Design Dropbox — A System Design Interview Question

System Design Dropbox....You might have used this file hosting service multiple times to upload and share the files or images but what if somebody asks you to design this gigantic system within just 45 minutes?

You need to tell your approach to design a system like dropbox (within 45 minutes or less) which has hundreds of software engineers working on it for a decade. **System Design dropbox** (or *design google drive or any other file sharing* and upload service) is a quite common question of the system design round. In this blog, let's discuss how to design a website like dropbox or google drive.

How Would You Design Dropbox?

Instead of talking a lot about the web kind of services, we will assume there is a sync client installed in your computer or system and that client is always looking into particular sync folders in which it is always monitoring the changes to the files and uploads it. Also, we won't be talking about how do we build cloud storage. We will use some cloud storage services like Amazon S3 or any other storage services that keep the file in the cloud.

1. Discuss About The Core Features

- Users should be able to upload/download, update and delete the files
- File versioning (History of updates)
- File and folder sync

2. Traffic

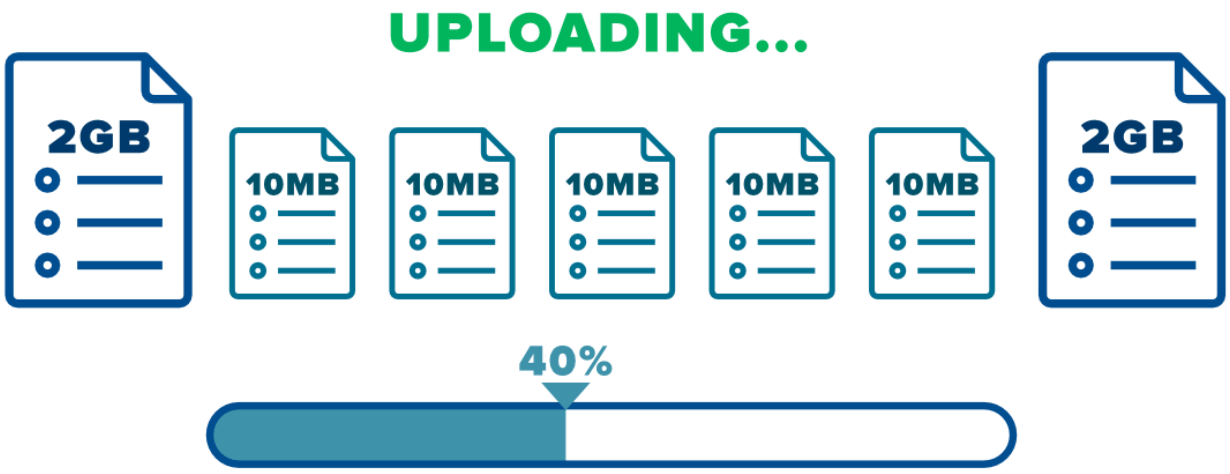
- 12+ million unique users
- 100 million requests per day with lots of reads and write.

3. Discuss The Problem Statement

A lot of people assume designing a dropbox is that all they just need to do is to use some cloud services, upload the file, and download the file whenever they want but that's not how it works. The core problem is "*Where and how to save the files?*".

Suppose you want to share a file that can be of any size (small or big) and you upload it into the cloud. Everything is fine till here but later if you have to make an update in your file then it's not a good idea to edit the file and upload the whole file again and again into the cloud. The reason is...

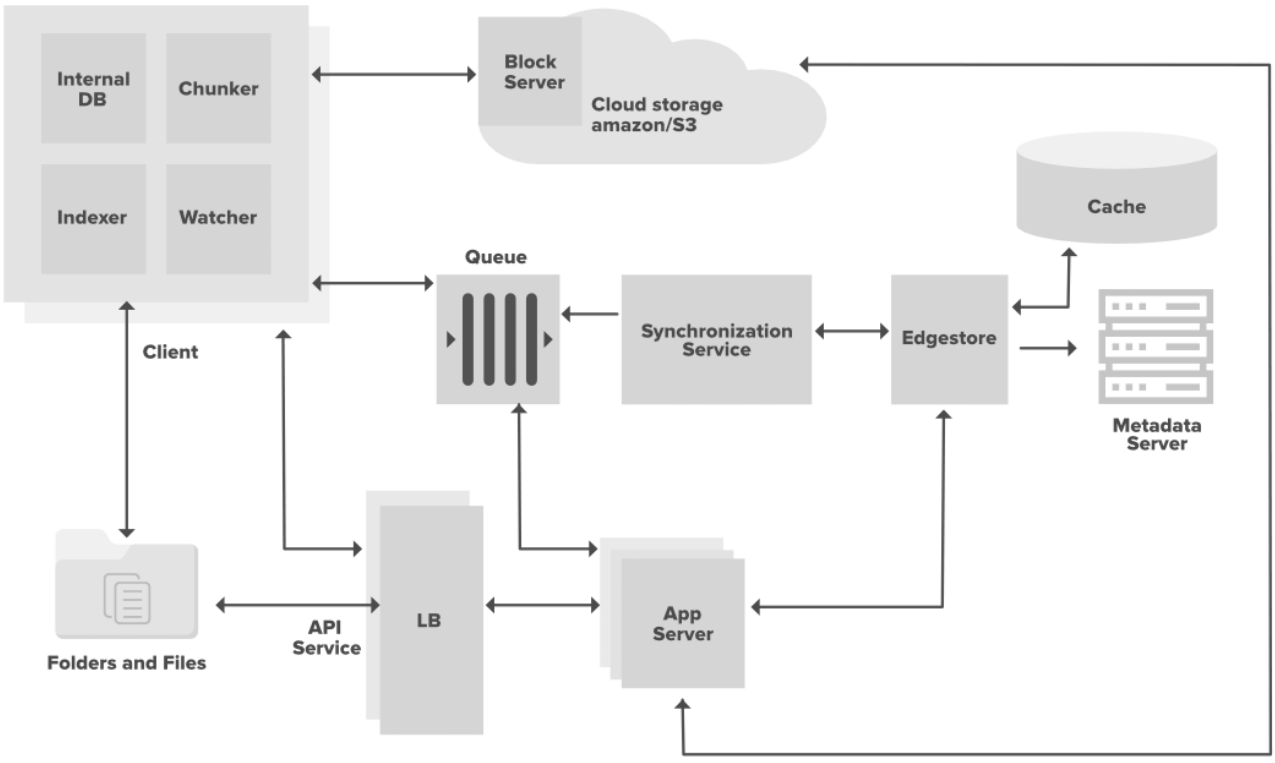
- **More bandwidth and cloud space utilization:** To provide a history of the files you need to keep multiple versions of the files. This requires more bandwidth and more space in the cloud. Even for the small changes in your file, you will have to back up and transfer the whole file into the cloud again and again which is not a good idea.
- **Latency or Concurrency Utilization:** You can't do time optimization as well. It will consume more time to upload a single file as a whole even if you make small changes in your file. It's also not possible to make use of concurrency to upload/download the files using multi threads or multi processes.



We can break the files into multiple chunks to overcome the problem we discussed above. There is no need to upload/download the whole single file after making any changes in the file. You just need to save the chunk which is updated (this will take less memory and time). It will be easier to keep the different versions of the files in various chunks.

We have considered one file which is divided into various chunks. If there are multiple files then we need to know which chunks belong to which file. To keep this information we will create one more file named a **metadata file**. This file contains the indexes of the chunks (chunk names and order information). You need to mention the hash of the chunks (or some reference) in this metadata file and you need to sync this file into the cloud. We can download the metadata file from the cloud whenever we want and we can recreate the file using various chunks.

Now let's talk about the various components for the complete system design solution of the dropbox service.



Let's assume we have a client installed on our computer (an app installed on your computer) and this client has 4 basic components. These basic components are Watcher, Chunker, Indexer, and Internal DB. We have considered only one client but there can be multiple clients belonging to the same user with the same basic components.

- The client is responsible for uploading/downloading the files, identifying the file changes in the sync folder, and handling conflicts due to offline or concurrent updates.

- The client is actively monitoring the folders for all the updates or changes happening in the files.
- To handle file metadata updates (e.g. file name, size, modification date, etc.) this client interacts with the Messaging services and Synchronization Service.
- It also interacts with the remote cloud storage (Amazon S3 or any other cloud services) to store the actual files and to provide folder synchronization.

Discuss The Client Components

- **Watcher** is responsible for monitoring the sync folder for all the activities performed by the user such as create, update or delete files/folders. It gives notification to the indexer and chunker if any action is performed in the files or folders.
- **Chunker** breaks the files into multiple small pieces called chunks and uploads them to the cloud storage with a unique id or hash of these chunks. To recreate the files these chunks can be joined together. For any changes in the files, the chunking algorithm detects the specific chunk which is modified and only saves that specific part/chunk to the cloud storage. It reduces the bandwidth usage, synchronization time, and storage space in the cloud.
- **Indexer** is responsible for updating the internal database when it receives the notification from the watcher (for any action performed in folders/files). It receives the URL of the chunks from the chunker along with the hash and updates the file with modified chunks. Indexer communicates with the Synchronization Service using the Message Queuing Service, once the chunks are successfully submitted to the cloud Storage.
- **Internal database** store all the files and chunks of information, their versions, and their location in the file system.

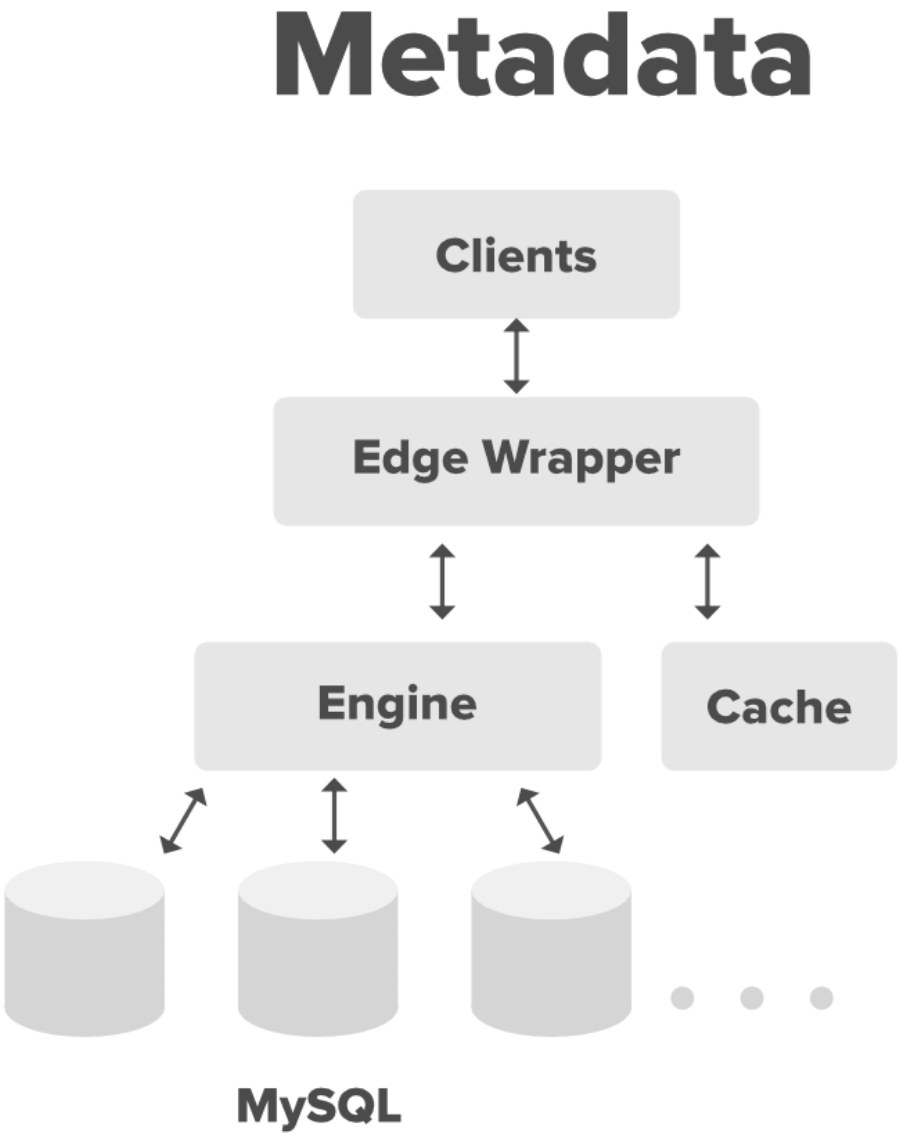
Discuss The Other Components

1. Metadata Database

The metadata database maintains the indexes of the various chunks. The information contains files/chunks names, and their different versions along with the information of users and workspace. You can use RDBMS or NoSQL but make sure that you meet the data

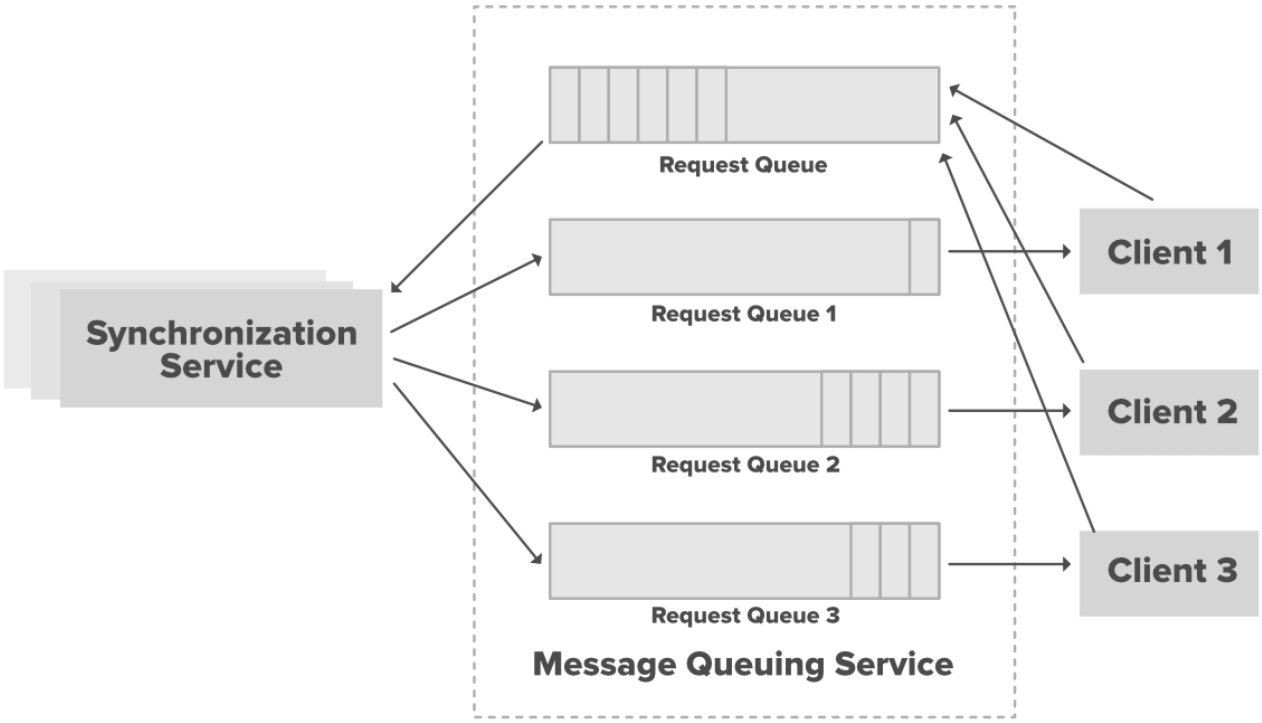
consistency property because multiple clients will be working on the same file. With RDBMS there is no problem with the consistency but with NoSQL, you will get eventual consistency. If you decide to use NoSQL then you need to do different configurations for different databases (For example, Cassandra replication factor gives the consistency level).

Relational databases are difficult to scale so if you're using the MySQL database then you need to use a database sharding technique (or master-slave technique) to scale the application. In database sharding, you need to add multiple MySQL databases but it will be difficult to manage these databases for any update or for any new information that will be added to the databases. To overcome this problem we need to build an edge wrapper around the sharded databases. This edge wrapper provides the ORM and the client can easily use this edge wrapper's ORM to interact with the database (instead of interacting with the databases directly).



2. Message Queuing Service

The messaging service queue will be responsible for the asynchronous communication between the clients and the synchronization service.



Below are the main requirements of the Message Queuing Service.

- Ability to handle lots of reads and writes requests.
- Store lots of messages in a highly available and reliable queue.
- High performance and high scalability.
- Provides load balancing and elasticity for multiple instances of the Synchronization Service.

There will be two types of messaging queues in the service.

- **Request Queue:** This will be a global request queue shared among all the clients. Whenever a client receives any update or changes in the files/folder it sends the request through the request queue. This request is received by the synchronization service to update the metadata database.
- **Response Queue:** There will be an individual response queue corresponding to the individual clients. The synchronization service broadcast the update through this response queue and this response queue will deliver the updated messages to each client and then these clients will update their respective files accordingly. The message

will never be lost even if the client will be disconnected from the internet (the benefit of using the messaging queue service).

- We are creating n number of response queues for n number of clients because the message will be deleted from the queue once it will be received by the client and we need to share the updated message to the various subscribed clients.

3. Synchronization Service

The client communicates with the synchronization services either to receive the latest update from the cloud storage or to send the latest request/updates to the Cloud Storage.

The synchronization service receives the request from the request queue of the messaging services and updates the metadata database with the latest changes. Also, the synchronization service broadcast the latest update to the other clients (if there are multiple clients) through the response queue so that the other client's indexer can fetch back the chunks from the cloud storage and recreate the files with the latest update. It also updates the local database with the information stored in the Metadata Database. If a client is not connected to the internet or offline for some time, it polls the system for new updates as soon as it goes online.

4. Cloud Storage

You can use any cloud storage service like Amazon S3 to store the chunks of the files uploaded by the user. The client communicates with the cloud storage for any action performed in the files/folders using the API provided by the cloud provider.