educative Ask a Question

# Evaluation of Google Docs' Design

Let's look at how we'll fulfill the non-functional requirements in a collaborative document editing system.

We'll cover the following

We've now explained the design and how it fulfills the functional requirements for a collaborative document editing service. This lesson will focus on how our design meets the non-functional requirements. In particular, we'll focus on consistency, latency, scalability, and availability.

## Consistency

We've looked at how we'll achieve strong consistency for conflict resolution in a document through two technologies: operational transformation (OT) and Conflict-free Resolution Data Types (CRDTs). In addition, a time series database enables us to preserve the order of events. Once OT or CRDT has resolved any conflicts, the final result is saved in the database. This helps us achieve consistency in terms of individual operations.

We're also interested in keeping the document state consistent across different servers in a data center. To replicate an updated state of a document within the same data center at the same time, we can use peer-to-peer protocols like Gossip protocol. Not only will this strategy improve consistency, it will also improve availability.

Quiz

Question

Why should we use strong consistency instead of eventual consistency for conflict resolution in a collaborative document editing service?

Show Answer

## Latency

Latency may feel like a challenge, specifically when two users are distant from each other or the server. However, users maintain a replica of documents at their end while data is being propagated through WebSockets to end-servers. Therefore, user-perceived latency will be low. Apart from this, users mostly tend to write textual data in a document that's small. Therefore, dissemination of data among different servers within the same facility and among different data centers or zones will be carried out with low latency. Also, files like videos and images can be stored in CDNs for quick serving because this content doesn't change frequently.

Practically speaking, there will be a limited number of readers and writers of an online document. For readers specifically, latency won't be an issue because the document will be loaded only once. So, most readers can be served from the same data center. For writers, an optimal zone should be selected as the centralized location between collaborators of the same document. However, for popular documents, asynchronous replication will be an effective method to achieve good performance and low latency for a large number of users. In general, achieving strong consistency becomes a challenge when replication is asynchronous.

## Availability

Our design ensures availability by using replicas and monitoring the primary and replica servers using monitoring services. Key components like the operations queue and data stores internally manage their replication.

Since we use WebSockets, our WebSocket servers can connect users to the session maintenance servers that will determine if a user is actively viewing or collaborating on a document. Therefore, keeping multiple WebSocket servers will increase the availability of the design. Lastly, we employ caching services and CDNs to improve availability in case of failures.

However, at the moment, we haven't devised a disaster recovery management scheme.

## Fulfilling Non-functional Requirements

| Requirements | Techniques |
|---|---|
| Consistency | • Gossip protocol to replicate operations of a document within t center<br>• Concurrency techniques like OT and CRDTs<br>• Usage of time series database for maintaining the order of ope<br>• Replication between data centers |

| | |
|---|---|
| Latency | • Employing WebSockets<br>• Asynchronous replication of data<br>• Choosing optimal location for document creation and serving<br>• Using CDNs for serving videos and images<br>• Using Redis to store different data structures including CRDTs<br>• Appropriate NoSQL databases for the required functionality |
| Availability | • Replication of components to avoid SPOFs<br>• Using multiple WebSocket servers for users that may occasionally<br>• Component isolation improves availability<br>• Implementing disaster recovery protocols like backup, replicatior zones, and global server load balancing<br>• Using monitoring and configuration services |
| Scalability | • Different data stores for different purposes enable scalability<br>• Horizontal sharding of RDBMS<br>• CDNs capable of handling a large number of requests for big files |

## Scalability

Since we've used microservice architecture, we can easily scale each component individually in case the number of requests on the operations queue exceeds its capacity. We can use multiple operations queues. In that case, each operations queue will be responsible for a single document. We can forward operations requested by different users that are associated with a single document to a specific queue. The number of spawned queues will be equal to the number of active documents. As a result, we're able to achieve horizontal scalability.

## Conclusion

In this chapter, we designed an online collaborative document editing service. In our design, we provided features like collaborative editing, keeping version history for reverting to older versions, giving users suggestions on frequently used terms and phrases, and view counts of a document. We also assessed the possibility of adding a chatting feature between users collaborating on the same document. A unique aspect of the design was the conflict resolution between concurrent editing operations by different users. We solved the concurrency issues through OT and CRDT.

← Back
Concurrency in Collaborative Editing

Next →
Introduction to Distributed System Failures

✓ Mark as Completed