Concurrency in Collaborative Editing

Let’s explore different methods of conflict resolution in concurrent collaboration on a document.

We'll cover the following

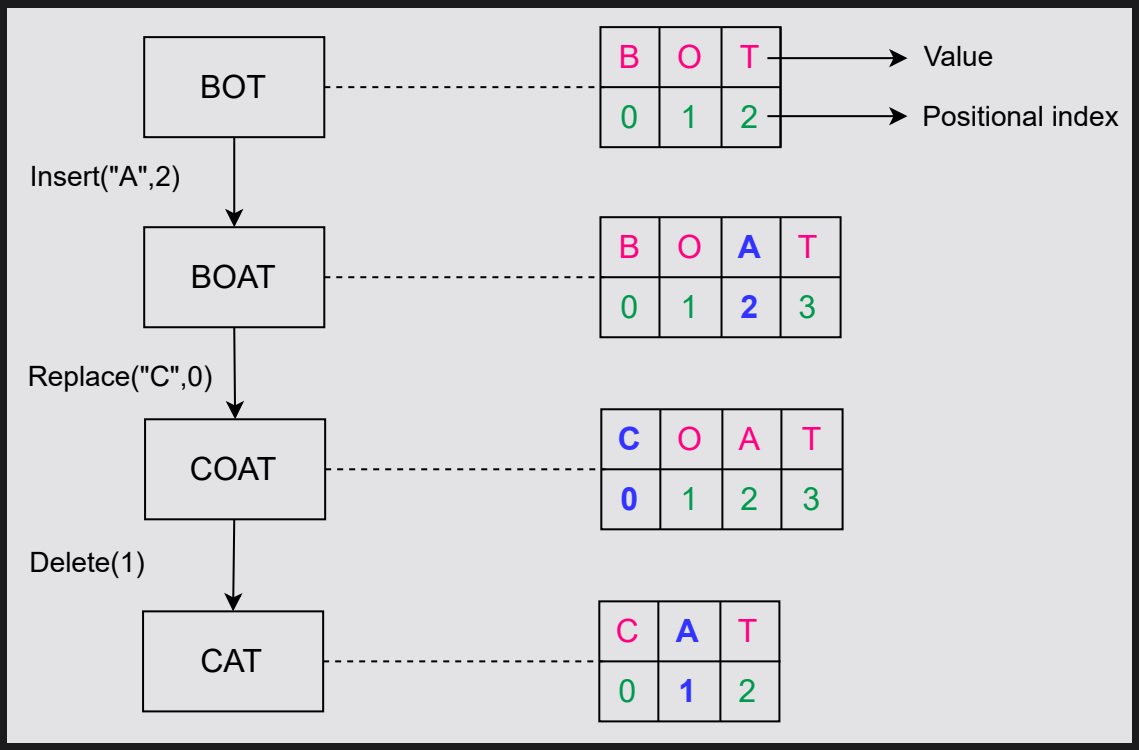
Introduction

We’ve discussed the design for a collaborative document editing service, but we haven’t addressed how we’ll deal with concurrent changes in the document by different users. However, before discussing concurrency issues, we need to understand the collaborative text editor.

What is a document editor?

A **document** is a composition of characters in a specific order. Each character has a value and a positional index. The character can be a letter, a number, a enter (↵), or a space. An index represents the character’s position within the ordered list of characters.

The job of the text or document editor is to perform operations like `insert()`, `delete()`, `edit()`, and more on the characters within the document. A depiction of a document and how the editor will perform these operations is below.



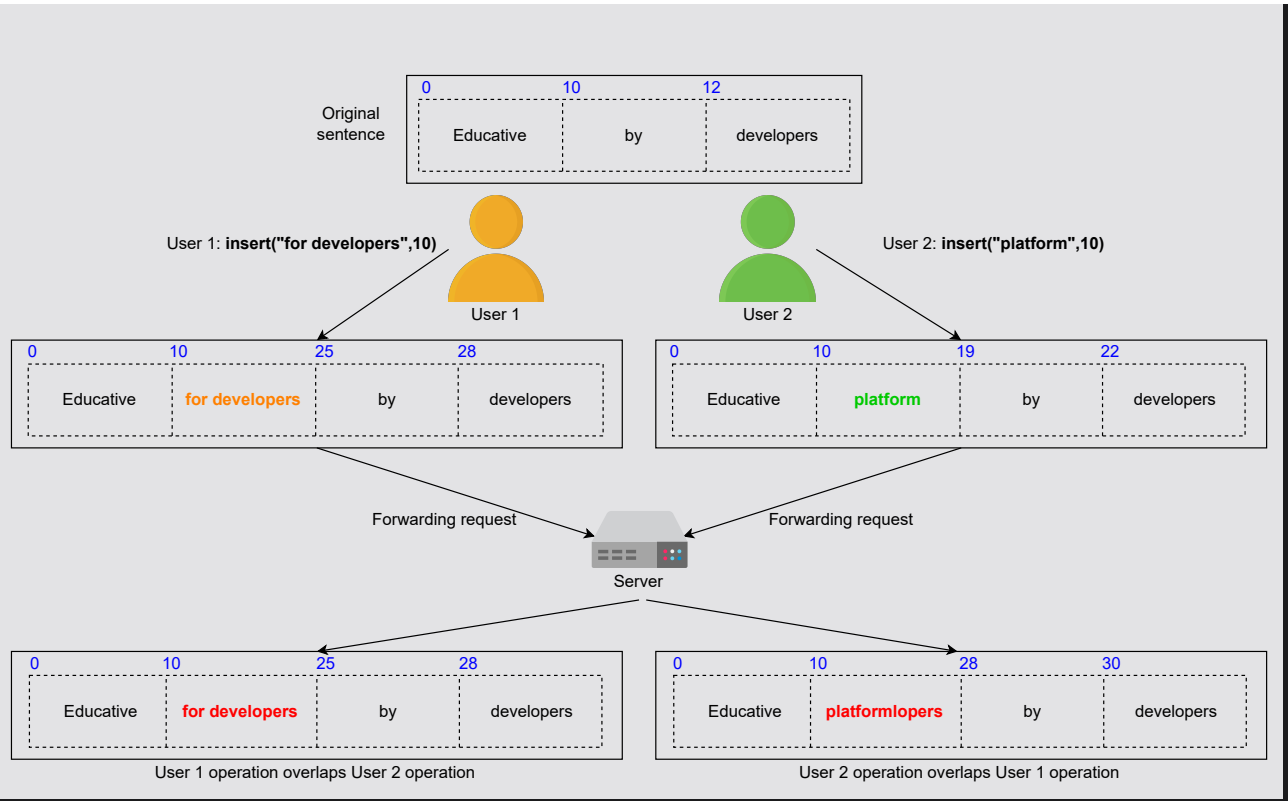
How a document editor performs different operations

Concurrency issues

Collaboration on the same document by different users can lead to concurrency issues. Conflicts may arise whenever multiple users edit the same portion of a document. Since users have a local copy of the document, the final status of the document may be different at the server from what the users see at their end. After the server pushes the updated version, users discover the unexpected outcome.

Example 1

Let's consider a scenario where two users want to add some characters at the same positional index. Below, we've depicted how two users modifying the same sentence can lead to conflicting results:



How two users modifying the same text can lead to concurrency issues

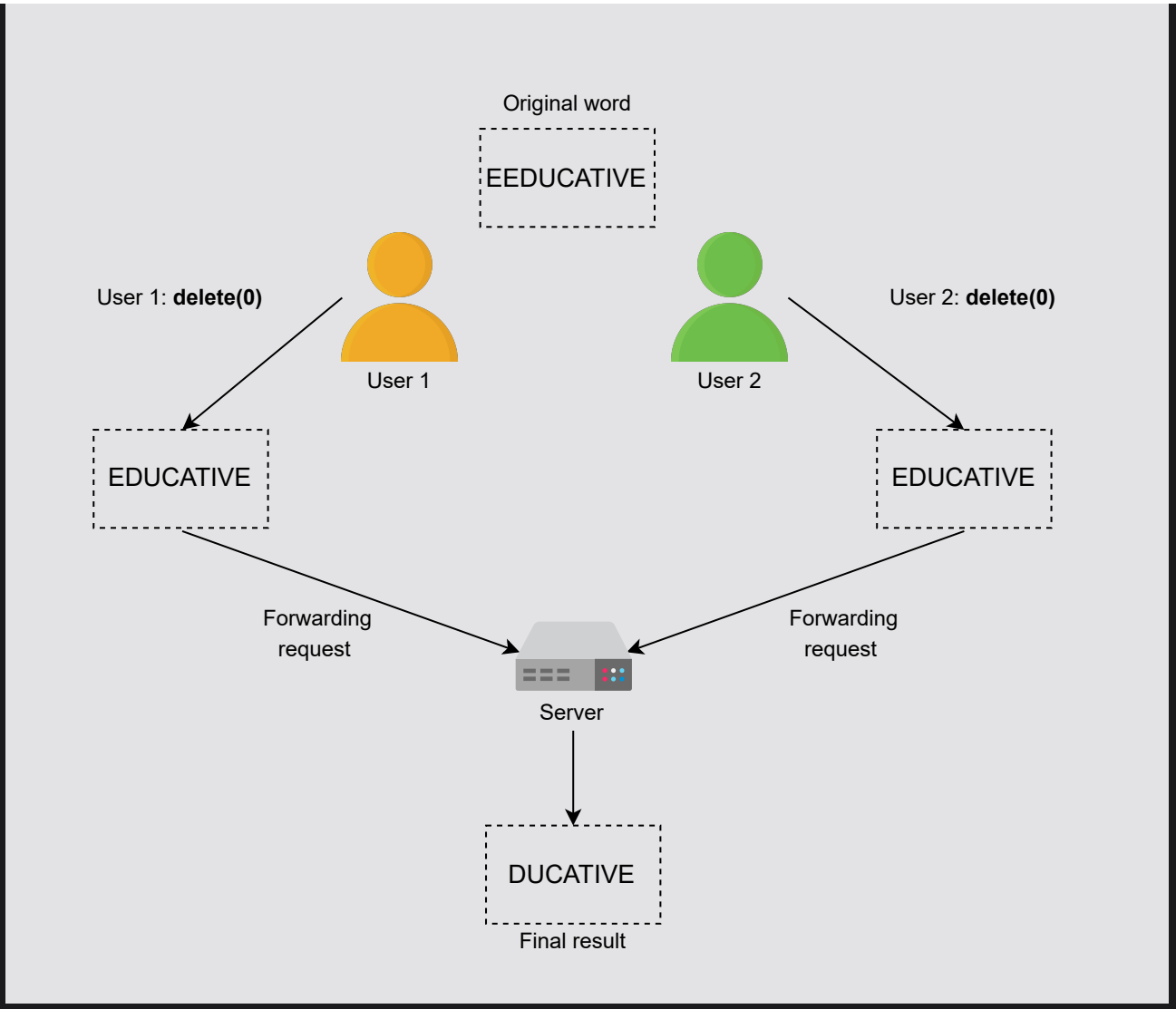
As shown above, two users are trying to modify the same sentence, “Educative by developers.” Both users are performing `insert()` at index `10`. The two possibilities are as follows:

- The phrase “for developers” overwrites “platform.” This leads to an outcome of “for developers.”
- The phrase “platform” overlaps “for developers.” This leads to an outcome of “platformlopers.”

This example shows that operations applied in a different order don’t hold the commutative property.

Example 2

Let’s look at another simple example where two users are trying to delete the same character from a word. Let’s use the word “EEDUCATIVE.” Because the word has an extra “E,” both the users would want to remove the extra characte Below, we see how an unexpected result can occur:



How deleting the same character can lead to unexpected changes

This second example shows that different users applying the same operation won't be idempotent. So, conflict resolution is necessary where multiple collaborators are editing the same portion of the document at the same time.

From the examples above, we understand that a solution to concurrency issues in collaborative editing should respect two rules:

- **Commutativity**: The order of applied operations shouldn't affect the end result.
- **Idempotency**: Similar operations that have been repeated should apply only once.

Below, we identify two well-known techniques for conflict resolution.

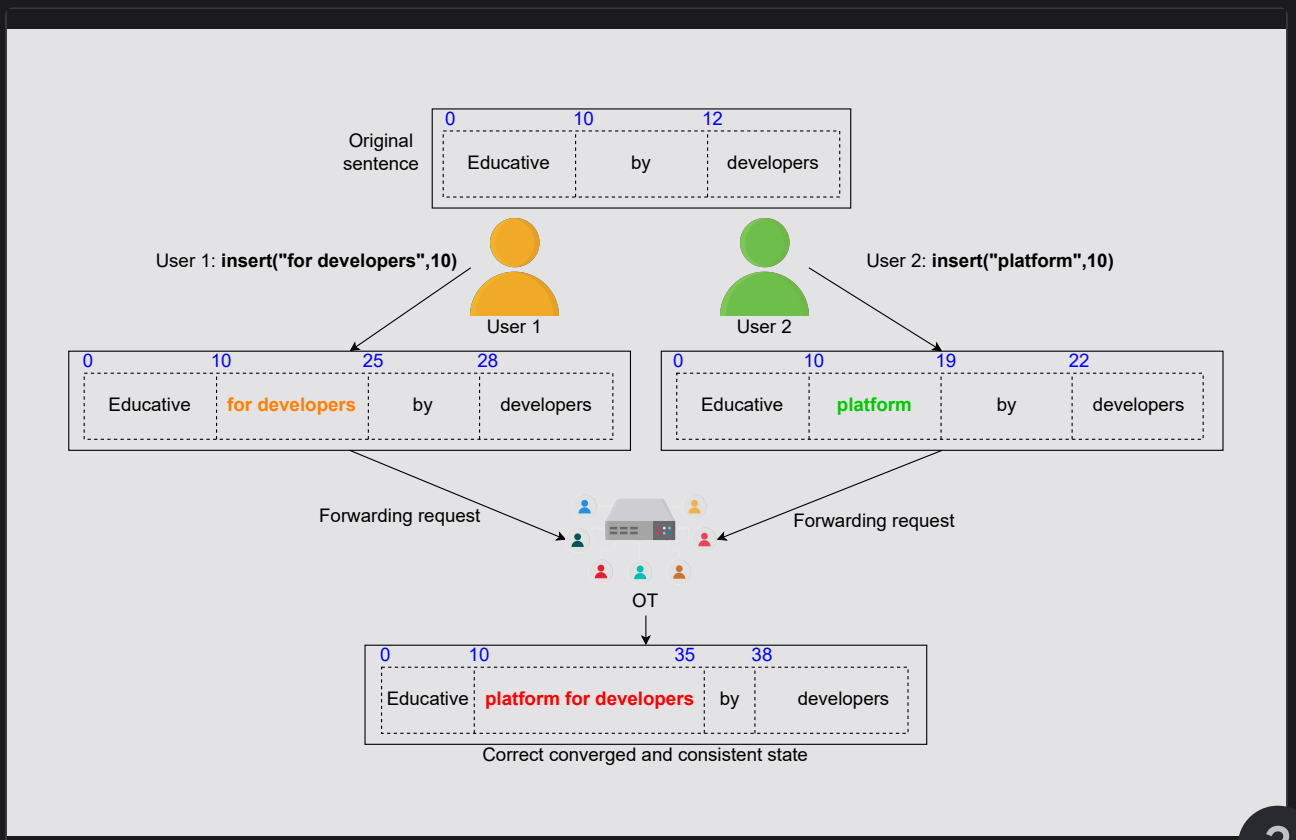
Techniques for conflict resolution

Let’s discuss two leading technologies that are used for conflict resolution in collaborative editing.

Operational transformation

Operational transformation (OT) is a technique that’s widely used for conflict resolution in collaborative editing. OT emerged in 1989 and has improved throughout the years. It’s a lock-free and non-blocking approach for conflict resolution. If operations between collaborators conflict, OT resolves conflicts and pushes the correct converged state to end users. As a result, OT provides consistency for users.

OT performs operations using the positional index method to resolve conflicts, such as the ones we discussed above. OT resolves the problems above by holding commutativity and idempotency.



Ensuring commutativity

1 of

Collaborative editors based on OT are consistent if they have the following two properties:

- **Causality preservation**: If operation `a` happened before operation `b`, then operation `a` is executed before operation `b`.
- **Convergence**: All document replicas at different clients will eventually be identical.

The properties above are a part of the **CC consistency model**, which is a model for consistency maintenance in collaborative editing.

Consistency Models in OT

OT has two disadvantages:

- Each operation to characters may require changes to the positional index. This means that operations are **order dependent**↴ on each other.
- It's challenging to develop and implement.

Operational transformation is a set of complex algorithms, and its correct implementation has proved challenging for real-world applications. For example, the Google Wave team took two years to implement an OT algorithm.

Conflict-free Replicated Data Type (CRDT)

The **Conflict-free Replicated Data Type (CRDT)** was developed in an effort to improve OT. A CRDT has a complex data structure but a simplified algorithm.

A CRDT satisfies both commutativity and idempotency by assigning two key properties to each character:

- It assigns a globally unique identity to each character.
- It globally orders each character.

Each operation now has an updated data structure:

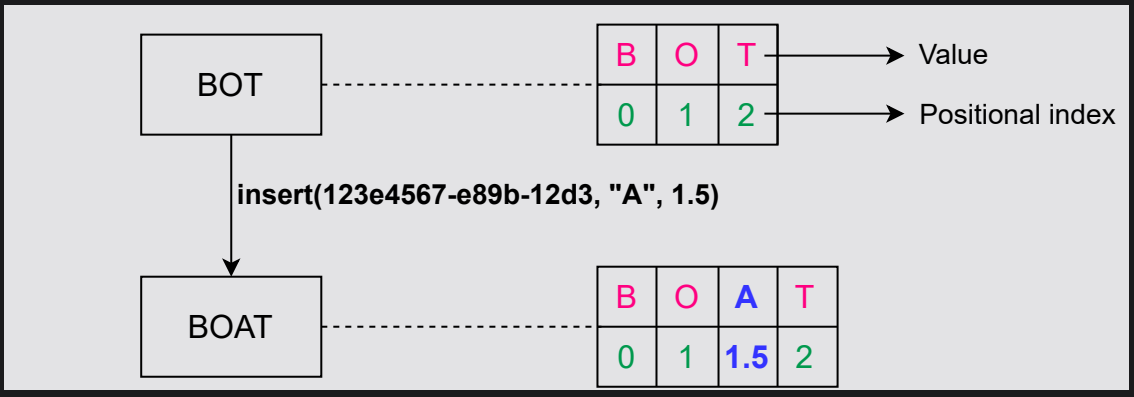
Data	Explanation	Example
SiteiD	Unique Site identifier in the form of a UUID	123e4567-e89b-12d3
SiteCounter	The sequence of operations from a site	87
Value	Value of character	"A"
PositionalIndex	Unique position of each character	4.5

Simplified data structure of a CRDT

The `SiteID` uniquely identifies a user's site requesting an operation with a `Value` and a `PositionalIndex`. The value of `PositionalIndex` can be in fractions for two main reasons.

- The `PositionalIndex` of other characters won't change because of certain operations.
- The order dependency of operations between different users will be avoided.

The example below depicts that a user from site ID `123e4567-e89b-12d3` is inserting a character with a value of `A` at a `PositionalIndex` of `1.5`. Although a new character is added, the positional indexes of existing characters are preserved using fractional indices. Therefore, the order dependency between operations is avoided. As shown below, an `insert()` between `O` and `T` didn't affect the position of `T`.



Preventing order dependency between operations

CRDTs ensure strong consistency between users. Even if some users are offline, the local replicas at end users will converge when they come back

online.

Although well-known online editing platforms like Google Docs, Etherpad, and Firepad use OT, CRDTs have made concurrency and consistency in collaborative document editing easy. In fact, with CRDTs, it's possible to implement a serverless peer-to-peer collaborative document editing service.

> **Note:** OT and CRDTs are good solutions for conflict resolution in collaborative editing, but our use of WebSockets makes it possible to highlight a collaborator's cursor. Other users will anticipate the position of a collaborator's next operation and naturally avoid conflict.

Quiz

Question 1

Why can't we use locks to synchronize between users?

Show Answer

1 of 2

Back

Completed

Next