead to a workable system. We have created one implementation for text that has the following constraints.

8 Follows

When users A and B have the same document X on their screen, and they proceed to make respective changesets A and B, it is no use to compute $m(A, B)$, because $m(A, B)$ applies to document X, but the users are already looking at document XA and XB. What we really want is to compute B' and A' such that

$$XAB' = XBA' = Xm(A, B)$$

"Following" computes these B' and A' changesets. The definition of the "follow" function f is such that $Af(A, B) = Bf(B, A) = m(A, B) = m(B, A)$. When we compute $f(A, B)$

- Insertions in A become retained characters in $f(A, B)$
- Insertions in B become insertions in $f(A, B)$
- Retain whatever characters are retained in *both* A and B

Example Suppose we have the initial document $X = (0 \to 8)[\text{“}baseball\text{”}]$ and user A changes it to "basil" with changeset A, and user B changes it to "below" with changeset B.

We have $X = (0 \to 8)[\text{“}baseball\text{”}]$
$A = (8 \to 5)[0 - 1, \text{“}si\text{”}, 7]$
$B = (8 \to 5)[0, \text{“}e\text{”}, 6, \text{“}ow\text{”}]$

First we compute the merge $m(A, B) = m(B, A)$ according to the constraints

$$m(A, B) = (8 \to 6)[0, \text{”}e\text{”}, \text{”}si\text{”}, \text{”}ow\text{”}] = (8 \to 6)[0, \text{“}esiow\text{”}]$$

Then we need to compute the follows $B' = f(A, B)$ and $A' = f(B, A)$.

$$B' = f(A, B) = (5 \to 6)[0, \text{“}e\text{”}, 2, 3, \text{“}ow\text{”}]$$

Note that the numbers 0, 2, and 3 are indices into $A = (8 \to 5)[0, 1, \text{“}si\text{”}, 7]$

0	1	2	3	4
0	1	s	i	7

$A' = f(B, A) = (5 \to 6)[0, 1, \text{”}si\text{”}, 3, 4]$
We can now double check that $AB' = BA' = m(A, B) = (8 \to 6)[0, \text{“}esiow\text{”}]$.

Now that we have made the mathematical meaning of the preceding pages complete, we can build a client/server system to support realtime editing by multiple users.

9 System Overview

There is a server that holds the current state of a document. Clients (users) can connect to the server from their web browsers. The clients and server maintain state and can send messages to one another in real-time, but because we are in a web browser scenario, clients cannot send each other messages directly, and must go through the server always. (This may distinguish from prior art?)

The other critical design feature of the system is that *A client must always be able to edit their local copy of the document, so the user is never blocked from typing because of waiting to send or receive data.*

10 Client State

At any moment in time, a client maintains its state in the form of 3 changesets. The client document looks like $A \cdot X \cdot Y$, where

A is the latest server version, the composition of all changesets committed to the server, from this client or from others, that the server has informed this client about. Initially $A = (0 \to N)[< \textit{initial document text} >]$.

X is the composition of all changesets this client has submitted to the server but has not heard back about yet. Initially $X = (N \to N)[0, 1, 2, \ldots, N-1]$, in other words, the identity, henceforth denoted I_N.

Y is the composition of all changesets this client has made but has not yet submitted to the server yet. Initially $Y = (N \to N)[0, 1, 2, \ldots, N-1]$.

11 Client Operations

A client can do 5 things.

1. Incorporate new typing into local state
2. Submit a changeset to the server
3. Hear back acknowledgement of a submitted changeset
4. Hear from the server about other clients' changesets
5. Connect to the server and request the initial document

As these 5 events happen, the client updates its representation $A \cdot X \cdot Y$ according to the relations that follow. Changes "move left" as time goes by: into Y when the user types, into X when change sets are submitted to the server, and into A when the server acknowledges changesets.

11.1 New local typing

When a user makes an edit E to the document, the client computes the composition $(Y \cdot E)$ and updates its local state, i.e. $Y \leftarrow Y \cdot E$. I.e., if Y is the variable holding local unsubmitted changes, it will be assigned the new value $(Y \cdot E)$.

11.2 Submitting changesets to server

When a client submit its local changes to the server, it transmits a copy of Y and then assigns Y to X, and assigns the identity to Y. I.e.,

1. Send Y to server,
2. $X \leftarrow Y$
3. $Y \leftarrow I_N$ (the identity).

This happens every 500ms as long as it receives an acknowledgement. Must receive ACK before submitting again. Note that X is always equal to the identity before the second step occurs, so no information is lost.

11.3 Hear ACK from server

When the client hears ACK from server,
$A \leftarrow A \cdot X$
$X \leftarrow I_N$

11.4 Hear about another client's changeset

When a client hears about another client's changeset B, it computes a new A, X, and Y, which we will call A', X', and Y' respectively. It also computes a changeset D which is applied to the current text view on the client, V. Because AXY must always equal the current view, $AXY = V$ before the client hears about B, and $A'X'Y' = VD$ after the computation is performed.

The steps are:

1. Compute $A' = AB$
2. Compute $X' = f(B, X)$
3. Compute $Y' = f(f(X, B), Y)$
4. Compute $D = f(Y, f(X, B))$
5. Assign $A \leftarrow A'$, $X \leftarrow X'$, $Y \leftarrow Y'$.
6. Apply D to the current view of the document displayed on the user's screen.

In steps 2,3, and 4, f is the follow operation described in Section 8.

Proof that $\mathbf{AXY = V \Rightarrow A'X'Y' = VD}$. Substituting $A'X'Y' = (AB)(f(B,X))(f(f(X,B),Y))$, we recall that merges are commutative. So for any two changesets P and Q,

$$m(P,Q) = m(Q,P) = Qf(Q,P) = Pf(P,Q)$$

Applying this to the relation above, we see

$$\begin{aligned} A'X'Y' &= ABf(B,X)f(f(X,B),Y) \\ &= AXf(X,B)f(f(X,B),Y) \\ &= AXYf(Y,f(X,B)) \\ &= AXYD \\ &= VD \end{aligned}$$

As claimed.

11.5 Connect to server

When a client connects to the server for the first time, it first generates a random unique ID and sends this to the server. The client remembers this ID and sends it with each changeset to the server.

The client receives the latest version of the document from the server, called HEADTEXT. The client then sets

$A \leftarrow$ HEADTEXT

$X \leftarrow I_N$

$Y \leftarrow I_N$

And finally, the client displays HEADTEXT on the screen.

12 Server Overview

Like the client(s), the server has state and performs operations. Operations are only performed in response to messages from clients.

13 Server State

The server maintains a document as an ordered list of *revision records*. A revision record is a data structure that contains a changeset and authorship information.

```
RevisionRecord = {
  ChangeSet,
  Source (unique ID),
  Revision Number (consecutive order, starting at 0)
}
```

For efficiency, the server may also store a variable called HEADTEXT, which is the composition of all changesets in the list of revision records. This is an optimization, because clearly this can be computed from the set of revision records.

14 Server Operations Overview

The server does two things in addition to maintaining state representing the set of connected clients and remembering what revision number each client is up to date with:

1. Respond to a client's connection requesting the initial document.
2. Respond to a client's submission of a new changeset.

14.1 Respond to client connect

When a server receives a connection request from a client, it receives the client's unique ID and stores that in the server's set of connected clients. It then sends the client the contents of HEADTEXT, and the corresponding revision number. Finally the server notes that this client is up to date with that revision number.

14.2 Respond to client changeset

When the server receives information from a client about the client's changeset C, it does five things:

1. Notes that this change applies to revision number r_c (the client's latest revision).
2. Creates a new changeset C' that is relative to the server's most recent revision number, which we call r_H (H for HEAD). C' can be computed using follows (Section 8). Remember that the server has a series of changesets,

   $$S_0 \to S_1 \to \ldots S_{r_c} \to S_{r_c+1} \to \ldots \to S_{r_H}$$

   C is relative to S_{r_c}, but we need to compute C' relative to S_{r_H}. We can compute a new C relative to S_{r_c+1} by computing $f(S_{r_c+1}, C)$. Similarly we can repeat for S_{r_c+2} and so forth until we have C' represented relative to S_{r_H}.
3. Send C' to all other clients
4. Send ACK back to original client
5. Add C' to the server's list of revision records by creating a new revision record out of this and the client's ID.

Additional topics

(a) Optimizations (strips, more caching, etc.)

(b) Pseudocode for composition, merge, and follow

(c) How authorship information is used to color-code the document based on who typed what

(d) How persistent connections are maintained between client and server