Design Google Docs

Google Docs is an online word processing tool that is part of Google's free, web-based Google Docs Editor package. It has many features and is a complex system. If you take a closer look at how it is built and how it works, you may realize that it is much more intricate than it appears at first glance. Without any much delay let’s see how Google Docs work :)

Key requirements

Google Docs is a large system with many features, including document storage, sharing, formatting, and editing. The system can be broken down into these main components:

- File organization: users can create and remove folders to organize their documents.
- Online editing: users can edit and format documents in real-time.
- Collaboration: multiple users can work on the same document simultaneously.
- Sharing and permissions: users can share documents and assign different permissions (e.g., owner, read-only, allow comments).

High Level Design

It’s always convenient to have a high-level overview of the system we’re trying to build. One way to simplify a complex system is to divide it into smaller components. Let's take a look at these components:

Storage and Formatting

Google Docs has a storage mechanism similar to an operating system, with concepts such as directories, files, and owners. A file object is a fundamental component of the system, and includes content, parent, owner, and additional metadata like creation date. The root directory has an empty parent, which represents the folder hierarchy.

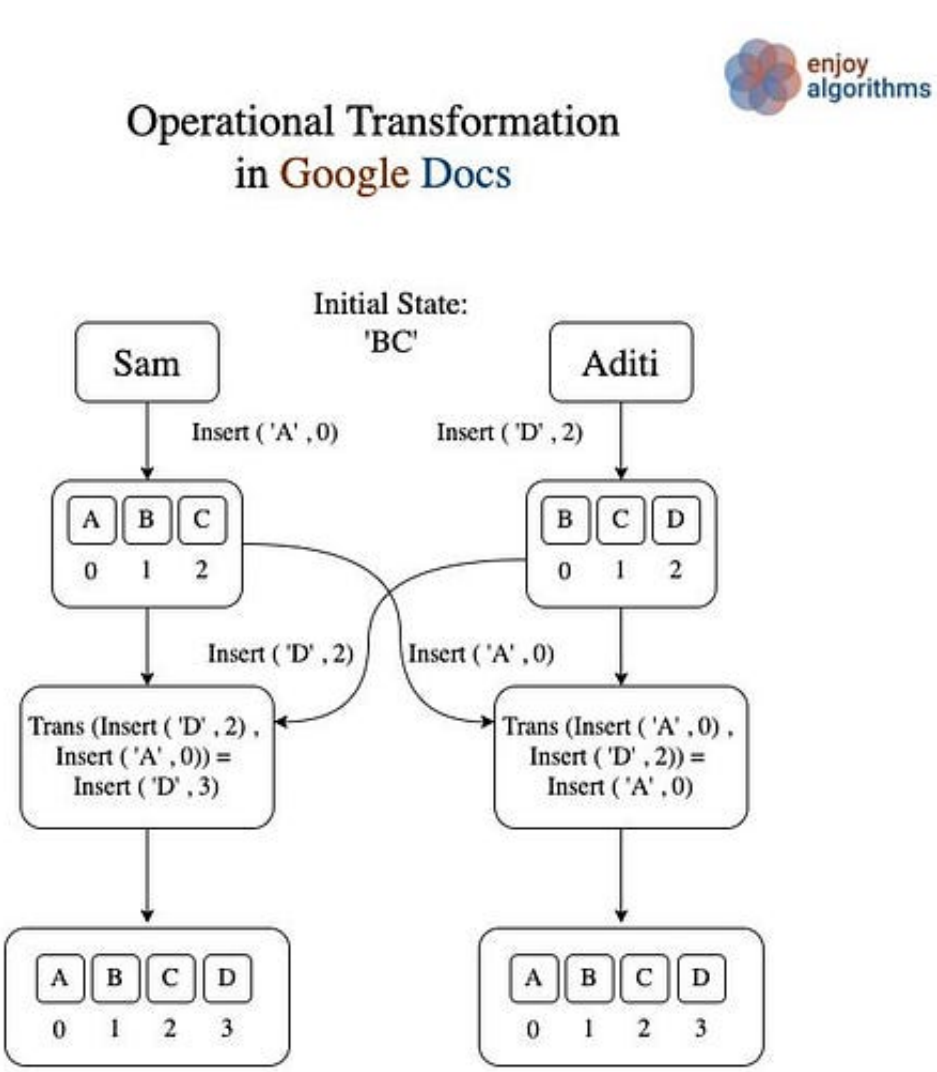
Concurrency

Google Docs allows multiple users to collaborate on the same document, but how does it achieve this? There are two types of algorithms used for collaborative editing: **Conflict-free Replicated Data Types (CRDTs**) and **Operational Transformation (OT)**.

Google Docs uses an event passing mechanism called operational transformation to maintain and synchronize a consistent state between users working on a shared document.

For example, if two users, Sam and Aditi, are working on the same document, which has an initial state "BC", and they both make changes to it. Their changes will be sent to the server along with their last revision. Specifically, if Aditi inserts the character "D" at position 2 and Sam inserts the character "A" at the beginning, the server will transform Aditi's change to be inserted at the 3rd index. The server keeps track of the complete revision history of each document, allowing each collaborator to edit the document.

The figure below shows how operational transformation works:



Access Control

Google Docs allows you to invite collaborators and assign them various levels of permission, such as read-only or owner. However, for more precise access control, you can use Role-Based Access Control (RBAC). RBAC ensures that employees only have access to the information and resources needed to fulfill their specific job responsibilities, and cannot access information that is not relevant to their role. This helps to protect sensitive data and maintain network security.

In the RBAC model, access can be restricted to specific actions, such as reading, creating, or editing files, based on various variables including authorization, responsibility, and job expertise. This ensures that only those with the appropriate permissions can perform certain actions, which helps to protect sensitive data and critical applications. RBAC has several advantages, including:

- Increasing operational effectiveness
- Increased visibility of the owner
- Cost reduction
- Reducing the risk of data breaches

How Google Docs work?

Designing a collaborative document editing platform can be challenging due to several factors. One difficulty is in allowing multiple users to edit the same document without conflicting changes. Another challenge is ensuring that simultaneous edits are effectively merged without causing issues. Both of these issues require careful planning and design in order to provide a seamless and efficient collaboration experience for users.

Operational transformation is a technique used in Google Docs to solve the first problem of allowing multiple users to edit a document simultaneously. The second problem is solved by the collaborative protocol. When you open a Google document, code is running in two places: your web browser and the Google servers. The code that runs in your browser is called the client. The client processes all of your changes to the document before sending them to the server, and then processes changes made by other editors after receiving them from the server.

To effectively collaborate in Google Docs, each client must keep track of four pieces of information:

- The latest revision received from the server.
- Any local modifications that have not yet been sent to the server.
- Any local modifications that have been sent to the server, but have not yet been acknowledged.
- The current state of the document as it is visible to the editor.

The server maintains three pieces of information:

- A list of all modifications it has received but has not yet processed.
- The complete history of all changes made to the document.

- The current state of the document as of the latest processed update.

By effectively using the information available to them, it is possible to design the client-server communication in a way that allows all editors to quickly process each other's updates in real-time. Let's examine how this is done in a document with a simple example.

Workflow

Let’s look into actual flow by considering an example:

1. Let’s say Bob begins the text by entering the word Hello at the top.
2. Bob keeps typing and adds the term world to his paper. John types an ‘!’ in his empty version of the document at the exact moment.
3. The edit was added to Bob’s client’s list of pending revisions. He then sent the update to the server and added it to his sent changes list.
4. John got Bob’s edit from the server and transformed it against his pending (‘!’)update using operational transformation (OT). As a result of the transition, John’s pending change was moved up five spaces to create a place for Bob’s Hello at the top of the document. When Bob and John received the notifications from the server, they both changed their last synced revision numbers to 1.
5. Following that, both Bob and John will send their pending changes to the server.
6. Because Bob’s change arrived before John’s, the server processed it first. Bob received a confirmation of the adjustment. The modification was delivered to John, who had it changed against his (‘!) change, which was still waiting.
7. The server received John’s pending modification, and John believes it should be Revision 2. The server, however, has already added Revision 2 to the revision log. The server will apply OT to John’s patch and save it as Revision 3.
8. The server started by comparing John’s transmitted change to all the other modifications committed since the last time John synced with the server. It turned John’s change against Bob’s in this example

('Google' at 6). As a result, John's change over-index was pushed by 6. This shift is identical to John's client's metamorphosis when he first received Bob's ('Hello' at 1).

The above example concludes with Bob and John receiving and acknowledging John's change, respectively. The server and both editors are staring at the same page at this point—Hello Google!

So this is how Google Docs work :)

Monitoring and Observability

To ensure the smooth functioning and high availability of our system, it is important to monitor and troubleshoot it by providing warnings when performance or availability decline. One way to do this is by integrating Google Cloud into our observability and alerting procedures.

Google Cloud services automatically provide observability data, such as metrics, logs, and trace data, which allows for a comprehensive view of the system. We can also use third-party monitoring, alerting, and notification solutions to transmit observability and event data for our Google Cloud services. This data can be extracted and integrated into our current reliability and alerting systems.

By combining these approaches, we can effectively monitor and troubleshoot our system to ensure it is running smoothly and efficiently.

Conclusion

In this blog post, we attempted to explain how Google Docs works. It can be challenging to design real-time products like Google Docs. We hope that by reading this article, you gained some insight into the inner workings of Google Docs and the considerations that must be made to ensure a seamless collaborative experience. If you have any thoughts or comments, please share them in the message section below.

Thanks to Suyash Namdeo for his contribution in creating the first version