Real-time P2P Collaborative Text Editor

Ke Er Xiong, BoYi Zhang, ZiYu Wang, De Li

Introduction

Real-time collaborative editing tools such as Google Docs have been extremely useful and is widely adopted for effective collaboration on various files and documents. However, many such systems rely on a central server for hosting and handling concurrent operations using operational transform. This introduces a single point of failure. We want to design and implement a P2P network for collaborative text editing that avoids the need of a central server and supports insertions and deletions on a shared document for up to 15 peers. From now on, we will refer to a peer as a node and a deletion or insertion on the shared document as an operation. The system will satisfy the following requirements:

- When the program starts, the user can immediately start editing. The user can join a network by connecting to an active node in the network. It can join a network even while it is already connected to another network. In case the two network will merge to form a single network and the document will be merged in a consistent way among all nodes.
- A node can also disconnect from the network, edit the document and reconnect later (offline editing).
- If all nodes fails or terminates, the shared document in that node will be lost. The user can choose to export a text file at any point.
- The network can withstand node failures and network partitioning.
- Operations applied locally at a node will be replicated to all other nodes in the system eventually. This is of course assuming that not all nodes with a replica fail at once.
- Concurrent operations can appear in different orders across different nodes.
- The shared document is eventually consistent: assuming no peer edits the document for a sufficient period of time, eventually all peers will have a consistent view of the document.
- The system is near real-time in the sense that it broadcast operations to other nodes as fast as possible as the network topology allows (e.g. in a complete graph this will be the latency between two nodes), and any node that received an operation will apply it immediately.

This system is to be implemented in Go based on an existing conflict-free replicated data type (CRDT) designed for collaborative text editing called Treedoc [1].

Proposed System Architecture

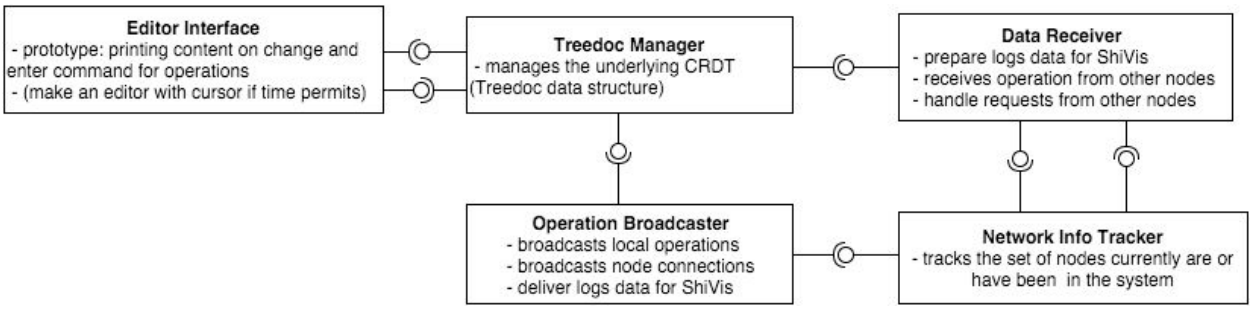


Figure: System architecture

Editor Interface

We are building the text editor as a command line application. Its usage is:
editor.go [localIp:port] -- start a new editor, other nodes connect on localip

The command line has access to following function calls by entering the following command:
connect(remoteIp:port) -- connect to network on remoteIp
disconnect() -- disconnect from the current network it's on
insert(pos, string) -- perform insertion of string at pos of current document
delete(pos, len) -- perform deletion of len character at pos of current document, len defaults to 1
pos is measured by number of characters before the position
exportDoc(path) -- export current document as txt file

The document will be printed whenever a change has occurred to it. (With a library we may be able to make this display in a separate view). We are also thinking of providing a more standard text editor GUI if time permits and the complexity is not too high. Some candidate libraries would be gocui [4] and termbox-go [5].

Node Management

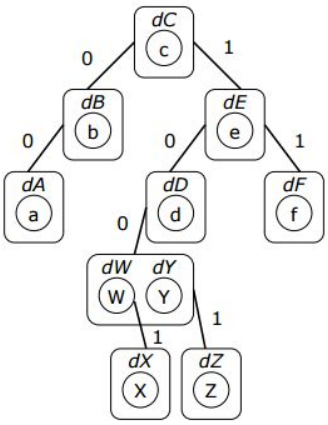
A node is uniquely identified by a uuid generated when it starts. Every time a node disconnect it will regenerates a new uuid to prevent conflict. Each operation is associated with the uuid and an incrementing counter which acts as a logical timestamp. Each time the user performs an operation on a node, its counter on that node will be incremented.

A node will store the following information:

- The nodes it's connected to. We will refer to those nodes as the neighbors of the node.
- All nodes it knows about in the network and what each node is connected to. We will refer to this information as network metadata.
- The replicated operations from other nodes.
- A local document represented internally with the chosen CRDT

Treedoc CRDT Implementation

Conflict-free replicated data type, or CRDT, is a data structure that achieves strong eventual consistency through monotonic operations [2]. We will be implementing an existing CRDT for collaborative text editing called TreeDoc [1] to ensure strong eventual consistency of the shared document in each node. TreeDoc stores the document as a binary tree. Each character is identified with a unique Position Id (PosId) which is a binary string describing a path in the binary tree (e.g. 0101 where 0 is the left child, 1 is a right child). The order of the characters in the document can be obtained by an infix-order traversal of the binary tree. (See figure on the right which is taken from [1].) When inserting character between two existing characters (character L and character R), it is inserted as the left child of character R or the right child of character L. When



deleting a character, it is marked as deleted and kept around as a tombstone in the tree. When concurrently inserting characters, character may be inserted at the same node of the tree. TreeDoc modified the binary tree to handle this case by making such nodes “major nodes” that contains a smaller subtree for each concurrent insertion. Each subtree is associated with a disambiguator (i.e. a unique ID that can be totally ordered so characters can still be ordered), and any PosId referring to characters in the major node needs to have this disambiguator. If we need to make any modification to the design of the CRDT for performance optimization purposes, we will prove it’s still a correct CRDT.

Topology of the Network and Joining the Network

The topology of the network is a complete graph, or at least attempts to be as complete as possible. A node can join the network by contacting and connecting to any node currently in the network and retrieving the list of all nodes it know are currently in the network. Then it will attempt to connect to other nodes in the system, retrieving the nodes that they know are in the network and repeat the step until it connects to all the nodes. TCP will be used for communication to provide reliable in order delivery and failure detection.

Maintaining the Network Metadata

The network metadata will be eventually consistent. Nodes periodically broadcast to their neighbors the nodes they are connected to. They also broadcast to a neighbor the network metadata that the neighbor may not have (e.g. in figure 2, node B does not need to send any metadata about C to A, but B will know A isn’t connected to D from its network metadata and send it to A). Assuming no change to the network occurs, eventually everyone will know network metadata for all nodes in the network.

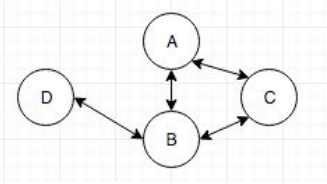


Figure: example of node network (D just joined B)

Failure and Disconnection From the Network

If a node fails, all its TCP connection close. It will be set as a closed connection from each node it was connected to. Eventually this information will be broadcast to every node in the system and every node will know the node is not currently connected to the network. When a network connection close and the node is still connected to the network, one of the node in the new will try to reconnect to the other periodically (e.g. smaller ip:port will contact larger ip:port). It’s up to a node to reconnect if it’s disconnected from the network.

Replication of Operations

When a node performs an operation, it will optimistically replicate it to all its neighbors by broadcasting to them. In some situations, an operation may not be replicated to all nodes in the system in its initial replication. Some of the situations includes:

- When a node disconnects and it only broadcasted to some of the nodes
- When a node joins and it hasn’t connected to all the nodes yet

- The network is not a complete graph

We will add metadata for each node. The metadata consist of the latest version of operation for each node in the network the node currently has replicated. This is replicated similarly to the process described in Maintaining the Network Metadata. If a node has a replica of an operation and finds out that one of its neighbor doesn't have it, then it sends this operation to the neighbor.

Network Merging, Partition and Offline Editing

We can view offline editing as merging a network of one node with another network. Healing a partition is simply merging two network. Partition is manually healed by a user by re-connecting to another network. Merging network simply involves replicating operations in one network to the other and vice versa.

Logging & Testing

We will test our application by running testing scripts that make multiple peers concurrently edit a single document. Editor outputs can be used to analyzes whether primary goals of our application (eg. eventual consistency, near real-time peer editing) are achieved. All operations done by a node and all communication between nodes will be logged for ShiViz analysis. Logs and ShiViz will be used to accurately evaluate our application's correctness when handling network edge cases such as node joining, rejoining, failure detection and network partitioning.

Challenges That We Expect to Resolve for This Project

Memory

Treedoc PosIds can grow unbounded. Reducing their size will reduce memory usage and network traffic. Is there any way to modified the TreeDoc structure to reduce their size?

Known Problems That We Will Not Resolve in This Project

Garbage Collection

A node can garbage collect Treedoc tombstones and replicas of an operation when all other nodes cannot refer to the tombstone again/has a replica of said operations. Since this system allows node to partition, it seems garbage collection is impossible as we can't distinguish a node that has failed as opposed to a node that has partitioned. A node that rejoin may also require operations from a long time ago so nodes cannot garbage collect anything. This is very inefficient, is there any way to avoid this?

Treedoc Performance Degradation

How fast is the performance degradation for TreeDoc due to accumulating tombstones and edits. How long can a shared document be edited before performance is a serious issue?

Timeline

Feb 27	All members should have read all the papers in the reference section
Feb 29	Project proposal
Mar 2	Implementations of simple CRDT for set as a primary to handling joining nodes

Mar 5	Very primitive command line interface to allow viewing, editing, and testing
Mar 15	Implementation of the barebone collaborative editing CRDT with no performance optimization. Implementation of a network where nodes can join/disconnect. Basic initial replication assuming a constant set of nodes.
Mar 18	Schedule Meeting with TA to update project status
Mar 25	Replication with edge cases such as network partitioning and node failures
Mar 28	An improved console editor interface (if project is on track)
Apr 2	Draft final report
Apr 5	Testing and tunings
Apr 11	Project code and final reports

SWOT Analysis

Strength ● We have the ability to meet frequently ● Our team is familiar with Git ● Team members have reasonable software development background through taking CPSC courses or Co-op ● We are interested in the project and the concepts involved	**Weakness** ● All team members have only two months of experience in Go ● Lack of knowledge about the use of some Go library (eg. Go Vector, Go Html) ● None of us are good at testing distributed system or automated testing in Go ● None of us are native English speakers
Opportunities ● There are existing papers of CRDTs on Treedoc [1] ● We found an existing GitHub project that implements simple CRDTs in Go [3]	**Threats** ● Projects and assignments from the other classes we are taking can pile up near the end of the term, resulting in insufficient time ● Unexpected problems or edge cases may be discovered when we test the system

References

[1] Nuno Preguica, Joan Manuel Marques, Marc Shapiro, Mihai Leia. A commutative replicated data type for cooperative editing. 29th IEEE International Conference on Distributed Computing Systems (ICDCS 2009), Jun 2009, Montreal, Quebec, Canada. IEEE Computer Society, pp.395-403, 2009, <10.1109/ICDCS.2009.20>.
[2] Marc Shapiro, Nuno Preguica, Carlos Baquero, Marek Zawirski. Conflict-free Replicated Data Types. [Research Report] RR-7687, 2011, pp.18.
[3] https://github.com/neurodrone/crdt
[4] https://github.com/jroimartin/gocui
[5] https://github.com/nsf/termbox-go