Part II Project Proposal

Alex Dalgleish amd96 Robinson

October 25, 2016

Introduction and Description of the Work

A common practice is to have a text document that requires editing by multiple people. For example, to schedule a meeting a manager may give their subordinates a list of possible times and require them to indicate which of those they would be able to attend. However, the common applications for this purpose involve editing a document through a central server such that the server has a record of all edits made to the document. To those very privacy conscious, this may not be desirable. From the Google Drive terms of service:[1]

> When you upload, submit, store, send or receive content to or through our Services, you give Google (and those we work with) a worldwide license to use, host, store, reproduce, modify, create derivative works (such as those resulting from translations, adaptations or other changes we make so that your content works better with our Services), communicate, publish, publicly perform, publicly display and distribute such content.

In distributed systems, only two of consistency, availability and partition-tolerance are available.[2] Often, availability and parition-tolerance are prioritised, sacrificing strong consistency, as service providers may prefer a service to be fault-tolerant and always accessible. Therefore, given that strong consistency is provably impossible, other weaker forms of consistency need to be maximised as much as possible.

Operation-based Conflict-free Replicated Data Types (CRDTs) are data structures designed for systems needing strong eventual consistency. That is, two nodes who receive the same set of CRDTs will be guaranteed to be in the same state as soon as the final one arrives at each.[3] The idea is that the data structure holds the update to the state (and not the state itself) and this is broadcast to all listening nodes (eg add 10 to x). A merge operation is then performed with the node's current state and the incoming CRDT state update. These operations must necessarily be commutative so that two nodes receiving two state updates in a different order can achieve the same state. In addition, there must

[1] Google Terms of Service - Privacy & Terms: https://www.google.com/intl/en/policies/terms/

[2] Seth Gilbert and Nancy Lynch, Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, ACM SIGACT News, Volume 33 Issue 2 (2002), pg. 51-59.

[3] Proposition 2.2, Marc Shapiro, Nuno Preguica, Carlos Baquero, Marek Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. [Research Report] RR-7506, Inria Centre Paris-Rocquencourt; INRIA. 2011, pp.50. <inria-00555588>

be some middleware to guarantee exactly-once message delivery. As these state updates are not idempotent by nature, message replication will result in erroneous state. For example, if one node broadcasts add 10 to x and another receives it twice (because the network duplicated the message) then the receiver will have added 20 to x and the two nodes won't converge to a consistent state.

I plan to develop a library for a collaborative text editor application that uses operation-based CRDTs to achieve strong eventual consistency and does not send data to a central server, such that only members asked to edit a document can know its contents.

Resources required

I will be using my own machine: 4x2GHz CPU, 8Gb RAM, 300Gb Disk, Ubuntu 16.04LTS I will keep everything in a git repository on GitHub. Weekly backups will also be done to an external hard drive. I will also need support for Tor on my machine.

Starting Point

- **Computer Science Tripos**
  My project will draw on knowledge gained from parts IA and IB of the Tripos. In particular, Concurrent and Distributed Systems provides theory background for CRDTs and Networking for TCP/IP and Peer-to-peer architectures. This is supplemented by Topics in Concurrency and Principles of Communications.

- **Code by Martin Kleppmann**
  A server for connecting clients over websockets and sending data between them (https://github.com/trvedata)

Substance and Structure of the Project

I plan to incrementally build a library for a peer-to-peer (P2P), collaborative, text editor. It will allow the concurrent editing of a single string of text by 2 clients over a network.

Initially, I will evaluate different possible platforms for my application to be built on e.g. web, desktop and Android Then, I will make sure I am familiar with a suitable language/framework for that platform.

I will write first a library for editing a single string of text (an ordered list of characters)

using operation-based CRDTs (Conflict-free Replicated Data Types). This will involve coming up with a data structure to represent the string, defining allowed operations for strings, and specifying how to perform those operations on a string.

Next, I will use this to write code that can send and receive CRDTs over a network. Martin Kleppmann has written a server application that can connect clients like these and enable them to communicate, so I can run my implementation with this.

Then, I will create a service that simply allows discovery of other clients editing the same document as you, and make clients communicate Peer-to-peer. This will involve knowing through some other channel a common identifier for a document, and a service which can link this identifier to others who know it.

Extensions:

a) Make a simple application that uses this library - this will have a very minimal UI, but will make it easier to use and test the library I have built

b) Support >2 users & allow coming and going at any time

c) Use the Tor network for all communication - this will involve sending data through a client-side proxy server, and using/creating Tor hidden services.

d) Make a more substantial Graphical User Interface.

In order to evaluate my project:

- I plan to measure different communication metrics such as latency and bandwidth usage for my application, and potentially how these scale with numbers of peers.
- I will compare these with the different versions of my application, as well as against the Google Realtime API, which provides concurrent editing functionality for collaborative applications.
- I will write unit tests for the CRDT library to test its correctness in specific cases.
- If the Tor extension is implemented, I can also inspect packets coming into one client and observe if there is any data identifying the host from which it came.

Success Criteria

By the end of the project I aim:

- To have a tested library for operation-based CRDTs which represent an ordered list of characters
- To have 2 versions of a library which a text editor application might use to collaboratively edit a document with other similar applications over a network. One which will use a central server to pass data, and another where data is sent directly between clients.
- To have a library which, when used by an application, provides collaborative editing functionality with sufficiently small latency between clients to be considered 'realtime'

Plan of work

17/10 - 31/10

- Evaluate the available platforms/languages/frameworks. At the moment these are Android, web/JS and Python and choose one to work in
- Research CRDTs, Peer-to-peer architectures and existing applications, and Tor
- Setup development environment (includes testing frameworks and build tools)
- Start writing CRDT library

31/10 - 14/11

- Finish CRDT library
- Start writing client-server version

GOAL : A working and tested CRDT library

14/10 - 28/11

- Finish client-server version
- Start Peer-to-peer version

28/11 - 5/12 + vacation

- Finish Peer-to-peer version
- Start on extensions

GOAL : 2 versions of library

23/1 - 3/2

- Write progress report
- Continue with extensions

GOAL : Submitted progress report

3/2 - 20/2

- Finish writing current extension
- Start collecting evaluation data

GOAL : Implementation code completed

20/2 - 6/3

- Collect evaluation data and interpret/display results

GOAL : A set of data displaying the project's success

6/3 - 15/3 + vacation

- Write first draft of dissertation

24/4 - 8/5

- Second draft of dissertation

8/5 - 19/5

- Final draft of dissertation