Dropbox System Design

What is Dropbox?

Dropbox is a cloud storage service that allows users to store their data on remote servers. The remote servers store files durably and securely, and these files are accessible anywhere with an internet connection. Usually, these servers are maintained by cloud storage providers and made available to users over a network through internet. Users pay for their cloud data storage.

Requirements of the System

Functional Requirements

- Users should be able to sign up using their email addresses and subscribe to a premium plan. If they don't subscribe, they will get 1 GB of free storage.
- Users should be able to upload and download files from any device.
- Users should be able to share files and folders with other users.
- Users should be able to upload files up to 1 GB.
- The system should support automatic synchronization across the devices.
- The system should support offline editing. Users should be able to add/delete/modify files offline, and once they come online, changes should be synchronized to remote servers and other online devices.
- ACID-ity is required: Atomicity, Consistency, Isolation, and Durability of all file operations should be guaranteed.

Non-functional requirements

- The system should be highly reliable. Any file uploaded should not be lost.
- The system should be highly available. Some of the functional and non-functional requirements before we start to design the system.

Capacity Estimation

Let's do some back-of-the-envelope calculations to estimate the bandwidth and storage required.

Assumptions

- The total number of users = 500 million.
- Total number of daily active users = 100 million
- The average number of files stored by each user = 200
- The average size of each file = 100 KB
- Total number of active connections per minute = 1 million

Storage Estimations

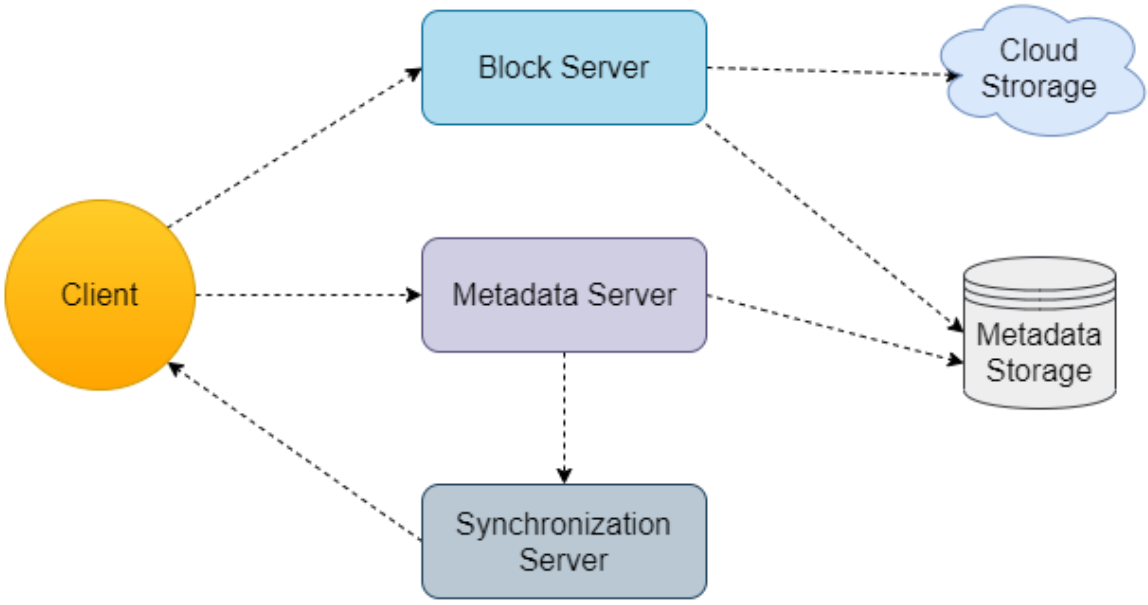
- Total number of files = 500 million * 200 = 100 billion
- Total storage required = 100 billion * 100 KB = 10 PB

High-Level Design of Dropbox

The system needs to deal with a huge volume of read and write data, and their ratio will remain almost the same. So while designing the system, we should focus on optimizing the data exchange between client and server.

At a high level, we need to store files and their metadata information like file name, file size, directory, etc., and who this file is shared with. So, we need servers to help the client upload/download files to cloud storage and some servers to facilitate updating metadata about files and users. We also need some mechanism to notify all clients whenever an update happens to synchronize their files.

As shown in the diagram below, Block servers will work with the clients to upload/download files from cloud storage, and metadata servers will keep metadata of files updated in a SQL or NoSQL database. Synchronization servers will handle the workflow of notifying all clients about different changes for synchronization.



Detailed Design

1. Client Application

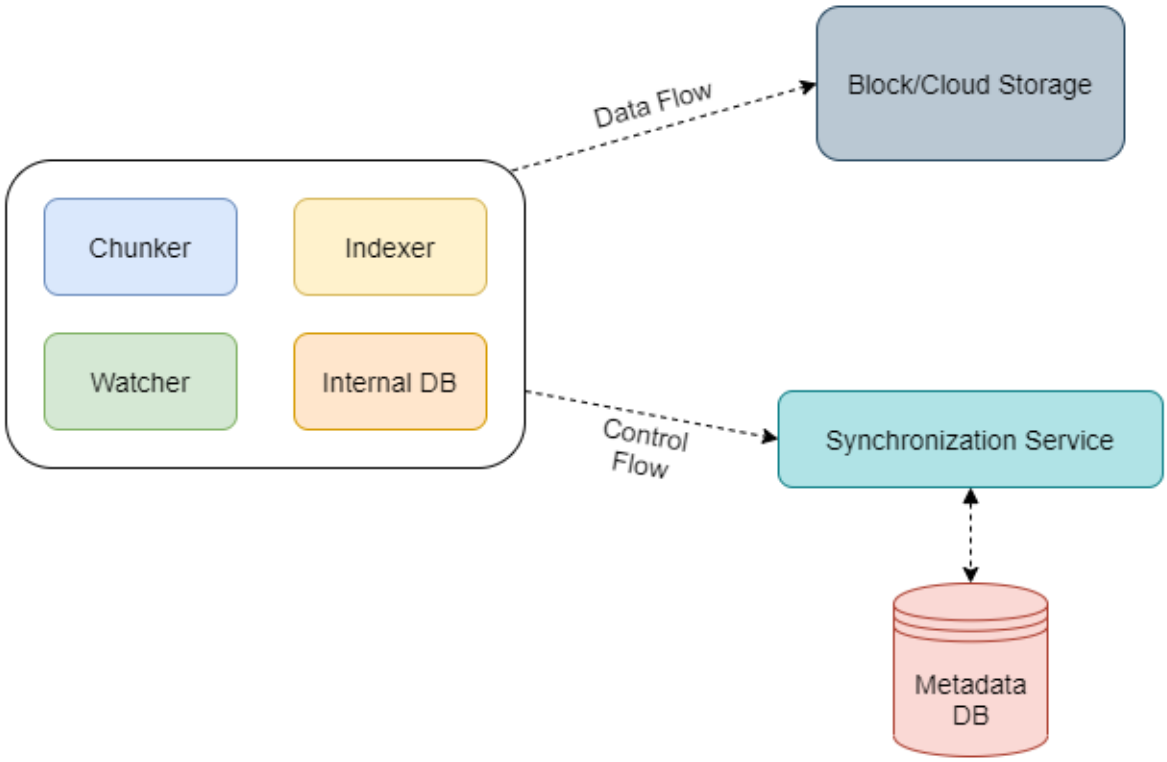
The client application is responsible for monitoring the changes in the workspace. It interacts with the synchronization service to process metadata updates like changes in the file name or contents. It is also responsible for indexing the file, sending the updated chunks to the cloud storage, and retrieving the same if other clients have updated the file.

Major components of the client application:

- **Watcher**: Responsible for monitoring and synchronizing the files and folders for any create, update or delete operation and then informing the indexer component to handle the changes.
- **Chunker**: A chunker's job is to split the files and the incremental changes into chunks of a suitable size ~4MB. These chunks can be later joined in the same order to reconstruct the original file. This component can intelligently sense the changes that are done on the file and transmit only those parts to the storage server.
- **Indexer:** Responsible for listening to the Watcher and maintaining the information about the chunks of files. The indexer also syncs this

data with the Metadata storage server using a synchronization service to successfully store chunks in cloud storage.

- **Internal Database**: Keeps maintains a record of the chunks and associated metadata. It allows for offline operations when the client is not connected to the Dropbox server.



2. Meta Service

The metadata database is responsible for maintaining the version and metadata information about files/chunks, users, and workspaces. It can be a relational database such as MySQL or a NoSQL database service such as DynamoDB. Regardless of the type of the database, the synchronization service should provide a consistent view of the files using a database, especially if more than one user is working with the same file simultaneously. Since NoSQL data stores do not support ACID properties in favor of scalability and performance, we need to incorporate the support for ACID properties programmatically in our synchronization service's logic if we opt for this kind of database. However, using a relational database can simplify the synchronization service

implementation as they natively support ACID properties.

3. Synchronization Service

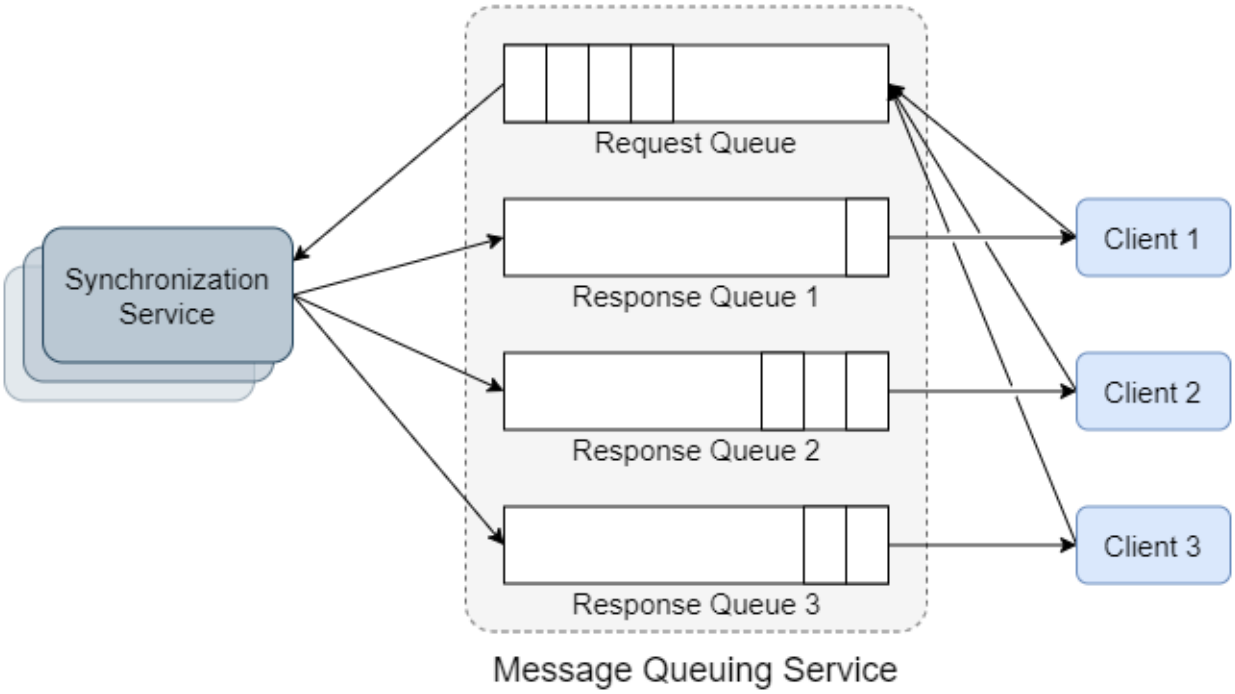
One of the critical components of cloud storage design. It processes all the client's updates on the file and synchronizes those updates across all the devices. It updates the client's local database to be in sync with the metadata stored on the server. All dropbox clients, including desktop, mobile, and web clients, talk to the synchronization service to get updates from the server or push updates to the server. This way, all clients are in sync with the master copy stored in the dropbox cloud. When the client is offline, all updates are stored locally, and when the client becomes online, the synchronization service syncs the data to metadata storage. The same is subsequently pushed to other clients or shared workspace users. It is also possible that two clients have made changes to the same file offline to handle such conflicts. Dropbox handles such scenarios by creating a conflicted copy and saving it with the editor’s username and the save date. Users will be required to resolve that conflict manually.

4. Message Queuing Service

An important part of our reference architecture is a messaging middleware that should handle substantial reads and writes. A scalable message queuing service that supports asynchronous message-based communication between clients and the synchronization service instances best fits our application's requirements.

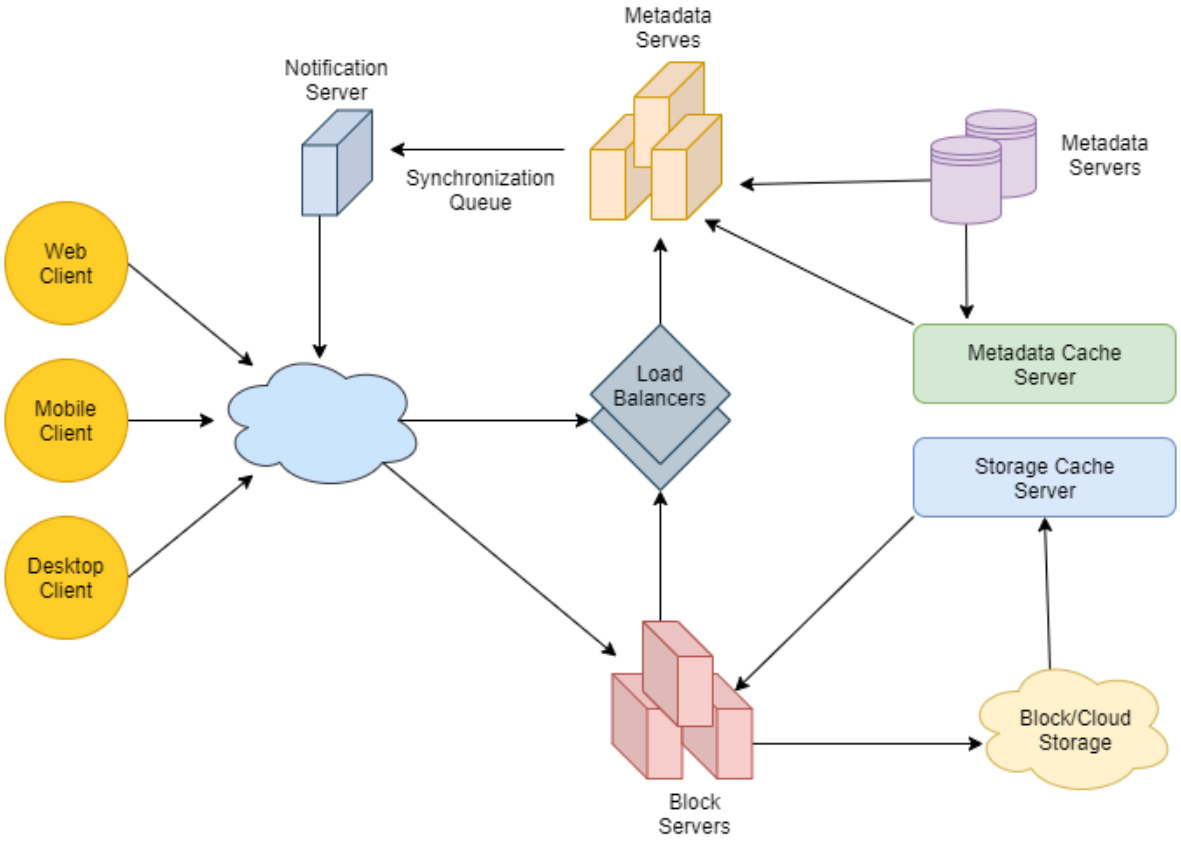
- The message queuing service supports asynchronous and loosely coupled message-based communication between distributed components of the system.
- This service should be of high performance, highly scalable, and persistently store any number of messages in a highly available and reliable queue.
- It also provides load balancing and elasticity for multiple instances of the Synchronization Service.

The figure below illustrates two types of queues that are used in our message queuing service. The **Request Queue** is a global queue that is shared among all clients. Clients' requests to update the metadata database through the synchronization service will be sent to the request queue. The response queues that correspond to individual subscribed clients are responsible for delivering the client's update messages. Since a message will be deleted from the queue once received by a client, we need to create separate **Response Queues** for each client to share an update message that should be sent to multiple subscribed clients.



5. Cloud Storage

Cloud storage stores the chunks of the files uploaded by the users. Clients directly interact with cloud storage to send and receive objects using the cloud provider's API. The separation of the metadata from the object storage enables our reference architecture to use any cloud storage as the back-end data store.



References for further reading

- Dropbox Architecture
- Streaming File Synchronization
- How We've Scaled Dropbox

Thanks to Navtosh for his contribution in creating the first version of this content. If you have any queries/doubts/feedback, please write us at contact@enjoyalgorithms.com. Enjoy learning, Enjoy system design, Enjoy algorithms!