Special seminar to bachelor thesis

Šimon Kocúrek

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

1.1 Problem statement

With time progressive web applications are becoming more and more popular. This movement aims to improve user experience, security and portability of applications. While allowing more complex applications running inside web browser environment.

With the rise of progressive web applications, many applications are starting to take advantage of the fact that all clients are connected to the server. One example of a system that requires this constant connection from all clients is a collaborative system.

Collaborative systems are parts of application, where clients can collectively operate on the same data. Each client can change the data and propagate the change to all other clients.

These changes can cause conflicts in case the actions two or more clients executed are mutually exclusive. Example of such mutually exclusive actions are editing and deleting same paragraph of text or adding different words on same position in text.

The problem we will tackle in this paper is a way to make collaborative systems work with offline mode. This will enable the client separated from the server in case of network partition to continue working and synchronize his state with the other clients when establishing connection to the server becomes possible again.

1.2 Solved problems

Detecting network partition: In order to activate the offline mode the event of network partition needs to be detected. Upon detection the offline mode will be activated and all failed request will be stored locally.

We can detect that partition occurred by either polling the server in specified time intervals. Other possible solution would be to catch failed request to the server. If the request due to a network related issue we know that the partition occurred.

As a part of partition detection problem, we need to check when the server becomes available again to synchronize local state with the server one. Unfortunately we have only one option here and that is pinging the server until we get a response.

When we are sure we have stable connection to the server we can submit all changes made since the partition occurred as well as download changes that were made by other clients in that time. After this finishes and we no longer have any changes stored locally, we can disable the offline mode.

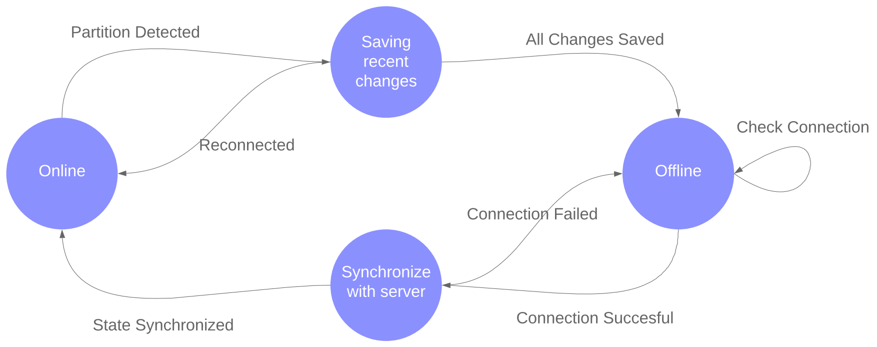

Fig. 1. State diagram of offline mode transitions.

Choosing synchronization method When implementing collaboration systems, we can use an already implemented library for synchronizing state across multiple clients. There are many options to choose from, with biggest differences coming from the synchronization method that was used.

These methods describe how the state should be stored, what messages are sent to the server and how to handle conflicts. As they are academically studied and proven to work, they can serve as an abstraction we will base our offline mode upon.

1. Conflict Free Replicated Data Types

   This is a synchronization method based on storing all data in a conflict free replicated data type[3]. This is a type with properties similar to set. Every operation performed on this data type has to be commutative, associative and idempotent.

   In practice such data types are not flexible enough, so a modified version of JSON format is used. This modification however isn't perfect. There is a need for custom garbage collector and a possibility of creating new data during conflicts.

2. Operational Transformation

   This synchronization method is among the oldest ones. With it's approach to synchronization being based on defining a set of operations that users can perform on the data. And transforming these operations so that after applying them, all clients arrive at the same state.

   Approach used by operational transformation is difficult to implement. This lead to multiple research papers being disputed and implementations abandoned. However this method is one of the most versatile as it can be modified to support undo actions, merging operations and is able to validate data constraints set by the server.

3. Differential Synchronization

   This method belongs to the less explored ones. Differential synchronization takes inspiration from version control systems and works by defining a `diff()` and `merge()` functions[1]. These functions are used to calculate, what changes user made to the data and to combine 2 different states.

   The main issue with this method lies in data redundancy. In order to perform `diff()` on states, it needs 2 additional copies of entire state on both the server and client. This makes the method very unreasonable choice for many online web applications.

Optimizing space and time complexity We will explore space and time complexities of various implementations of offline mode. The main focus will be put on space complexity as storage space of websites is very limited. Usually set by the user, the space can vary from 2.5MB of storage space to 10MB, or 50MB if database is used.

While time complexity is mostly based on synchronization method we choose. We can try to improve upon the initial complexities by merging changes made across time into a single bigger change. This could mean that after reconnecting, we could send all state to the server in a single request.

Resolving conflicts When working in an online mode, the changes in state are minimal as every change is sent to the server immediately. This is reflected in the conflict resolution algorithms. When state conflicts occur, they are resolved automatically, with no user action required.

In our case, we need to be able to detect, when conflicts are large enough, throwing away any changes can be harmful. In this case the default algorithm merging behavior will be overridden and user prompted to make a decision what changes should be kept.

2 Existing solutions

2.1 Google Docs

This is an example of a widely used collaboration system. With it's implementation based on Operational Transformation method Google Docs serve as an example that a robust collaborative system can be built using it.

The approach Google Docs chose for implementing offline mode is a bit lacking. The application relies on a browser extension to support offline mode. In case the extension is not installed a notification is shown and editing disabled.

In our case requiring an extension to enable usage of our offline mode is quite cumbersome. Especially when we can use browser storage mechanisms to store our data.

2.2 Meteor

A less known solution to synchronization in a collaboration system is a Meteor framework. The meteor by itself does not provide a synchronization functionality. What it does however provide is a web application solution with offline mode working out of the box.

In case the user loses connection to the server the clients stores all server requests locally, while hiding the fact that server is unavailable by performing optimistic updates.

This can be easily modified to work with any synchronization method by implementing synchronization push notifications from the server. The main downside of this however would be that we would be tied to a specific backend stack chosen by the framework, which is unwanted if we want a general solution.

2.3 Git

Version control systems are applications that require offline synchronization. The most popular example, known among developers is git. Git works in a way similar to Differential Synchronization by using diffs and merges.

However the problem we face, the git version control system doesn't have is a need for low data redundancy as it has direct access to file system, while browser storage limits us to something around 5 megabytes of data.

Because this system always works offline, the default conflict resolution strategy is to let user merge everything by himself and allow submitting to the server only once this is done.

3 Suggested solution

3.1 Choosing Synchronization method

In order to implement an offline mode in a collaboration system, we first need a working version of such system working when all clients are constantly online. For this purpose we need to choose the synchronization method most suitable for our requirements. We will write our library with offline mode on top of this method implementation.

Based on a comparison table, we chose Operational Transformation as a method most suitable for offline mode. Since it has very data redundancy, as well as very stable implementations to choose from.

Table 1. Synchronization method characteristics based on requirements

	Conflict Resolution	**Pros**	**Cons**
CRDT	Data type implementation	Simple implementation Doesn't require server	Not reliable conflict resolution Garbage collected
OT	Server-side `transform()` function	Guaranteed eventual consistency Supports constraint checking Undo operation support	Difficult implementation Operations are not idempotent
DS	Custom `diff()` and `merge()` methods	Control over conflict resolution Low network overhead	High data redundancy

Based on this choice we looked for implementations usable in the browser. Our solution will be built on top of Open Source javascript library ShareDB (`github.com/share/sharedb`), as it provides us with simple websocket based implementation of Operational Transformation.

3.2 Choosing storage mechanism

Before we start implementing the offline mode we need some storage mechanism, where all changes made during offline session will be kept. While keeping all data in memory would simplify the system as no additional API would be needed and no storage limits would be presented, it would introduce a problem where user would lose all his progress after his first page refresh.

As it is natural to refresh, when connection is lost, we need a solution that works even after user refreshes the website. When we limit ourselves to widely supported options, we are left with cookies, local storage and index db[2].

We ruled out cookies, as the data does not need to be sent to the server and back on each request. While index db has higher limit for stored data size, it requires structured data stored in tables, while operations we need to store are JSON objects. Based on this reasoning we chose local storage as a mechanism, that can survive both page refresh and session end, while storing performed operations.

3.3 Implementation

Our plan for implementation is to start with a simple frontend and backend demonstrating online and offline mode synchronization between clients. This implementation will be built upon libraries that offer simple solution for online synchronization.

We implement offline mode by intercepting each operation request and instead of sending, store it in local storage. As we store in a key-value store, we will need the operations stored in order they happened. As well as make sure our keys are not in conflict anything someone using our library might also use.

After we have working offline mode, we start implementing state transitions of online to offline mode and offline to online mode. Implementing this means we need to listen to connection changes `window.addEventListener('offline', function);` and `window.addEventListener('online', function);`. Then perform a needed backup, or synchronization based on state diagram defined before.

3.4 Comparisons

Once we have a working library with state transitions we will explore possible optimizations for and sending operations to minimize reconnection latency and space requirements.

References

1. N. Fraser. Differential synchronization. In *DocEng'09, Proceedings of the 2009 ACM Symposium on Document Engineering*, pages 13–20, 2 Penn Plaza, Suite 701, New York, New York 10121-0701, 2009.
2. S. Michael, M. Chris, G. Milind, p. Kevin, and S. Eric. Client-side storage. 2018.
3. M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In X. Défago, F. Petit, and V. Villain, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 386–400, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.