Computer Science Tripos – Part II – Project Proposal

Conflict Free Document Editing with Different Technologies

J. Send, Trinity Hall

Originator: J. Send

10 October 2016

Project Supervisor: S. Kollmann

Director of Studies: Prof. S. Moore

Project Overseers: Prof. T. Griffin & Prof. P. Lio

Introduction

Background

In a world of ever increasing connectivity, collaborative features of applications will take on greater and greater roles. Popular services such as Google Docs offer real-time editing of documents by multiple users, a type of interaction that will move from being a special offering by few applications to a common and expected interface.

The key property that must be implemented to achieve concurrent editing is eventual consistency, meaning that all connected users should end up with the same result after receiving all changes to the document — even if edits conflict [1]. There are two main technologies that are used to enable concurrent editing of a document (plain text or otherwise). One approach is Operational Transforms (OT), which that generally relies on having a central server receive, serialize, transform, and relay edits occurring simultaneously to each client. OT is notoriously difficult to implement correctly as incoming operations have to be transformed against preceding ones on each client, such that the result converges [5]. The server may also be required to make some transformations. Due to this, central server must be able to read all the operations being performed by clients. Thus, unless the server is trusted and secure, OT-based services cannot provide any security or privacy guarantees.

The alternative, newer technology uses Conflict Free Replicated Datatypes (CRDTs). Instead of resolving conflicts and guaranteeing eventual consistency by transforming operations against each other, CRDTs use special datastructures that guarantee that no operations will conflict [3]. There are many types of CRDTs that are tailored for different situations. One example is a simple up-down counter which could be implemented as two locally replicated registers, one for increments and one for decrements, where the current state is their difference[4]. Compared to OT, there is no interdependence between edits

(as long as the network protocol can guarantee in-order delivery), which means CRDT-based systems can do away with the server and be implemented using peer to peer (P2P) protocols. This lends itself to security (encryption is now possible between endpoints), and possibly more scalability and efficiency.

This project is first concerned with exploring and developing a P2P CRDT concurrent text editor, and secondly comparing it to the OT-based client/server approach available in the open source library ShareJS. Several extensions are also possible, listed in later sections.

Resources required

The primary external resource I will need is the Javascript library ShareJS, which is published under the MIT license on GitHub [2].

Additionally, I am developing on my personal computer, a Thinkpad T440s with 8 GiB of RAM, 128 GB of hard drive space, and a low wattage dual core Intel CPU running at 1.60GHz. The primary development environment is Ubuntu Linux, though Windows 10 is also available on the same machine.

Git with Github is used as both a version control system and a cloud backup. Dropbox provides continuous cloud backups as well. Secondary development machines are any of the MCS computers.

Starting point

I have some knowledge of the open source library ShareJS from a past internship, which I aim to leverage when evaluating and comparing it to my system. My knowledge of CRDTs and the relevant adding/insert/merge algorithms stems mostly from a high level explanation provided by Martin Kleppmann, along with a diagram. This will be the starting point for my from-scratch implementation of the concurrent text editor.

Since I have no experience writing test cases and performance profiling, nor network simulation, I will have to learn how to do these.

Lastly, I may consult various papers on CRDTs, as well as my supervisor's work in the area, if required.

Work to be done

Overview

I plan to implement a simulation of P2P CRDT text editing using Typescript. Following this, my project will focus on comparing an existing OT-based concurrent document editing library (ShareJS) to my implementation, in order to draw conclusions about their relative network and memory efficiency, and scalability. It is highly likely that my system will need some optimization, which can feed back into my evaluation and comparison

process. In the case that these phases do not take too long, there are several possible extensions. The first would be to add a networking layer to the simulation - in effect turning the it into a usable library. The second would be researching and implementing 'undo' and 'move' operations, which are relatively open research problems.

Detailed Project Structure

1. **Core CRDT Development:** Consider and decide CRDT datastructures. Then detail how I expect the insert/delete/merge algorithms to work on paper, followed by implementing these. Lastly, I need to learn frameworks for testing my implementation. The tests for correctness should include hand-crafted unit tests to confirm expected behavior of intermediate execution steps and convergence of results across clients, along with generated test loads to check correct convergence on all clients.

2. **Implement Simulation:** Model having an arbitrary number of clients each running the CRDT algorithms, and simulate networking between these clients. Because this is P2P, it may be worth adding functionality for a variety of network topologies.

3. **Set up ShareJS and Compare:** Set up the ShareJS environment, mirror functionality and setups between two systems as much as possible, and create corresponding performance profiling tests for both systems. These will focus on network efficiency, memory usage, and scalability.

4. **Tune Implementation:** There will likely be opportunity for some optimization, which will feed back into the performance comparisons in the previous step and help evaluate the optimizations themselves.

5. **Extensions:** The first extension is implementing a proper P2P network stack and remove the simulated networking. Next would be researching undo and move operations and perhaps try to implement one or both of these.

Possible extensions

There are two extensions of varying difficulty:

- (Easier) Replace networking simulation with a P2P networking library. The end result of this extension should be a ready to deploy Typescript (compiled to Javascript) library.

- (Difficult) Research prior work on undo and move functionality using CRDTs. If something suitable is found, implement it. Otherwise, attempt to work toward my own solution.

Success criteria

These are the main success criteria associated with my project

1. A concurrent text editor based on CRDTs has been implemented.

2. The concurrent text editor passes all correctness tests.

3. Quantitative results comparing ShareJS and the CRDT based system have been obtained and analyzed.

Timetable

Planned starting date is 16/10/2011.

1. **Michaelmas weeks 2–3** Develop CRDT datastructure and algorithms on paper. Read into P2P networks and simulating them.

2. **Michaelmas weeks 4–5** Lay out project files and implement network simulation with support for different P2P topologies.

3. **Michaelmas weeks 6–8** Implement CRDT datastructures and algorithms, and connect these to network simulation.

4. **Michaelmas vacation** Learn an appropriate testing framework, write and generate unit tests for correctness of implementation. Fix any bugs discovered by the testing process. Set up ShareJS environment. Begin outlining progress report.

5. **Lent weeks 0–1** Complete progress report. Mirror functionality of ShareJS to the setup of my system. Start writing performance benchmarks and scalability tests for both systems.

6. **Lent weeks 2–4** Execute tests and analyze results. Try to explain differences and similarities observed. Tune my implementation and evaluate various optimizations. Begin writing dissertation.

7. **Lent weeks 5–6** Continue writing dissertation and optimizing system. Begin research for extension which implements proper networking stack.

8. **Lent weeks 7–8** Continue writing dissertation. Before terms ends, review and peer-review (including supervisor) incomplete draft. Implement networking extension. Research undo and move operations with CRDTs.

9. **Easter vacation:** Finish dissertation draft. Work on undo and move extensions for system.

10. **Easter term 0–2:** Edit and proof read dissertation. Work on extensions.

11. **Easter term 3:** Proof read and then submit early to concentrate on exams.