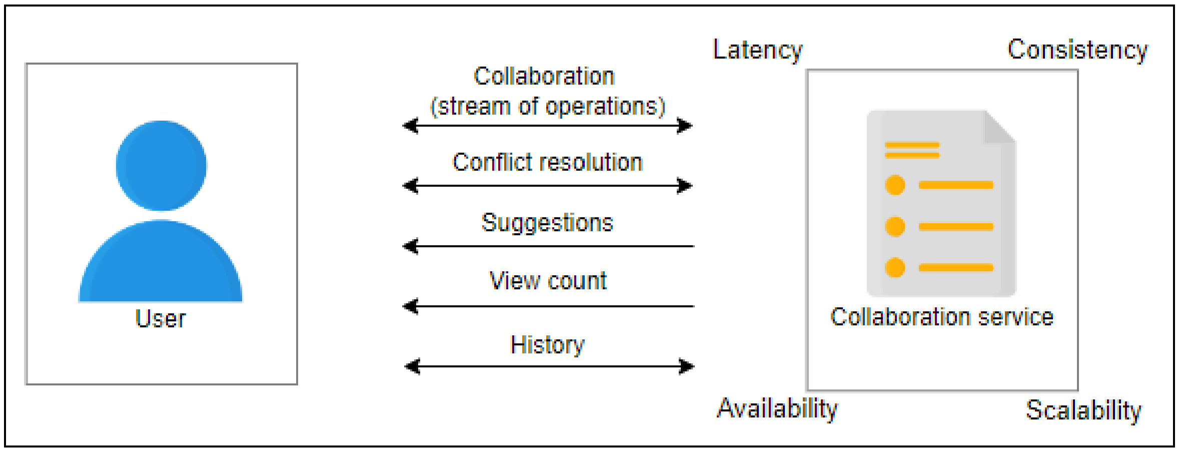
Collaborative Document Editing Service

The aim of the article is to design collaborative editing applications like Google Docs or Etherpad.

Functional requirements

- Multiple users should be able to edit a document simultaneously. Also, a large number of users should be able to view a document.
- The system should push the edits done by one user to all the other collaborators. The system should also resolve conflicts between users if they're editing the same portion of the document.
- The user should get suggestions about completing frequently used words, phrases, and keywords in a document, as well as suggestions about fixing grammatical mistakes.
- Editors of the document should be able to see the view count of the document.
- The user should be able to see the history of collaboration on the document.
- A user can add a comment to the document.
- Should send a notification (Email, GCM notification) to the subscribers of that space like confluence.



Non-functional Requirements

- **Latency**: Different users can be connected to collaborate on the same document. Maintaining low latency is challenging for users connected from different regions.
- **Consistency**: The system should be able to resolve conflicts between users editing the document concurrently, thereby enabling a consistent view of the document. At the same time, users in different regions should see the updated state of the document. Maintaining consistency is important for users connected to both the same and different zones.
- **Availability**: The service should be available at all times and show robustness against failures.
- **Scalability**: A large number of users should be able to use the service at the same time. They can either view the same document or create new documents.

Resource estimation

Let's make some resource estimations based on the following assumptions:

- We assume that there are 80 million daily active users (DAU).
- The maximum number of users able to edit a document concurrently is 20.
- The size of a textual document is 100 KB.
- Thirty percent of all the documents contain images, whereas only 2% of documents contain videos.

- The collective storage required by images in a document is 800 KB, whereas each video is 3 MB.
- A user creates one document in one day.

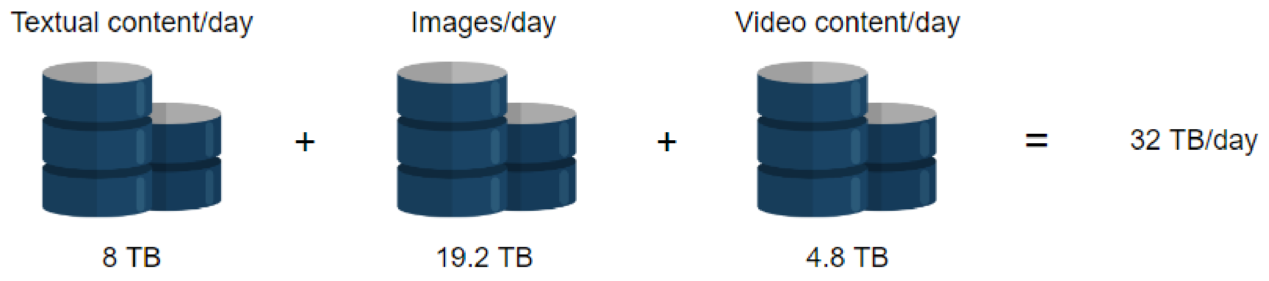
Based on these assumptions, we'll make the following estimations.

- Total number of documents in a day: $80M \times 1 document = 80M$ documents in a day
- Storage for each textual document: $80M \times 100 KBs = 8 TBs$
- Storage required for images for one day:

$$\frac{80M \times 30}{100} \times 800KBs = 19.2\ TBs$$

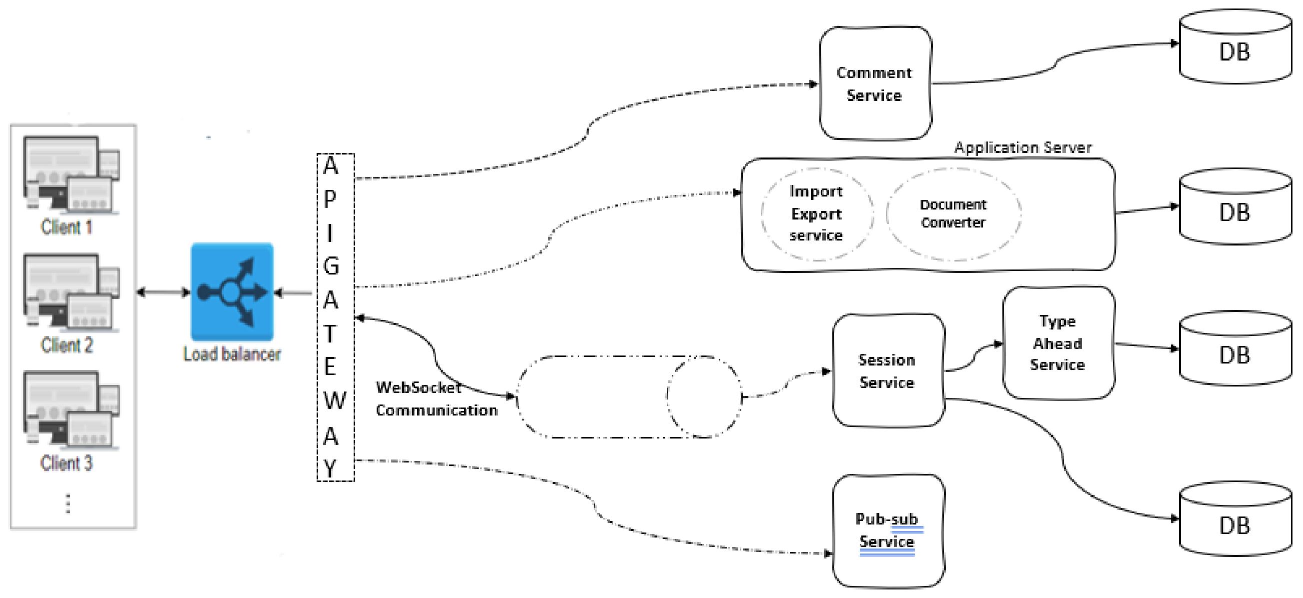
- Storage required for video content for one day: (Two percent of documents contain videos.)

$$\frac{80M \times 2}{100} \times 3MBs = 4.8\ TBs$$



Storage required by online collaborative document editing service per day

High-Level Design



Detailed Component Design

Why should we use WebSockets instead of HTTP methods? Why are WebSockets optimal for this kind of communication?

WebSockets offer us the following characteristics:

- They have a long-lasting connection between clients and servers.
- They enable full-duplex communication. That is, we can simultaneously communicate from client to server and vice versa.
- There's no overhead of HTTP requests or response headers.

The lightweight nature of WebSockets reduces the latency and allows the server to push changes to clients as soon as they are available.

API Gateway :

Is an API management tool that sits between a client and a collection of backend services.

An API Gateway takes all the API calls from clients, then routes them to the appropriate microservice with request routing, composition, and protocol translation.

In this case, clients interact with the API GATEWAY for any operation to be performed. The gateway can connect first Authenticate and permission service whether understanding you have a permit or not in the current documents.

Advantages

- Single entry point & API composition
- Security
- Dynamic Service Discovery
- Service Partition HIdden
- Hiding MS
- Circuit Breaking

Processing queue

Since document editing requires frequently sending small-sized data (usually characters) to the server, it's a good idea to queue this data for periodic batch processing. We'll add characters, images, videos, and comments to the processing queue. Using an HTTP call for sending every minor character is inefficient. Therefore, we'll use WebSockets to reduce overhead and observe live changes to the document by different users.

Data Stores

User Info Storage: For saving users' information and document-related information for imposing privilege restrictions.

Comments Storage: We can use NoSQL for storing user comments for quicker access.

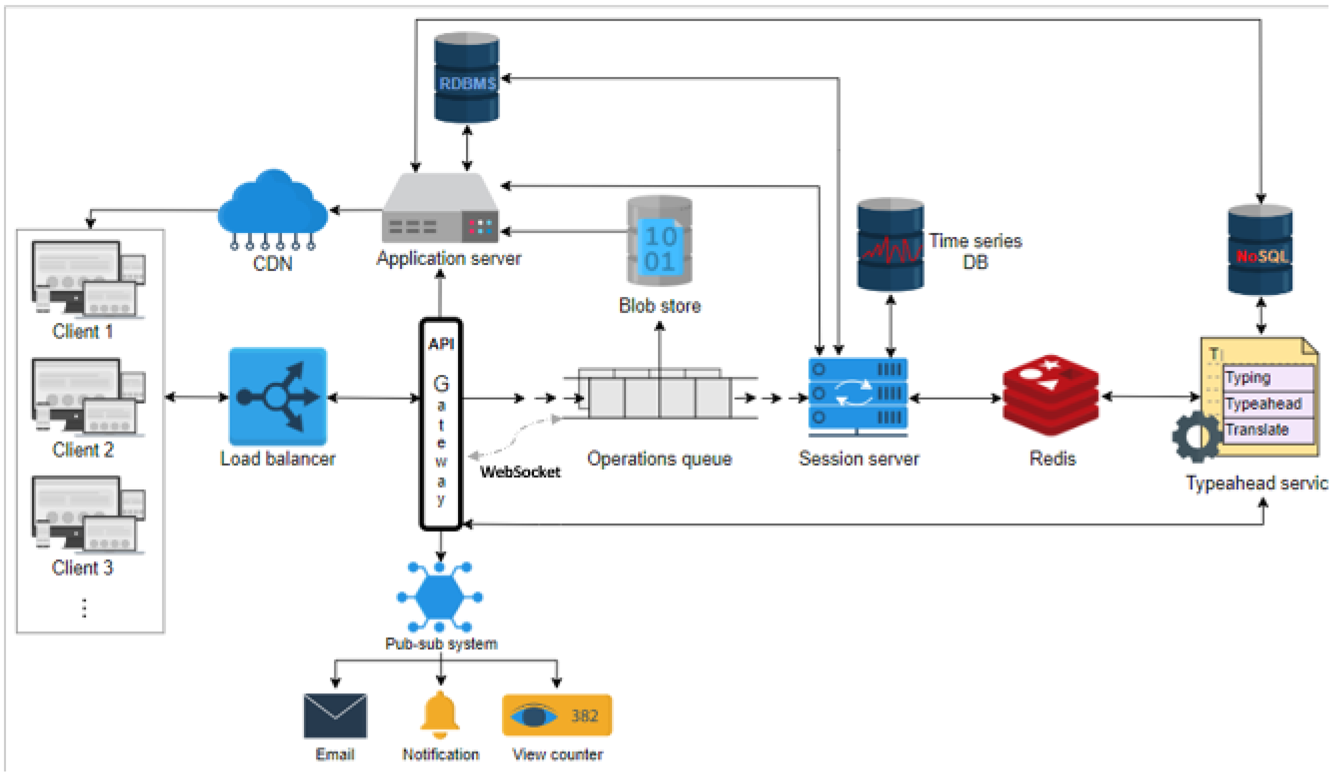
Edit History Storage: we can use a time series database like Influx DB or Elastic Search DB. It keeps on saving a copy of the history of our operations into the Time Series DB. Want to deliver it back to a particular version or if you want to check who edited this particular line or this particular word we can easily get that information Time Series DB

Videos and Images Storage: Blob Storage like AWS S3.

Data Cache: We can use distributed cache like Redis and a CDN to provide end users good performance. We use Redis specifically to store different data structures, including user sessions, features for the typeahead service, and frequently accessed documents. The CDN stores frequently accessed documents and heavy objects, like images and videos.

Application servers: The application servers will run business logic and tasks that generally require computational power. For instance, some documents may be converted from one file type to another (for example, from a PDF to a Word document) or support features like import and export. It's also central to the attribute collection for the recommendation engine.

Other components: Other components include the session servers that maintain the user's session information. We'll manage document access privileges through the session servers. Essentially, there will also be configuration, monitoring, pub-sub, and logging services that will handle tasks like monitoring and electing leaders in case of server failures, queueing tasks like user notifications, and logging debugging information.



Workflow

In the following steps, we'll explain how different requests will get entertained after they reach the API gateway:

- **Collaborative editing and conflict resolution**: Each request gets forwarded to the operations queue. This is where conflicts get resolved between different collaborators of the same document. If there are no conflicts, the data is batched and stored in the time series database via session servers. Data like videos and images get compressed for storage optimization, while characters are processed right away.

- **History**: It's possible to recover different versions of the document with the help of a time series database. Different versions can be compared using DIFF operations that compare the versions and identify the differences to recover older versions of the same document.
- **Asynchronous operations**: Notifications, emails, view counts, and comments are asynchronous operations that can be queued through a pub-sub component like Kafka. The API gateway generates these requests and forwards them to the pub-sub module. Users sharing documents can generate notifications through this process.
- **Suggestions**: Suggestions are in the form of the typeahead service that offers autocomplete suggestions for typically used words and phrases. The typeahead service can also extract attributes and keywords from within the document and provide suggestions to the user. Since the number of words can be high, we'll use a NoSQL database for this purpose. In addition, the most frequently used words and phrases will be stored in a caching system like Redis.
- **Import and export documents**: The application servers perform a number of important tasks, including importing and exporting documents. Application servers also convert documents from one format to another. For example, a `.doc` or `.docx` the document can be converted into `.pdf` or vice versa. Application servers are also responsible for feature extraction for the typeahead service.

Access Control

You can invite collaborators (a person who works jointly on an activity or project; an associate) to Collaborative Document Editor, and we can keep a list of collaborators with relevant permissions such as read-only, owner, and so on.

When someone wants to accomplish something specific, the system asks for his consent. However, for a better access control mechanism, we can use **RBAC (Role-based Access Control)**.

RBAC systems are based on a user's roles and responsibilities. Users aren't given access to systems, roles are assigned to users, and access is granted to roles.

In the role-based access control data model, access can be limited to certain actions such as reading, creating, or editing files.

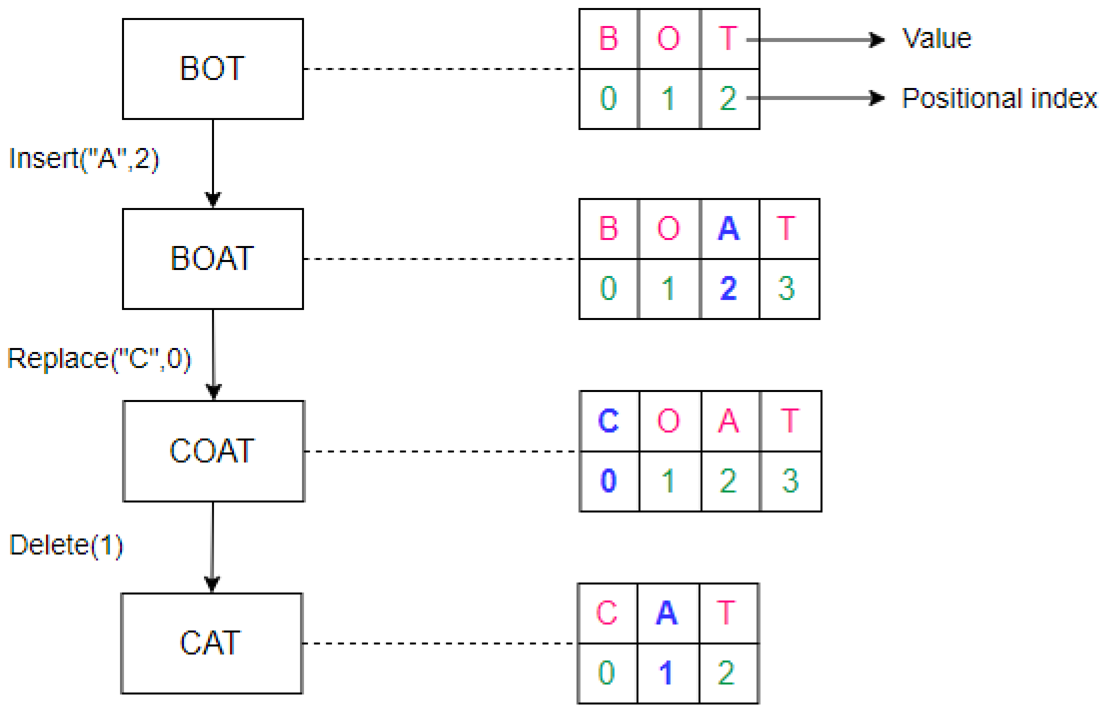
Access can be restricted to particular actions, such as reading, producing, or changing files, based on various variables, including authorization, responsibility, and job expertise.

RBAC-enabled solutions are more capable of safeguarding sensitive data and critical applications.

What is a document editor and the Challenges does it have?

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, an 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

It's pretty challenging to design a collaborative document editing platform.

- One difficulty is devising a system for allowing numerous persons to edit in the same area without conflicting edits.
- The second issue is ensuring that when many modifications occur simultaneously, each one is effectively merged.

The first difficulty is solved by two types of algorithms that are used for collaborative editing.

- **Conflict-free Replicated Data Types (CRDTs)**
- **Operational Transformation (OT)**

and the second problem is solved by the **collaborative protocol.**

Operational Transformation (OT)

It is an event passing mechanism known as ***Operational transformation***, which maintains and synchronizes a consistent state between users working on a shared document.

You'll need code running in two locations to open a collaborative Editor: your browser and our servers. The code that runs in your browser is referred to as a **client**. The client in the document editor processes all of your adjustments before sending them to the server and then processes other editors' changes after receiving them from the server.

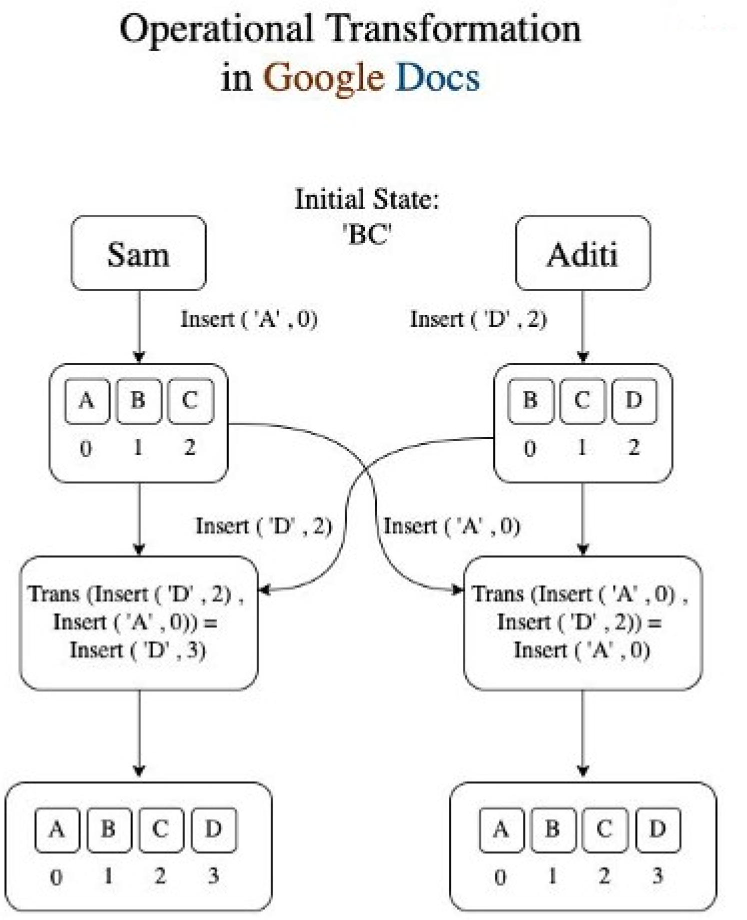
Each client must maintain track of four pieces of information to interact in Editor:

- The most recent revision was delivered from the server to the client.
- Any modifications are done locally that have not yet been communicated to the server.
- Any modifications performed locally are communicated to the server, but the server has not yet acknowledged them.
- The current state of the document is visible to that editor.

The server remembers three things:

- This is a list of all the modifications it has received but has not yet been processed.
- The complete history of all changes that have been made.
- The document's current state as of the latest processed update.

Suppose we have two users Sam and Aditi, simultaneously working on the same document, which has an initial state 'BC'. When Sam and Aditi edit the doc, their respective changes will be sent to the server together with their last revision. More specifically, Aditi's change is inserting the character "D" at position 2, which will be transformed to inserting at the 3rd index as Sam adds "A" in the beginning. Hence the server keeps track of the complete revision history of each document and allows each collaborator to edit the document. The figure below shows how operational transformation works:



Summary of OT

When a data replica changes, the client represents the updates as one or more “operations” (each one encapsulates the change that was made, not the outcome of that change) and shares them with a **central server**. Multiple clients make concurrent changes, so they will each be in different states. They send their operations to the server, which accommodates these different starting states by deciding on the order the outstanding set of operations should be applied (using rules such as last write wins). It then transforms the operations before propagating them to each client to apply to their copy of the data.

Conflict-free Replicated Data Types (CRDTs)

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.

Fulfilling Non Functional REquirements

Consistency

- Gossip protocol to replicate operations of a document within the same data center
- Concurrency techniques like OT and CRDTs
- Usage of time series database for maintaining the order of operations
- Replication between data centers

Latency

- Employing WebSockets
- Asynchronous replication of data
- Choosing an optimal location for document creation and serving
- Using CDNs for serving videos and images

- Using Redis to store different data structures including CRDTs
- Appropriate NoSQL databases for the required functionality

Availability

- Replication of components to avoid SPOFs
- Using multiple WebSocket servers for users that may occasionally disconnect
- Component isolation improves availability
- Implementing disaster recovery protocols like backup, replication to different zones, and global server load balancing
- Using monitoring and configuration services

Scalability

- Different data stores for different purposes enable scalability
- Horizontal sharding of RDBMS
- CDNs capable of handling a large number of requests for big files

References:

https://www.educative.io/courses/grokking-modern-system-design-software-engineers-managers/gkR35nNwJXj