Seamless Synchronization for Collaborative Web Services

Abstract. Collaborative web services, which allow multiple people to work together on the same data, are becoming increasingly popular. However, current state-of-the-art frameworks for interactive client-side replication cannot handle network disruptions well, or suffer from large metadata overhead when clients are short-lived. This demonstration will show SystemX[1], a generic web middleware for data synchronization in browser-based applications and interactive groupware. It offers a fine-grained data synchronization model, using state-based Conflict-free Replicated Data Types, and leverages Merkle-trees in the data model for efficient synchronization. We provide an interactive demonstration of a public drawing application that workshop attendees can test and experiment with. We will also demonstrate the robustness in disconnected and offline settings.

Keywords: CRDTs · Online collaboration · Eventual Consistency.

1 Introduction and motivation

The use of online software services to collaborate remotely has been increasing in the last decade, especially in the last year due to the COVID-19 pandemic. Collaborative groupware applications, such as Google Docs or Microsoft Whiteboard, allow people to work together on the same document, without them being present in the same geographic location. People can work from anywhere they want, even in unstable network conditions, or while being offline. When a connection is available, changes should be replicated to all other client replicas within 1-2 seconds to keep the user experience interactive. Five seconds is the absolute maximum before users are becoming annoyed [9]. When offline, the user should be able to work further on the local copy of the data. Once the user comes back online, any changes should be replicated as fast as possible. This is especially important in unstable network conditions, where there is a limited time frame available to replicate all updates. The requirement for offline support implies the evolution to a more client-centric architecture, in which the different clients all become the authoritative data replicas [1]. This is in contrast to the classical client-server model, where the server is responsible for both data and business logic, typically organized in a data-tier and a business-tier. While this gives reasonable good performance when online, it comes at a cost of higher latency for clients located geographically far from the main server.

The most used client-server technology for collaborative groupware is Operational Transformation (OT) [3]. For example, Google Docs uses OT as synchronization technology. It uses a central server that transforms the conflicting

[1] Name of the system has been anonymized for double-blind reviewing.

operations for each replica to allow them to be applied in a different order on the other replicas. However, these transformations are rather complex and resource-intensive on the server, limiting the scalability of this technique. Moreover, OT only works for short-time disconnections and cannot be used when the client is offline for a longer time.

Several client-centric frameworks exist for collaborative web services. They rely on Conflict-free Replicated Data Types (CRDTs) [10] to automatically resolve any conflicts that would arise from multiple people editing the same data. There are several kinds of CRDTs. Operation-based CRDTs (CmRDTs) must still send all operations between the replicas using a reliable, exactly-once, message channel, similar to OT. However, no central component to transform these operations is required, as all operations are commutative. CmRDTs are used in Yjs [8] and Automerge [4,5]. State-based CRDTs (CvRDTs) do not use operations, but instead, they send the full state to other replicas, who will merge that state with their local state. CvRDTs are not suitable for client-centric interactive applications, as the full state is too expensive to send every time. It can however be used to replicate data between backend servers. Delta-state-based CRDTs [7] use vector clocks to calculate which part of the data needs to be sent to other replicas. They require much less of the message channel compared to operation-based CRDTs, however, the total size of the metadata will grow with every client that makes an edit. Especially in a web-based environment, where clients are often short-lived, the metadata will become larger over time, reducing the interactive performance. Delta-state-based CRDTs are used in Legion [6].

This demonstration will show SystemX [2], a generic web middleware for data synchronization in the context of web-based services and interactive groupware. SystemX leverages nested state-based CRDTs and Merkle-trees to efficiently replicate changes. Compared to state-of-the-art frameworks, SystemX offers:

- continuous and interactive synchronization of online web clients,
- prompt resynchronization of offline clients when they come back online,
- no meta-data explosion.

Application developers can leverage SystemX to create collaborative services that are resilient against network failures. SystemX offers a flexible data model, with fine-grained synchronization and automatic conflict resolution. Online web clients achieve interactive synchronization, making it possible for several people to work fluently on the same document. However, if no internet connection is available, such as in a tunnel or an airplane, clients can continue on their local copy. SystemX is especially robust against these offline periods, and is able to quickly replicate all missed updates, and achieve the same interactive performance as before within seconds. This robustness also makes SystemX interesting for the field-services industry, where technicians are often on the road going from customer to customer for technical interventions. A stable internet connection is not always available on their location, however, writing off all used materials is important for correct billing and inventory. Using SystemX, those offline reports will be synchronized quickly when an internet connection is available again, even when multiple technicians are working on the same job.

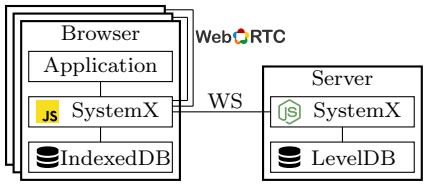

Fig. 1. Deployment architecture

```
<script src="systemx-browser.js"></script>
<script>
SystemX("ws://localhost:8080").then(
  async (systemx) => {
    await systemx.set("obj11.color", "#f00");
  }
)
</script>
```

Fig. 2. Public API example

2 Overview of the SystemX framework

SystemX is a JavaScript framework for application developers to synchronize data between browser-clients. SystemX provides Strong Eventual Consistency out-of-the-box, without letting the developer worry about it. Conflicts are solved automatically by the framework.

Data model. SystemX can be used to replicate JSON data structures containing strings, numbers, booleans, and objects; the latter can include any of those recursively. SystemX uses this tree-structure of the JSON data to create a Merkle-tree internally, which is used for efficient synchronization. State-based CRDTs are used to resolve conflicts under-the-hood. Application developers do not need to concern themselves with these internals. However, they need to be aware that data is only eventually consistent. Since we are using state-based CRDTs, there is little required from the message channel, compared to existing operation-based approaches. There is also no need to keep track of clients or client-specific metadata such as vector clocks.

Architecture and API. The deployment architecture of SystemX is depicted in Fig. 1. SystemX provides a JavaScript API for web applications to read and modify the tree-structured data. All data is stored locally in the browser using the IndexedDB key-value store, which is built-in in every modern browser. Data is replicated to a server running on NodeJS using a direct WebSocket connection. This WebSocket connection and the server are also used as a signaling channel to set up peer-to-peer WebRTC connections between the other browser instances. Once a WebRTC connection is initialized, the different SystemX replicas can replicate the changes directly with each other. This reduces the latency to replicate changes to other browser instances, and also improves the scalability, as the central server is no longer a bottleneck. Fig. 2 shows an example code snippet using the public API of SystemX. It connects to the WebSocket endpoint of the NodeJS server. Developers can then use the CRUD operations `get`, `set` and `del`, and the path in the tree, to retrieve and modify data. The full data is immediately stored locally, and in the background SystemX will replicate the changes to the server and to any other connected browser clients.

Internal synchronization protocol. Internally, the synchronization protocol always runs directly between two different replicas, either browser-to-browser or

browser-to-server. The protocol uses the Merkle-tree to find out which part of the tree needs to be sent to the other replica. If the local hash does not match the hash on the remote replica, then the corresponding state-based CRDT will be used to merge the remote state with the local state. We refer to Anonymous et al. [2] for a detailed specification of this CRDT merge operation.

3 Interactive Demonstration

We demonstrate SystemX with an interactive web-based drawing application for all the demo attendees world-wide. The web application provides a drawing that can be edited by multiple users simultaneously, and any conflicts that might arise will be solved automatically by the underlying CRDTs. The attached video shows several browser instances running on geographically distributed VMs via remote desktop. This demonstrates the interactive latency with worldwide collaboration when everyone is online. SystemX is especially robust against network failures, and we demonstrate this with two scenario's. First, the server is stopped, yet, all browser clients can continue to work together by using the peer-to-peer network between them. Second, one browser instance loses its internet connection temporarily. We also demonstrate that both the online clients, as well as the offline client, can continue to work on their copies of the data. When the internet connection is restored, we demonstrate that the changes are merged quickly, and interactive performance is resumed. Since the demo application is public, participants of the conference can also use their own laptop or mobile device to make changes to the drawing interactively.

References

1. Anonymous: Title omitted for double-blind reviewing. In: EdgeSys '19 (2019)
2. Anonymous: Title omitted for double-blind reviewing. IEEE Transactions on Parallel and Distributed Systems (2021)
3. Ellis, C.A., Gibbs, S.J.: Concurrency control in groupware systems. SIGMOD Rec. (1989)
4. Kleppmann, M., Beresford, A.R.: A conflict-free replicated json datatype. IEEE Transactions on Parallel and Distributed Systems (2017)
5. Kleppmann, M., Beresford, A.R.: Automerge: Real-time data sync between edge devices. In: MobiUK'18 (2018)
6. van der Linde, A., Fouto, P., Leitão, J.a., Preguiça, N., Castiñeira, S., Bieniusa, A.: Legion: Enriching internet services with peer-to-peer interactions. In: WWW '17 (2017)
7. van der Linde, A., Leitão, J.a., Preguiça, N.: Δ-crdts: Making δ-crdts delta-based. In: PaPoC '16 (2016)
8. Nicolaescu, P., Jahns, K., Derntl, M., Klamma, R.: Near real-time peer-to-peer shared editing on extensible data types. In: GROUP '16 (2016)
9. Nielsen, J.: Usability Engineering. Nielsen Norman Group (1993)
10. Shapiro, M., Perguiça, N., Baquero, C., Zawirski, M.: Conflict-free replicated data types. In: SSS 2011 (2011)

A Requirements for the demo

The online web application will be deployed at a public URL. Preferably there is a stable WiFi connection available in the room to allow participants to interactively use this application simultaneously.

The demonstration video is available at: `https://youtu.be/aI4YcSjuEzA`.