Special seminar to bachelor thesis

Simon Kocurek

University of Pavol Jozef Safarik in Kosice, Šrobárova 2, 041 80 Košice, Slovakia
rektor@upjs.sk

Abstract. In the paper we focus on solving the issues caused by going offline inside a state synchronizing network. In order to keep the state consistent across all clients in a network, various synchronization methods are used. These methods expect all clients to be available at any time and don't describe situations where some of the clients go offline and later attempt to reconnect. First we analyze a sample of synchronization methods, while comparing the ways they could support offline mode. After that we choose the most fitting method and provide a detailed description of possible modifications allowing client re-connections without loss of data or state conflicts. Finally we implement and compare these modifications.

Keywords: Synchronization · Offline · Implementation

1 Introduction

With time progressive web applications are becoming more and more popular[1]. This movement aims to improve user experience, security and portability of applications.[2] As a part of this movement it is becoming standard practice to allow editing single document by multiple users at the same time. Examples of this are Google Docs, or Dropbox web applications where you can see cursor of each user editing document you are looking at in real time.

Part of the progressive web application specification is making your application network independent[3]. This is achieved by using various persistent storage mechanisms provided by modern browsers[4].

When creating progressive web application with previously mentioned ability of multiple users to edit shared document in real-time, we arrive at an issue. The feature usually is not network independent. This issue stems from the way the collaborative system[5] is implemented, as it relies on usage of synchronization methods.

These methods are academically studied and proven to work. Some examples include usage of operational transformation[6], used in Google Docs application. And use of conflict-free replicated data types[8] used in Visual Studio Code Live Share[9].

OT has been studies as early as 1989[7]. Due to it's complexity many published articles about operational transformation were proven wrong. As they didn't eventually converge to a state, where all users working on same document see the same thing.

Use of conflict-free replicated data type is much newer method, which found use in distributed systems for sharing data. While it's main benefit is that it can work in peer to peer environment[11]. And is significantly simpler than OT.

The reason these methods aren't network independent is that they solve a problem of synchronizing state across all connected clients. In other words, they assume all clients have uninterrupted network connection. In the real world, however, this is not the case. Network partitions are something we need to count with[10]. This applies to data centers, public networks, but also home networks.

Synchronizing states becomes difficult once we start considering the possibilities of users editing, or deleting parts of document other users are working on. This causes so called conflicts. A concept also used in version control systems. To resolve these conflicts we have various strategies, varying in what input they get and if they require user intervention[12].

We will implement a solution that keeps the properties of the most suitable synchronization method, but is network independent, as well as able to resolve larger state conflicts without user intervention. This will improve the user experience of collaboration systems using progressive web applications, as the user no longer has to fear losing his work in case of network loss.

To allow easier integration of our solution into existing systems, we plan on creating a wrapper[13] on top of a synchronization method of choice that would handle saving user changes, as well as synchronizing with other clients upon re-connection.

Our goal is therefore to examine current synchronization methods, their implementations usable from the web-browser client and compare their pros and cons. After choosing the most suitable one, we will take a closer look at browser storage mechanisms, for saving user state in offline mode. Analyze current solutions for offline modes of progressive web applications and offline state synchronization. And use these findings during implementation.

1.1 Illustration example

To demonstrate the problems that current online collaboration systems face you can think of this example:

Let's suppose we have a progressive web application featuring collaboration system. All synchronization is done using a server-side as the source of truth. Suppose there is an application with 3 clients working together on one document. All 3 of them have stable connection and so their changes are merged together without any issues, or conflicts.

In this manner they continue working, until one of clients loses connection. From this moment on 2 different documents exist. The one on disconnected clients computer and the one shared across 2 clients with internet connection.

Traditionally all work form the disconnected client would be lost. This is the first problem we tackle in this thesis. Storing changes made even when there are no clients to share them with. This might possibly lead to redundant data being stored. To mitigate this problem we take space complexity of our solution into consideration.

After some time t, the disconnected client connects back to the network. Our main goal now is to merge the 2 completely different states into one without any interference or loss of work.

After merging all 3 clients again have the same document and can continue working.

1.2 Overview of current state

We approached the problem in order the goals were defined. First we tackled the current implementations of synchronization methods used in collaboration systems, also looking at their capability of supporting offline mode.

Differential Synchronization[14]

Meteor https://github.com/chfritz/meteor-serversync

Table 1. Synchronization method characteristics based on requirements

	CRDT	OT	DS
Low memory overhead	yes	yes	no
Undo support	no	no	yes
Client - Server	no	yes	yes
Optimistic updates	no	yes	no
Info about other users	yes	yes	yes

2 Suggested solution

By exploring options we decided for a client-server solution. We will test inside web browser environment. Use a library for CRDT and see how it goes. Ideally we want a library that builds on top.

We chose Local Storage as it is widely supported

What remains is start coding. Node-JS. https://github.com/automerge/automerge

Simple working of our solution is:

```
connected = has_internet_connection()

def on_reconnect():
    connected = True
    synchronize()

def on_disconnect():
    connected = False

def synchronize():
    while storage.has_data():
        operation, data = storage.get_data()
        sync[operation](data)

def insert(data):
    if connected:
        sync.insert(data)
    else:
        storage.insert(data)

def delete(data):
    if connected:
        sync.delete(data)
...
```

3 Conclusion

We have covered the issue we are solving in this paper. What are the benefits of solving it in our motivation. As well as why the issue exists in the first place.

By analyzing current state of the problem we were able to tell why collaboration systems usually don't consider the event of network partitions.

References

1. Karolina Gawron *"Should I Consider PWA?" Remarkable Possibilities of Progressive Web Apps.*
   www.monterail.com/blog/
   should-i-consider-pwa-remarkable-possibilities-of-progressive-web-apps
2. *Progressive Web App Checklist.*
   developers.google.com/web/progressive-web-apps/checklist
3. Chris Mills *Progressive web app advantages.*
   developer.mozilla.org/en-US/docs/Web/Apps/Progressive
   /Advantages#Network_independent
4. html5test.com/results/desktop.html
5. *Collaborative software.*
   en.wikipedia.org/wiki/Collaborative_software
6. *Operational transformation.*
   en.wikipedia.org/wiki/Operational_transformation
7. Ellis C. A., GIbbs S. J. *Concurrency Control in Groupware Systems. SIGMOD Rec., Issue Date: June 1989.*
8. *Conflict-free replicated data type.*
   en.wikipedia.org/wiki/Conflict-free_replicated_data_type
9. Amanda Silver *Introducing Visual Studio Live Share.*
   code.visualstudio.com/blogs/2017/11/15/live-share
10. *When do network partitions occur in practice?.*
    www.quora.com/When-do-network-partitions-occur-in-practice
11. Russell Sullivan *CRDTs explained - supercharge your serverless with CRDTs at the Edge.*
    serverless.com/blog/crdt-explained-supercharge-serverless-at-edge
12. Waldemar Korłub *Data synchronization Conflict resolution strategies.*
    enauczanie.pg.edu.pl/moodle/pluginfile.php/253757/
    mod_resource/content/0/07%20Data%20synchonization.pdf
13. *Wrapper definition.*
    www.techopedia.com/definition/4389/wrapper-software-engineering
14. Neil Fraser *Differential Synchronization. DocEng'09, Proceedings of the 2009 ACM Symposium on Document Engineering.*
    neil.fraser.name/writing/sync/eng047-fraser.pdf