Special seminar to bachelor thesis

Simon Kocurek

University of Pavol Jozef Safarik in Kosice, robrova 2, 041 80 Koice, 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 these states becomes more difficult when we start considering the possibility of one user deleting what the other just edited. Such issues, also called conflicts are addressed and solved by using already existing synchronization methods. Most of these are academically proven to work and so are a safe choice for implementation.

However, one big issue that isn't addressed by synchronization methods are users with flaky internet connection. When user disconnects all of his work is lost. This not only causes frustration, but also prevents these applications from being used in the enterprise environment, where losing progress is not acceptable.

Fixing this issue in a easy to implement manner could improve the experience of using web applications that allow collaboration of multiple users. And therefore could speed up incorporation of these applications in various spheres, such as education, management or communication. Where synchronization methods are already in use, but haven't gained much popularity. With one reason being fear of losing work in case of disconnecting from the network.

Our main goal is therefore to create a library extending or modifying a proven synchronization method, to store necessary parts of state and publish them in a reasonable manner upon reconnection.

To achieve this we examine popular choices for synchronization. Then we take a close look at browser storage mechanisms. As well as finding and comparing existing solutions of offline synchronization while listing their drawbacks, limiting them from being widely used.

1.1 Illustrational example

To illustrate the problems that current applications with multi-user collaboration face you can think of this example. 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

Currently out of the 3 defined goals, we are going over the first 2 points. We explored the problem statement, different methods and how things are implemented. Their issues, drawbacks and pros in implementations.

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.

Nvrh rieenia, postupy, algoritmy, schmy, diagramy, tabuky, at, o je potrebn... Rozsah 2 3 strany.
Displayed equations are centered and set on a separate line.

$$x + y = z \tag{1}$$

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

Kd programu sa pe nasledovne:

```
program Inflation (Output)
save_to_local_storage()
save_internally()

  const   MaxYears = 10;
  var     Year: 0..MaxYears;
          Factor1, Factor2, Factor3: Real;
end.
```

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.

We described their inner workings and explained our choice of

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.

o sa spravilo, o sa plnuje, at.

Poakovanie. Ak si ho niekto zasli, treba tam uvies (napr. Kolega, ktor pomha s Javou, at).

References

1. Author, Karolina Gawron: Title, Should I Consider PWA? Remarkable Possibilities of Progressive Web Apps: Url, https://www.monterail.com/blog/should-i-consider-pwa-remarkable-possibilities-of-progressive-web-apps
2. Title, Progressive Web App Checklist: Url, https://developers.google.com/web/progressive-web-apps/checklist
3. Author, Chris Mills: Title, Progressive web app advantages: Url, https://developer.mozilla.org/en-US/docs/Web/Apps/Progressive/Advantages#Network_independent
4. Url, https://html5test.com/results/desktop.html
5. Title, Collaborative software: Url, https://en.wikipedia.org/wiki/Collaborative_software
6. Title, Operational transformation: Url, https://en.wikipedia.org/wiki/Operational_transformation
7. Author, Ellis C. A., GIbbs S. J.: Title, Concurrency Control in Groupware Systems, Journal: SIGMOD Rec., Issue Date: June 1989

8. Title, Conflict-free replicated data type: Url, https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type
9. Author, Amanda Silver: Title, Introducing Visual Studio Live Share: Url, https://code.visualstudio.com/blogs/2017/11/15/live-share
10. Title, When do network partitions occur in practice?: Url, https://www.quora.com/When-do-network-partitions-occur-in-practice
11. Author, Russell Sullivan: Title, CRDTs explained - supercharge your serverless with CRDTs at the Edge: Url, https://serverless.com/blog/crdt-explained-supercharge-serverless-at-edge/