Joshua Send

Conflict Free Document Editing with Different Technologies

Computer Science Tripos – Part II

Trinity Hall

20th March 2017

Proforma

Name:	**Joshua Send**
College:	**Trinity Hall**
Project Title:	**Conflict Free Document Editing with Different Technol**
Examination:	**Computer Science Tripos – Part II, June 2017**
Word Count:	**1587**[1]
Project Originator:	Joshua Send
Supervisor:	Stephan Kollmann

Original Aims of the Project

TODO[2]

Work Completed

TODO

Special Difficulties

TODO

[1] This word count was computed by `detex diss.tex | tr -cd '0-9A-Za-z \n' | wc -w`

[2] A normal footnote without the complication of being in a table.

Declaration

I, Joshua Send of Trinity Hall, being a candidate for Part II of the Computer Science Tripos [or the Diploma in Computer Science], hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed TODO [signature]

Date TODO [date]

Contents

List of Figures

Acknowledgements

TODO

Chapter 1

Introduction

Real time interaction between users is becoming an increasingly important feature to many applications, from word processing to CAD to social networking. This dissertation examines trade offs that should be considered when applying the prevailing technologies that enable concurrent use of data in applications. More specifically, this project implements and analyzes a concurrent text editor based on Convergent Replicated Data Types (also known as Conflict-free Replicated Data Types), CRDT in short, in comparison to an editor exploiting Operational Transformations (OT) as its core technology.

1.1 Motivation

Realtime collaborative editing was first motivated by a demonstration in the Mother of All Demos by Douglas Engelbart in 1968 [**MotherDemo**]. From that time, it took several decades for implementations of such editing systems to appear. Early products were released in the 1990's, and the 1989 paper by Gibbs and Ellis [**Ellis1989**] marked the beginning of extended research into operational transformations. Due to almost 20 years of research, OT is a relatively developed field and has been applied to products that are commonly used. The most familiar of these is likely to be Google Docs, which seems to behave in a predictable and well understood way. One reason Google Docs is so widely used might be that it follows users' expectations for how a concurrent, multi-user document editor should work. Importantly, this includes lock-free editing and independence on a fast connection, no loss of data, and the guarantee that everyone ends up with the same document when changes are complete. These are in fact the goals around which OT and CRDTs have developed.

The convergence, or consistency, property above is the hardest to provide it is easy to create a system where the last writer wins, but data is lost in the process. In a distributed system such as a shared text editor, the CAP theorem tells us we cannot guarantee all three of consistency, availability, and partition-tolerance [**Gilbert2005**]. However, if we forgo strong consistency guarantees and settle for eventual consistency, we are able provide all three [http://gun.js.org/distributed/matters.html]. As we will see,

achieving eventual consistency is non-trivial. The two prevailing approaches, operational transformations and commutative replicated data structures are discussed in detail the Preparation section.

1.2 Overview

This project aims to examine the trade-offs made when implementing highly distributed and concurrent document editing with Operational Transformations (OT) versus with Convergent Replicated Data Types (CRDTs). To do this I have designed experiments which expose statistics about network and processor usage, memory consumption, and scalability, and run these experiments on an environment built around the open source library ShareJS (which implements OT) along with a comparative system I created based on a specific CRDT. The system meets the originally proposed goals of implementing a concurrent text editor based on CRDTs which passes all tests for correctness; quantitative analysis is presented in the Evaluation section [section ref].

The custom CRDT on which the collaborative text editor is based is described in detail in the Implementation [section ref] section. In contrast to the OT-based library ShareJS, my system also runs on a peer to peer network architecture instead of a traditional client-server model. The lack of a server reduces the number of stateful parts in the overall system, at the expense of more complex networking. I managed this complexity by using a simulated peer to peer architecture. The simulation allows me to control the precise topology, link latencies, and protocol and explore advantages and disadvantages of using a P2P approach.

One extension, adding undo functionality to the CRDT, was also completed. Two different approaches are presented, both of which were conceived before reading related literature. One is similar to the approach taken in Logoot-Undo, which is discussed below. As far as I am aware, in the context of CRDTs for text editing, both are novel.

1.3 Related Work

Part of the challenge of this project was to develop my CRDT and associated algorithms based only on an explanation of the required functionality provided by Martin Kleppmann. As a result, my solution is not optimal in all aspects, and could be improved upon in the future. It also falls into the class of 'tombstone' CRDTs, which mark elements as deleted rather than fully removing them, which forces the data structures to grow continuously over time. Other CRDTs are 'tombstone-less' and do not suffer from this inefficiency. Existing CRDTs of both types are discussed here.

1.3.1 Treedoc

Treedoc [reference needed] is a replicated text buffer; an ordered set that supports insert-at and delete operations. This CRDT gets its name from the tree structure used to encode identifiers and order elements in the set. Each node in the tree contains one character, and the string contained in the buffer is retried using infix traversal. Each client has a copy of the same tree, and can insert new nodes at any time. Two concurrent inserts at the same node are merged as two 'mini-nodes' within one tree node. Each insert is tagged with a unique client identifier which comes from an ordered space. Using the identifier order in combination with infix traversal creates a total ordering over the characters contained in the tree. With the total order, all clients with copies of the tree will retrieve the same string from their Treedoc. Having a total order is an important property used to guarantee eventual consistency in CRDTs.

[perhaps insert Figure of large nodes/mininodes from the paper]

Deletes in this Treedoc are handled by marking a node as deleted (but the node remains in the structure). Thus Treedoc falls into the class of 'tombstone' CRDTs. As deletes and inserts are not guaranteed to result in a balanced tree, the authors propose an expensive commitment protocol to rebalance it. Not only is this inefficient, but also rather contrary to the spirit of CRDTs. However, trees do present an elegant solution to the problem of dense identifier spaces: we must be able to insert a new identifier between any two given identifiers at any point in time.

1.3.2 Logoot

Logoot [citation needed] belongs to the class of text CRDTs which do not require tombstones for deletion. It achieves this by totally ordering identifiers, rather than relying on implicit causal dependencies between identifiers (which Treedoc embeds in the tree's branches). Logoot does generate identifiers using a tree, but each identifier contains the full path in the tree, which frees it of dependence on other nodes. This means that to delete, any client can simply remove the identifier and the data it tags.

Logoot also favors marking larger blocks of text with identifiers, rather than per-character. This, in combination with not needing tombstones, promises major efficiency gains over CRDTs such as Treedoc. Even further, two papers [citation needed] [citation needed] offer optimizations on top of the basic Logoot implementation by improving the strategy used to allocate new identifiers in the generator tree. However, these algorithms are specific to Logoot and of little relevance to this project.

Logoot is important as an example of a tombstone-free CRDT for text. Additionally, subsequent research enabled 'undo' and 'redo' functionality for this CRDT, which is described below.

1.3.3 Logoot-Undo

CRDTs generally struggle to provide an undo mechanism since the concept of reversing an update to the data structure is fundamentally contrary to the key property of CRDTs: commutativity of operations. For example, reversing an insert is not commutative with the original insertion. If it were, the removal of a nonexistent element, followed by its insertion would have to result in the same thing as insertion followed by removal. In the first case, the element is present, while in the second it is removed. These outcomes clearly are not the same.

Logoot Undo [citation needed] proposes to resolve this by essentially tagging each identifier with a 'visibile' counter. An undo of an insertion would decrement it, while redo would increment it. If the 'visible' counter is positive, the characters are visible. As discussed in [REFERENCE NEEDED], this leads to some rather unexpected behavior. However, this approach is viable since increments and decrements commute and guarantee eventual convergence.

Chapter 2

Preparation

2.1 Consistency Models

2.1.1 What is "Conflict Free"

One important question to answer is, what is the exact definition of conflict free. There appears to be more than one way of interpreting it. On one hand, there is the user's intuitive idea that any of their own operations should behave as if they were the only users on the system. On the other hand, there is the data-centric view of conflict. In this case, operations conflict if they are concurrent and modify the same data or index in a text buffer. Conflict free then means that no data is lost, and after all operations are exchanged the resulting data is converged to the same state.

The common conflicting operations in text editing are inserting characters into the same index of a shared text buffer, or simultaneously deleting the same characters. The second is easy to make conflict-free, and both the user and data oriented definitions of conflict agree – deleting a character concurrently or on a single user system should still result in the character disappearing. In the case of inserting text into the same index, the definitions cannot agree. Both users expect their own text to appear in the index they inserted at. However, in order to satisfy the data-centric definition we are not allowed to lose data, and must eventually present both users with the same result. The solution is to let one user 'win' and insert their characters at the desired index, and shift the other users' characters to appear after. Both operational transformations and CRDTs achieve this in fundamentally different ways.

[create figure]

2.1.2 CCI Consistency Model

The commonly used consistency model for concurrent document editing is the CCI model. The definition here is borrowed from [citation needed].

- **Consistency:** All operations ordered by a precedence relation, such as Lamports happened-before relation [citation needed], are executed in the same order on every replica.
- **Convergence:** The system converges if all replicas are identical when the system is idle.
- **Intention Preservation:** The expected effect of an operation should be observed on all replicas. This is commonly accepted to mean:
  - *delete* A deleted line must not appear in the document unless the deletion is undone.
  - *insert* A line inserted on a peer must appear on every peer; the order relation between the document lines and a newly inserted line must be preserved on every peer.
  - *undo* Undoing a modification makes the system return to the state it would have reached if this modification was never produced.

Much of the work in OT and CRDTs goes into ensuring convergence. The given definition of intention preservation is accepted, but the given may produce some unexpected results as we will see when discussing Undo in [reference needed].

2.2 Achieving Eventual Consistency

As mentioned briefly in the prior section, operational transformations and CRDTs aim to achieve eventual convergence on all clients. The common conflicting operations that must be given special consideration are concurrently inserting characters at the same index, and deleting the same character, and deleting a character while moving its position. Discussion of the more complicated undo operation can be found in the Implementation [section referece] section.

2.2.1 Operational Transformations

The easiest way to understand how operational transformations work is by example. The following three figures discuss each of the scenarios in turn.

2.2.2 Convergent Replicated Data Types

2.2.3 ShareJS

ShareJS [citation needed] is an open source Javascript library implementing Operational Transformations which can be deployed on web browsers or NodeJS [footnote citation]

clients. It is the core resource around which I built the comparative system to collect statistics from. To this end, it is useful to know more precisely how ShareJS operates and what kind of behavior might be expected. As are a large variety of algorithms that can enable OT [citation needed - Analysis of Operational Transformations Algorithms], rather than tracking down the papers ShareJS is based on, much of what is summarized below was deduced by reading its source code.

ShareJS documents are versioned, and each operation applies to a specific version. The version number is used to transform operations against each other and detect concurrent changes. The supported operations are insert and delete, and the resulting modifications are sent as JSON to the server.

An Insert operation for adding text at index 100 in document version 1:

```
{v:1, op:[{i:'Hello World', p:100}]}
```

A delete operation

```
{v:1, op:[{d:'Hello', p:100}]}
```

Multiple operations may be sent in one packet

```
{v:1, op:[{d:'World', p:100}, {i:'Cambridge', p:110}]}
```

ShareJS uses a server that provides a global, serialized order of operations to be applied on each client. This means the order of all non-local operations will be the same on each client. The server also transforms concurrent operations against each other, but has the choice of rejecting an operation if the target document version is too old.

One important fact is that ShareJS only ever allows one packet to be in flight from a particular client to a server. Each packet can

2.3 Analysis

2.3.1 Memory

2.3.2 Network

2.3.3 Processor Load

2.3.4 Client-Server versus Peer to Peer

2.4 Starting Point

2.5 Requirements Analysis

2.6 Software Engineering

2.6.1 Libraries

2.6.2 Languages

2.6.3 Tooling

2.6.4 Backup Strategy

2.7 Early Design Decisions

Chapter 3

Implementation

This section will...

3.1 CRDT-based system

3.1.1 Overview

3.1.2 Custom CRDT

3.1.3 Network simulation

3.2 ShareJS Comparative Environment

3.2.1 Overview

3.3 Experiments and Automated Log Analysis

3.3.1 Separation of Concerns

3.3.2 Experiment Design

3.3.3 Log Analysis

Chapter 4

Evaluation

Chapter 5

Conclusion

Appendix A

Latex source

A.1 diss.tex

```
\documentclass[12pt,a4paper,twoside,openright]{report}


\usepackage[pdfborder={0 0 0}]{hyperref}    % turns references into hyperlinks
\usepackage[margin=25mm]{geometry}  % adjusts page layout
\usepackage{graphicx}  % allows inclusion of PDF, PNG and JPG images
\usepackage{verbatim}
\usepackage{docmute}   % only needed to allow inclusion of proposal.tex
\usepackage{url}
\usepackage[parfill]{parskip}

\usepackage{fancyvrb,newverbs,xcolor}


\usepackage[UKenglish]{babel}% Recommended
\usepackage[bibstyle=numeric,citestyle=numeric,backend=biber,natbib=true]{biblatex}

\addbibresource{refs.bib}% Syntax for version >= 1.2


\raggedbottom                           % try to avoid widows and orphans
\sloppy
\clubpenalty1000%
\widowpenalty1000%

\renewcommand{\baselinestretch}{1.1}    % adjust line spacing to make
                                        % more readable



\definecolor{cverbbg}{gray}{0.93}
\newenvironment{cverbatim}
 {\SaveVerbatim{cverb}}
 {\endSaveVerbatim
  \flushleft\fboxrule=0pt\fboxsep=.5em
  \colorbox{cverbbg}{\BUseVerbatim{cverb}}%
  \endflushleft
}
\newenvironment{lcverbatim}
 {\SaveVerbatim{cverb}}
 {\endSaveVerbatim
  \flushleft\fboxrule=0pt\fboxsep=.5em
  \colorbox{cverbbg}{%
```

```
    \makebox[\dimexpr\linewidth-2\fboxsep][l]{\BUseVerbatim{cverb}}%
  }
  \endflushleft
}
\newcommand{\ctexttt}[1]{\colorbox{cverbbg}{\texttt{#1}}}
\newverbcommand{\cverb}
  {\setbox\verbbox\hbox\bgroup}
  {\egroup\colorbox{cverbbg}{\box\verbbox}}


\begin{document}




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Title


\pagestyle{empty}

\rightline{\LARGE \textbf{Joshua Send}}

\vspace*{60mm}
\begin{center}
\Huge
\textbf{Conflict Free Document Editing with Different Technologies} \\[5mm]
Computer Science Tripos -- Part II \\[5mm]
Trinity Hall \\[5mm]
\today  % today's date
\end{center}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Proforma, table of contents and list of figures

\pagestyle{plain}

\chapter*{Proforma}

{\large
\begin{tabular}{ll}
Name:               & \bf Joshua Send                       \\
College:            & \bf Trinity Hall                      \\
Project Title:      & \bf Conflict Free Document Editing with Different Technologies \\
Examination:        & \bf Computer Science Tripos -- Part II, June 2017  \\
Word Count:         & \bf 1587\footnotemark[1] \\
Project Originator: & Joshua Send                    \\
Supervisor:         & Stephan Kollmann                     \\
\end{tabular}
}
\footnotetext[1]{This word count was computed
by \texttt{detex diss.tex | tr -cd '0-9A-Za-z $\tt\backslash$n' | wc -w}
}
\stepcounter{footnote}


\section*{Original Aims of the Project}


TODO\footnote{A normal footnote without the
complication of being in a table.}


\section*{Work Completed}

TODO
```

```
\section*{Special Difficulties}

TODO

\newpage
\section*{Declaration}

I, Joshua Send of Trinity Hall, being a candidate for Part II of the Computer
Science Tripos [or the Diploma in Computer Science], hereby declare
that this dissertation and the work described in it are my own work,
unaided except as may be specified below, and that the dissertation
does not contain material that has already been used to any substantial
extent for a comparable purpose.

\bigskip
\leftline{Signed TODO [signature]}

\medskip
\leftline{Date TODO [date]}

\tableofcontents

\listoffigures

\newpage
\section*{Acknowledgements}

TODO

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% now for the chapters

\pagestyle{headings}

\chapter{Introduction}

Real time interaction between users is becoming an increasingly important feature to many applications, from word processing


\section{Motivation}

Realtime collaborative editing was first motivated by a demonstration in the Mother of All Demos by Douglas Engelbart in 196

The convergence, or consistency, property above is the hardest to provide  it is easy to create a system where the last writ


\section{Overview}
This project aims to examine the trade-offs made when implementing highly distributed and concurrent document editing with O

The custom CRDT on which the collaborative text editor is based is described in detail in the Implementation [section ref] s

One extension, adding undo functionality to the CRDT, was also completed. Two different approaches are presented, both of wh

\section{Related Work}

Part of the challenge of this project was to develop my CRDT and associated algorithms based only on an explanation of the r

\subsection{Treedoc}

Treedoc [reference needed] is a replicated text buffer; an ordered set that supports insert-at and delete operations. This C

[perhaps insert Figure of large nodes/mininodes from the paper]

Deletes in this Treedoc are handled by marking a node as deleted (but the node remains in the structure). Thus Treedoc falls

\subsection{Logoot}
```

```
Logoot [citation needed] belongs to the class of text CRDTs which do not require tombstones for deletion. It achieves this b

Logoot also favors marking larger blocks of text with identifiers, rather than per-character. This, in combination with not

Logoot is important as an example of a tombstone-free CRDT for text. Additionally, subsequent research enabled 'undo' and 'r


\subsection{Logoot-Undo}

CRDTs generally struggle to provide an undo mechanism since the concept of reversing an update to the data structure is fund

Logoot Undo [citation needed] proposes to resolve this by essentially tagging each identifier with a 'visibile' counter. An


\chapter{Preparation}


\section{Consistency Models}

\subsection{What is "Conflict Free"}

One important question to answer is, what is the exact definition of conflict free. There appears to be more than one way of

The common conflicting operations in text editing are inserting characters into the same index of a shared text buffer, or s

[create figure]

\subsection{CCI Consistency Model}
The commonly used consistency model for concurrent document editing is the CCI model. The definition here is borrowed from

\begin{itemize}
\item \textbf{Consistency:} All operations ordered by a precedence relation, such as Lamports happened-before relation [cita

\item \textbf{Convergence:} The system converges if all replicas are identical when the system is idle.

\item \textbf{Intention Preservation:} The expected effect of an operation should be observed on all replicas. This is commo

\begin{itemize}
\item \textit{delete}  A deleted line must not appear in the document unless the deletion is undone.

\item \textit{insert}  A line inserted on a peer must appear on every peer; the order relation between the document lines an

\item  \textit{undo}  Undoing a modification makes the system return to the state it would have reached if this modification

\end{itemize}

\end{itemize}

Much of the work in OT and CRDTs goes into ensuring convergence. The given definition of intention preservation is accepted,


\section{Achieving Eventual Consistency}

As mentioned briefly in the prior section, operational transformations and CRDTs aim to achieve eventual convergence on all

\subsection{Operational Transformations}

The easiest way to understand how operational transformations work is by example. The following three figures discuss each o


\subsection{Convergent Replicated Data Types}


\subsection{ShareJS}
```

```
ShareJS [citation needed] is an open source Javascript library implementing Operational Transformations which can be deploye

ShareJS documents are versioned, and each operation applies to a specific version. The version number is used to transform o

An Insert operation for adding text at index 100 in document version 1:
\begin{lcverbatim}
{v:1, op:[{i:'Hello World', p:100}]}
\end{lcverbatim}

A delete operation
\begin{lcverbatim}
{v:1, op:[{d:'Hello', p:100}]}
\end{lcverbatim}

Multiple operations may be sent in one packet
\begin{lcverbatim}
{v:1, op:[{d:'World', p:100}, {i:'Cambridge', p:110}]}
\end{lcverbatim}

\vspace{5mm}

ShareJS uses a server that provides a global, serialized order of operations to be applied on each client. This means the or

One important fact is that ShareJS only ever allows one packet to be in flight from a particular client to a server. Each pa


\section{Analysis}

\subsection{Memory}

\subsection{Network}

\subsection{Processor Load}

\subsection{Client-Server versus Peer to Peer}

\section{Starting Point}

\section{Requirements Analysis}

\section{Software Engineering}

\subsection{Libraries}

\subsection{Languages}

\subsection{Tooling}

\subsection{Backup Strategy}

\section{Early Design Decisions}



\chapter{Implementation}

This section will...

\section{CRDT-based system}

\subsection{Overview}

\subsection{Custom CRDT}

\subsection{Network simulation}
```

```
\section{ShareJS Comparative Environment}

\subsection{Overview}

\section{Experiments and Automated Log Analysis}

\subsection{Separation of Concerns}

\subsection{Experiment Design}

\subsection{Log Analysis}


\chapter{Evaluation}


\chapter{Conclusion}



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% the bibliography
\addcontentsline{toc}{chapter}{Bibliography}
\printbibliography

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% the appendices
\appendix

\chapter{Latex source}

\section{diss.tex}
{\scriptsize\verbatiminput{diss.tex}}

\section{proposal.tex}
{\scriptsize\verbatiminput{proposal.tex}}

\chapter{Makefile}

\section{makefile}\label{makefile}
{\scriptsize\verbatiminput{makefile.txt}}

\section{refs.bib}
{\scriptsize\verbatiminput{refs.bib}}


\chapter{Project Proposal}

\input{proposal}

\end{document}
```

A.2 proposal.tex

```
% Note: this file can be compiled on its own, but is also included by
% diss.tex (using the docmute.sty package to ignore the preamble)
\documentclass[12pt,a4paper,twoside]{article}
\usepackage[pdfborder={0 0 0}]{hyperref}
\usepackage[margin=25mm]{geometry}
\usepackage{graphicx}
\usepackage{parskip}
```

```
\usepackage[UKenglish]{babel}% Recommended
\usepackage[bibstyle=numeric,citestyle=numeric,backend=biber,natbib=true]{biblatex}

\addbibresource{proposal_bibliography.bib}


\begin{document}

\begin{center}
\Large
Computer Science Tripos -- Part II -- Project Proposal\\[4mm]
\LARGE
Conflict Free Document Editing with Different Technologies\\[4mm]

\large
J.~Send, Trinity Hall

Originator: J.~Send

10 October 2016
\end{center}

\vspace{5mm}

\textbf{Project Supervisor:} S.~Kollmann

\textbf{Director of Studies:} Prof.~S.~Moore

\textbf{Project Overseers:} Prof.~T.~Griffin \& Prof.~P.~Lio

% Main document

\section*{Introduction}

\subsection*{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~\cite{Technion}. There are two main technologies that are used to enable concurrent editing of a
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~\cite{sun1998operational}. 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 guaran
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 t
listed in later sections.

\subsection*{Resources required}

The primary external resource I will need is the Javascript library ShareJS, which is published under the MIT license on Git

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.

\subsection*{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.

\section*{Work to be done}

\subsection*{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 conclusi
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.

\subsection*{Detailed Project Structure}

\begin{enumerate}

\item \textbf{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.


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

\item \textbf{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.

\item \textbf{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.

\item \textbf{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.

\end{enumerate}



\subsection*{Possible extensions}

There are two extensions of varying difficulty:

\begin{itemize}

\item (Easier) Replace networking simulation with a P2P networking library. The end result of this extension should be a rea

\item (Difficult) Research prior work on undo and move functionality using CRDTs. If something suitable is found, implement

\end{itemize}
```

```
\section*{Success criteria}

These are the main success criteria associated with my project

\begin{enumerate}
\item A concurrent text editor based on CRDTs has been implemented.

\item The concurrent text editor passes all correctness tests.

\item Quantitative results comparing ShareJS and the CRDT based system have been obtained and analyzed.
\end{enumerate}


\section*{Timetable}

Planned starting date is 16/10/2011.

\begin{enumerate}

\item \textbf{Michaelmas weeks 2--3} Develop CRDT datastructure and algorithms on paper. Read into P2P networks
and simulating them.

\item \textbf{Michaelmas weeks 4--5} Lay out project files and implement network simulation with support for different
P2P topologies.

\item \textbf{Michaelmas weeks 6--8} Implement CRDT datastructures and algorithms, and connect these to network simulation.

\item \textbf{Michaelmas vacation} Learn an appropriate testing framework, write and generate unit tests for correctness of

\item \textbf{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.

\item \textbf{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.

\item \textbf{Lent weeks 5--6} Continue writing dissertation and optimizing system. Begin research for extension
which implements proper networking stack.

\item \textbf{Lent weeks 7--8} Continue writing dissertation. Before terms ends, review and peer-review (including superviso
incomplete draft. Implement networking extension. Research undo and move operations with CRDTs.

\item \textbf{Easter vacation:} Finish dissertation draft. Work on undo and move extensions for system.

\item \textbf{Easter term 0--2:} Edit and proof read dissertation. Work on extensions.

\item \textbf{Easter term 3:} Proof read and then submit early to concentrate on exams.

\end{enumerate}

\newpage

\printbibliography

\end{document}


\end{filecontents}
```

Appendix B

Makefile

B.1 makefile

B.2 refs.bib

```
@misc{MotherDemo,
  title = {The Mother of All Demos, Reel 3},
  howpublished = {\url{https://archive.org/details/XD300-25_68HighlightsAResearchCntAugHumanIntellect&start=286}},
  note = {Accessed: 2017-03-01}
}

@article{Ellis1989,
author = {Ellis, C A and Gibbs, S J},
title = {{Concurrency Control in Groupware Systems}},
year = {1989}
}

@article{Gilbert2005,
author = {Gilbert, Seth and Lynch, Nancy},
pages = {51--59},
title = {{Brewer ' s Conjecture and the Feasibility of Consistent , Available , Partition-Tolerant Web Services}},
year = {2005}
}

@BOOK{Lamport86,
TITLE = "{LaTeX} --- a document preparation system --- user's guide
and reference manual",
AUTHOR = "Lamport, L.",
PUBLISHER = "Addison-Wesley",
YEAR = "1986"}

@REPORT{Moore95,
TITLE = "How to prepare a dissertation in LaTeX",
AUTHOR = "Moore, S.W.",
YEAR = "1995"}
```

Appendix C

Project Proposal

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 [**Technion**]. 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 [**sun1998operational**]. 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 [**preguica2009commutative**]. 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[**Shapiro2011**]. 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 [**ShareJS**].

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.