Designing and Implementing Multi-user Distributed Text Editor

# Introduction

It is required to design and implement a multi-user distributed text editor, which will allow several users to collaborate in reading and editing text documents in a real-time environment where changes at any user will be broadcasted to all the other users at the same exact time of the changes. It is more like we need to design and implement a clone of Google Documents application.

# System Design

There are several functional and non-functional requirements that we must follow in order to start our system design plan.

## Functional Requirements

* Users can change a document at the same time without any conflict
* Allow sharing documents between users through a unique document ID and hyper-link

## Non-Functional Requirements

* The system must support multiple clients or autonomous agents like an API for sharing and updating data.
* The system should be distributed across multiple clients or server nodes.
* The system should be robust
* The system should be able to continue operation even if one of the participant nodes crashes
  + If three clients are collaborating on the same document and one client failed, the other two should continue collaboration on the system achieving reliability.
* It should be possible to recover the state of the node following a crash, to continue operation
  + Retrieving the final state of the document when the node goes back online
* The system should maintain multiple replicas for fault tolerance.

## Design Constraints

* Concurrency
  + Since several users are working on the same document
  + Operational Transformation
* Latency
  + Clients are working in different places, and the connection is established through the internet, so there is a latency between each and all clients when they are collaborating on the same document.
  + RESTful vs Publish/Subscribe Architectures.

## Operational Transformation

In order to provide real-time and collaborative environment in a text editor, we must consider any conflicts that may arise when more than two nodes are collaborating. For example, if a node inserted some text at position x, and another node deleted the text that exists at position x at the same time. Here, we present Operational Transformation, which is a technology that aims to solve conflicts in real-time collaborative editing environments. In order to do that, we must maintain consistency between local replicas of documents, since each client have its own local copy of the document.

A document will be stored as a sequence of operations in order of execution instead of plain text. So, we need a collaboration protocol to understand when to apply changes. We thought about identifying possible operations into three types:

* Insert Text
* Delete Text

Whenever we edit a document, all the changes are appended to the document saving these operations in one of those three types. In addition to saving operations by each user in a changelog database.

## WebSocket vs HTTP

## RESTful vs Publish/Subscribe Architectures

# High-Level System Design

# References