Peer-to-Peer Note Taking App

CS244B Project Report
Fatma Tlili - Nilanjan Chatterjee

Abstract:

We implement a real time, peer-to-peer collaborative text editor, which ensures synchronization and handles conflicts. Our system uses RPC [3] to connect and communicate between the different servants. RPC ensures portability and interoperability between RPC peers. We also adopt Conflict-Free Replicated Data Type [2] (CRDT) to detect and handle conflicts. The CRDT ensures that the insert and delete operations are commutative and idempotent. i.e insertion and deletion can happen interchangeably between users without getting out of sync, and repeated deletion is a no-op. Our system works for multiple nodes making sure that the process is smooth, the conflicts are handled in a reasonable manner and the updates are made in real time.

Introduction:

Non-collaborative editing across peers involves multiple iterations of sharing documents with each editing at a time and results in multiple versions. This process ends up taking a long time, since simultaneous edits need to be manually resolved which is time-taking. As such there is an inherent need for a real time editor by multiple peers. This poses many challenges since conflicts need to be resolved in real time, automatically. We look at various scenarios that produce conflicts and find ways to resolve them. We examine one such protocol called “Operational Transform” and discuss its advantages and disadvantages. Finally we discuss our implementation using CRDT protocol.

Challenges:

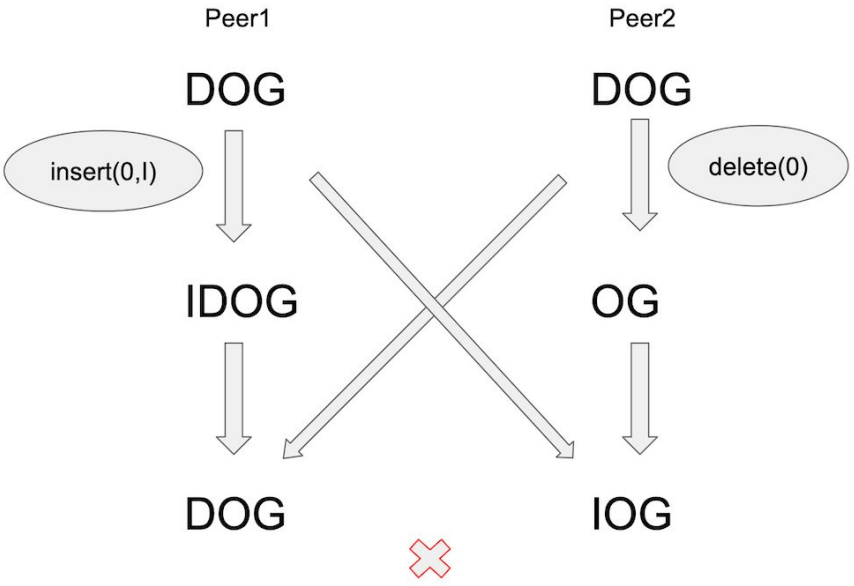
In order build a collaborative text editor we require two main conditions:

1- Commutativity: Insert and delete operations that happen concurrently must be commutative. i.e regardless of the order of the operations, the document converges to the same state.

2- Idempotency: Repeated delete operations result in one delete only. Note that we

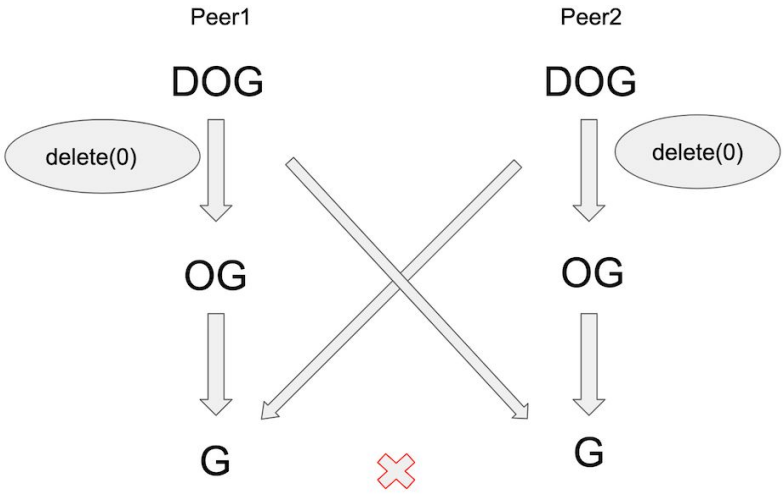
don’t require idempotency in repeated insert operations as those can be detected and corrected by the users, unlike the delete operations.

With absence of a synchronizing mechanism, let us see what happens when Peer1 does an “insert” and Peer2 does a “delete” at position 0:



The operations don’t commute

As we see, the insert and delete operations happen in a different order at each location and the documents don’t converge. This means our operations are not commutative. In another scenario, both peers try to delete the same character and they end up deleting two characters:

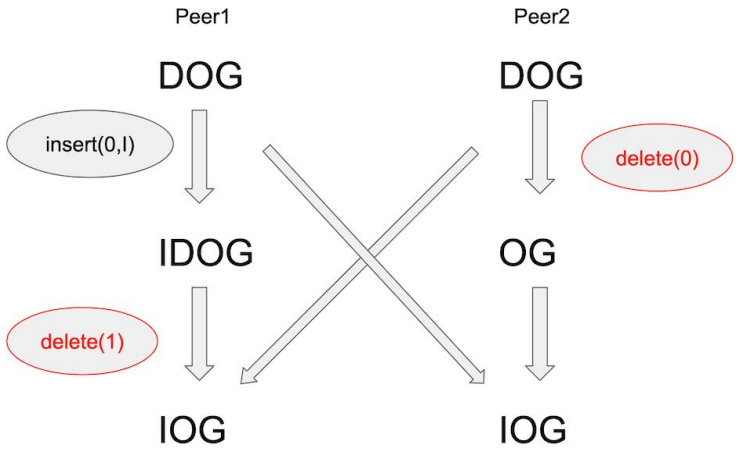


The operations are not idempotent

This means the delete operations are not idempotent, meaning if the same character is deleted from multiple locations, it does multiple deletes. This is dangerous since the user cannot always keep track of accidental deletes. To address these issues, we look at state of the art protocols that can resolve conflicts in real time.

Related Work - Operation Transform:

One technique that tries to address the issue is “Operation Transform” [1]. This algorithm which was proposed in 1989, aimed to transmit changes between users in order to reach eventual consistency. The algorithm detects operations from a starting state and finds the correct order of each edit. These operations are then transformed to achieve consistency. The figure below shows how the delete operation is transformed to achieve the intended goal by the peers.



Operation Transform Example

OT Challenges:

While OT had many forms over the years, the two main versions that have survived are *server-based OT* and *OT with transform property 2.* The latter was too complex and not scalable that very few applications have adopted it, and the former requires a centralized server. Thus conflict resolution continued to be a challenge until the appearance of CRDT.

Conflict Free Replicated Data Type:

Since most of the approaches that target Eventual Consistency are ad-hoc and error prone, or centralized, CRDT [2] is a relatively new approach (2011), which achieves consistency and synchronization while allowing a simpler implementation and a peer-to-peer architecture.

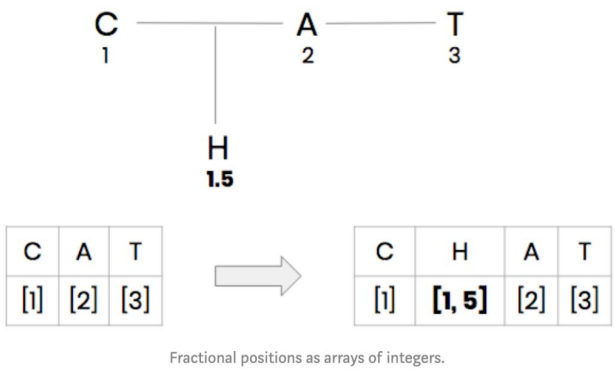
While OT preserves the underlying structure of the text editor and develops an algorithm to ensure commutativity and idempotency by modifying the operations, CRDT uses a different, more complex underlying data structure to guarantee the requirements by changing the overall state. We use sequence CRDT which is a data structure for sequential data that can be replicated across multiple machines in a network; these replicas are guaranteed to converge in a concurrent and automatic manner, regardless of the failures of some nodes.

1- Character Objects:

There is no reason to treat characters as absolute values and positions, instead, CRDTs represent characters as data structures which encapsulate multiple properties of the characters allowing for a much simpler algorithm than OT.
The main properties for a character object are:

- Globally Unique Characters: CRDT gives unique ID's to each character which makes achieving idempotency in deletes ops trivial
- Globally Ordered Characters: CRDT requires preserving a global order of the characters, which makes the delete and insert operations commutative.

There are different ways in which we can guarantee the above properties. In our implementation, the character object has the following attributes:
Site ID: unique ID to the user who created this character
Site Counter: The operation number from that particular user, incremented after each insert. To ensure that the characters are unique, new identifiers are generated by combining the site ID and site counter.
Value: The actual character
Global Index: A vector of integers representing indices as floats where the integers represent the digits after the fraction. We limit all the indices to be less than 1 so it's easily comparable. Using fractional indices allows the characters to be globally ordered such as shown in the figure below [4].



Fractional positions as arrays of integers.

2- Edge Cases:

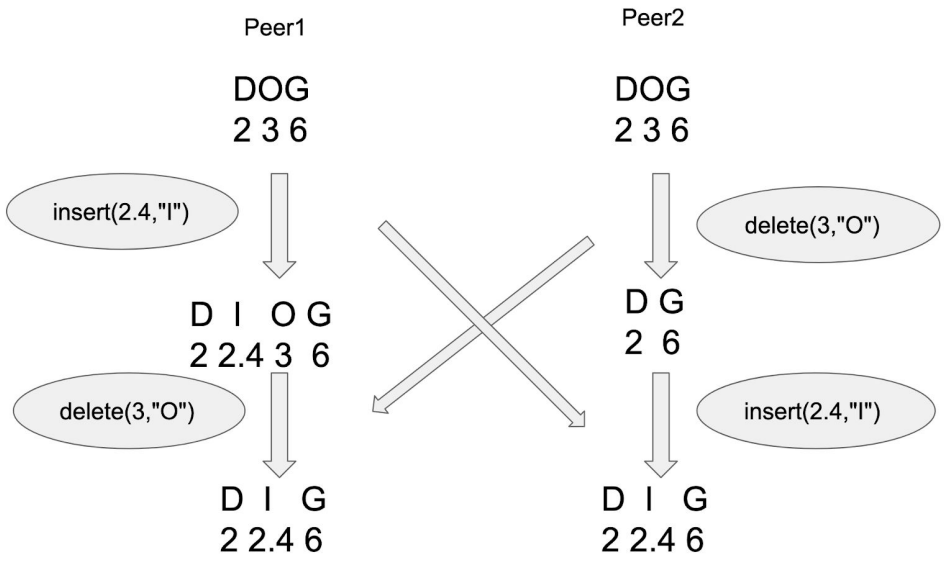
Inserting at the same position: If 2 peers are trying to insert at the same position, either one of the orders will be acceptable, but we have to ensure synchronization between the peers. Thus we only insert at the requested position if there is no higher identifier at the next position. Otherwise we keep walking to the right until we find a position with a smaller identifier and insert there.
Out of order operations: If a delete operation of a character precedes its insert operation due to network delays, we implement a version vector for each peer, to track the operations order. It stores the delete operations that can't be applied and gets updated after receiving every insert from other peers to check if any of the inserts can be cancelled out by the buffered delete.

3- Trade-offs:
While CRDT simplifies the implementation to great extent and minimizes the number of corner cases to be taken care of compared to OT, it introduces a lot of metadata per character. Our code has functions to serialize and deserialize the characters' data and prepare it to be sent over the network. However this increases the overall memory consumption.

4- Protocol Summary:

1. A peer inserts a letter to their text editor
2. That change is added to their CRDT and converted into a character object.
3. That local insertion is then broadcasted out to the rest of the peers using RPC.
4. Those received operations are verified against the Version Vector before being incorporated into the editor.

The figure below shows an example of how operations commute using CRDT.



Insert and Delete operations commute using CRDT

Peer-to-Peer Architecture:

Our first implementation was using sockets and had multiple issues in connecting peers to each other. To solve this issue we tried to use an off-the-shelf solution and found many libraries that offered the service. Some examples are grpc, rpclib. We ultimately used “rpcgen”. rpcgen [3] is an interface generator precompiler for Sun Microsystems ONC RPC. It offers a network interface which is compatible with our C++ system and allows us to implement a peer-to-peer architecture.

Conclusion and Future Work:

We implement a peer-to-peer collaborative text editor which relies on CRDT to handle conflicts. Our code which is implemented in C++ uses QT for the user interface. We would like to build a framework for a web application instead which would allow further testing with multiple nodes at a time.

References:

[1] Sun, C., & Ellis, C. (1998, November). Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work* (pp. 59-68).

[2] Shapiro, M., Preguiça, N., Baquero, C., & Zawirski, M. (2011, October). Conflict-free replicated data types. In *Symposium on Self-Stabilizing Systems* (pp. 386-400). Springer, Berlin, Heidelberg.

[3] Corbin, J. R. (1991). Rpcgen. In *The Art of Distributed Applications* (pp. 179-206). Springer, New York, NY.

[4] Fractional indices image taken from: https://conclave-team.github.io/conclave-site/#conflict-free-replicated-data-type-crdt