BuddyCode – design doc

Abstract

Requirements

- Collaborative editor
- Contributing to a document requires knowing it's unique id
- Sharing the document URL allows any user to join the session
- Any number of users able to contribute to the same document if they know the secret/url
- Synchronization to a backend DB for long term storage

Architecture

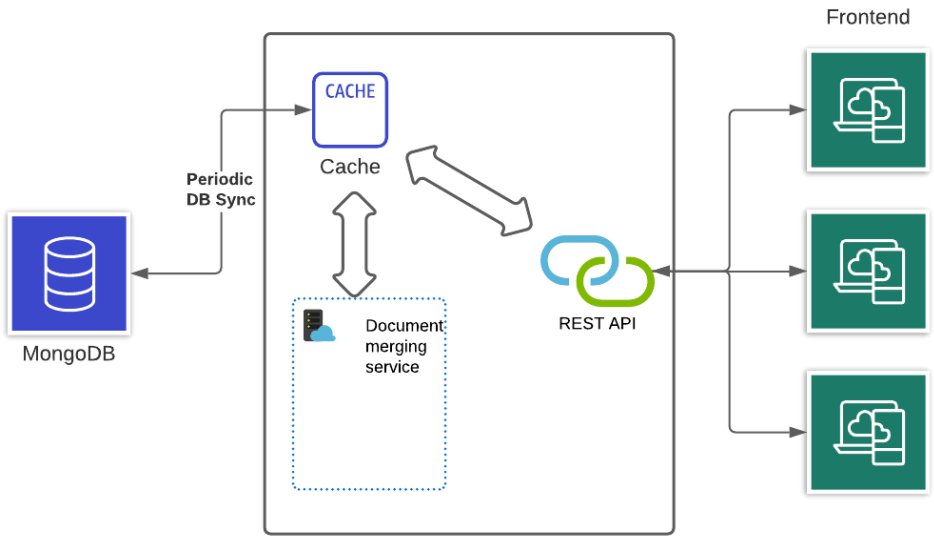
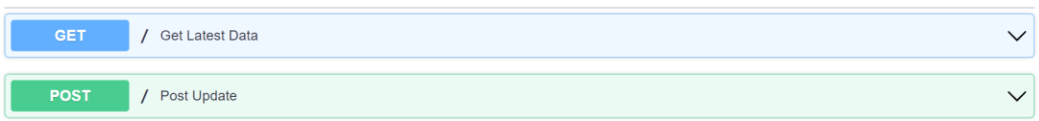


Figure 1 - Schematic of proposed BuddyCode architecture

API

Frontend <-> cache communication happens over a REST API. The API provides two endpoints:



Document synchronization

Brief overview of typical approaches and their limitations (adapted from: Neil Fraser - Differential Synchronization):

- Locking – a shared document may be edited by only one collaborator. A refinement could be implemented where subsections of the document are locked, thus facilitating multiple editors changing the document at any one time. This approach prevents close collaboration and unsuitable the connectivity is unreliable.
- Event passing – each "event", be it typing, cut, paste, replacement, etc. is captured and reflected for all users. Implementing this approach in a browser is non-trivial and is not naturally convergent (=a missed packet might lead to subsequent edits to be applied incorrectly).
- Three-way merges – clients send their document to the server which does n-1 merges (where n is number of concurrent users) and publishes the result to all clients. This approach is inherently half-duplex, meaning if you're typing, you're not getting updates. As this approach results in infrequent merges, merge-collisions are common. As such, automated merging becomes fragile and often requires manual conflict resolution.
- Differential synchronization – a symmetric algorithm (from the point of view of the client/server) where a cycle of backend diff/merge operations happen. As such, there is no requirement for "the chickens to stop moving so we can count them". Consequently, diffs are smaller, faster, and less likely to collide. Furthermore, this approach provides full-duplex channel so users can see document updates while editing their copy. (Note: can we control cursor position in the editor? If not, the cursor will jump every sync cycle and the user will be interrupted)

Differential synchronization

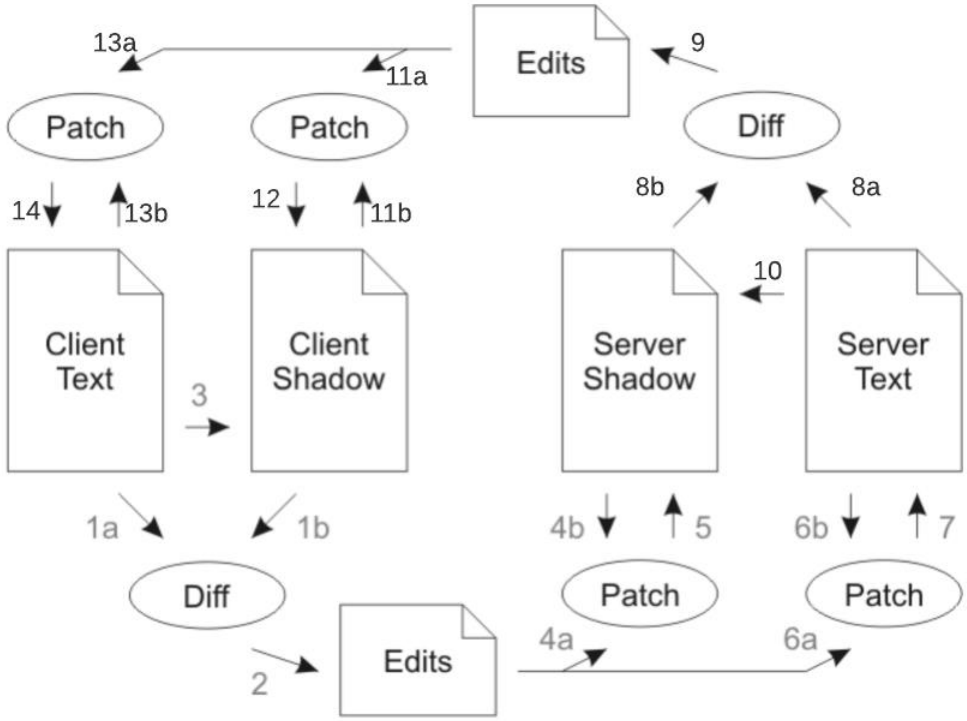


Figure 2 - Schematic of differential synchronization in a client-server scenario

Overview:

- Initially both client/server shadow are synchronized
- As both client/server potentially edited their text independently: c_text != c_shadow == s_shadow != s_text
- Synchronization occurs in two half cycles:
    - Diff(c_shadow, c_text) -> c_diff -> patch(c_diff, s_shadow) -> c_text == s_shadow
      -> patch(c_diff, s_text) -> c_text merged into s_text
    - Diff(s_shadow, s_text) -> s_diff -> patch(s_diff, c_shadow) -> s_text == c_shadow
      -> patch(s_diff, c_text) -> s_text+c_text changes merged Into c_text
      ,
- By the end of one full synchronization cycle, c_text == s_text
- Assuming c_text has changed before the synchronization completed, another cycle would bring documents back in sync

Details:

Although in the diagram above, the same patch is applied to both shadow and text, that is not the case. For this approach to work, the patch applied to the shadow should be fragile. A fragile patch can fail, in which case an a resync of the shadow needs to happen before continuing with the merge. On the other hand, the patch applied to the text is fuzzy and best effort. This kind of patch can find approximate location to apply the patch, even if the document has changed. However, if any patches failed to apply, they will show up in the next half-cycle and be patched out. An example data flow of this scenario from Neil Fraser - Differential Synchronization is reproduced below:

a. Client Text, Common Shadow and Server Text start out with the same string: "Macs had the original point and click UI."
b. Client Text is edited (by the user) to say: "Macintoshes had the original point and click interface." (edits underlined)
c. The Diff in step 1 returns the following two edits:

```
@@ -1,11 +1,18 @@
 Mac
+intoshe
 s had th
@@ -35,7 +42,14 @@
 ick
-UI
+interface
 .
```

d. Common Shadow is updated to also say: "Macintoshes had the original point and click interface."
e. Meanwhile Server Text has been edited (by another user) to say: "Smith & Wesson had the original point and click UI." (edits underlined)
f. In step 4 both edits are patched onto Server Text. The first edit fails since the context has changed too much to insert "intoshe" anywhere meaningful. The second edit succeeds perfectly since the context matches.
g. Step 5 results in a Server Text which says: "Smith & Wesson had the original point and click interface."
h. Now the reverse process starts. First the Diff compares Server Text with Common Shadow and returns the following edit:

```
@@ -1,15 +1,18 @@
-Macintoshes
+Smith & Wesson
 had
```

i. Finally this patch is applied to Client Text, thus backing out the failed "Macs" -> "Macintoshes" edit and replacing it with "Smith & Wesson". The "UI" -> "interface" edit is left untouched. Any changes which have been made to Client Text in the mean time will be patched around and incorporated into the next synchronization cycle.

Note: the client/server nomenclature above is for simplicities sake. As the approach is symmetric, the algorithm is peer-to-pee. In the next section, we'll describe scaling this approach to a multi-user scenario.

Scaling to a multi-user setting

Reproduced below are two schematics from Neil Fraser - Differential Synchronization. The schematics outline how to compose differential synchronization into a multiuser-server architecture. Furthermore, using the same approach, server to server synchronization could be implemented:

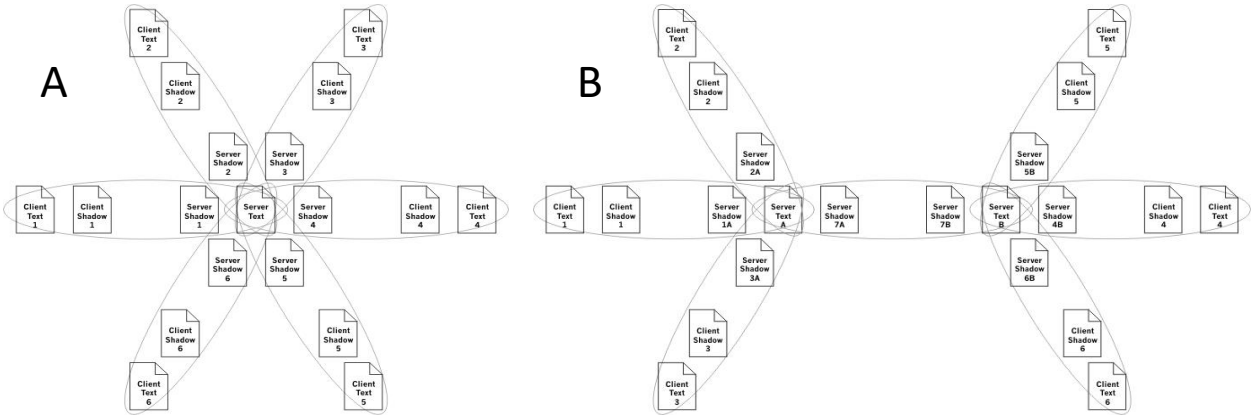


Figure 3 - Applying differential synchronization to a multiuser-server scenario can be seen in A. The same approach can be applied to server-server synchronization as per B

For further details, please refer to the source.