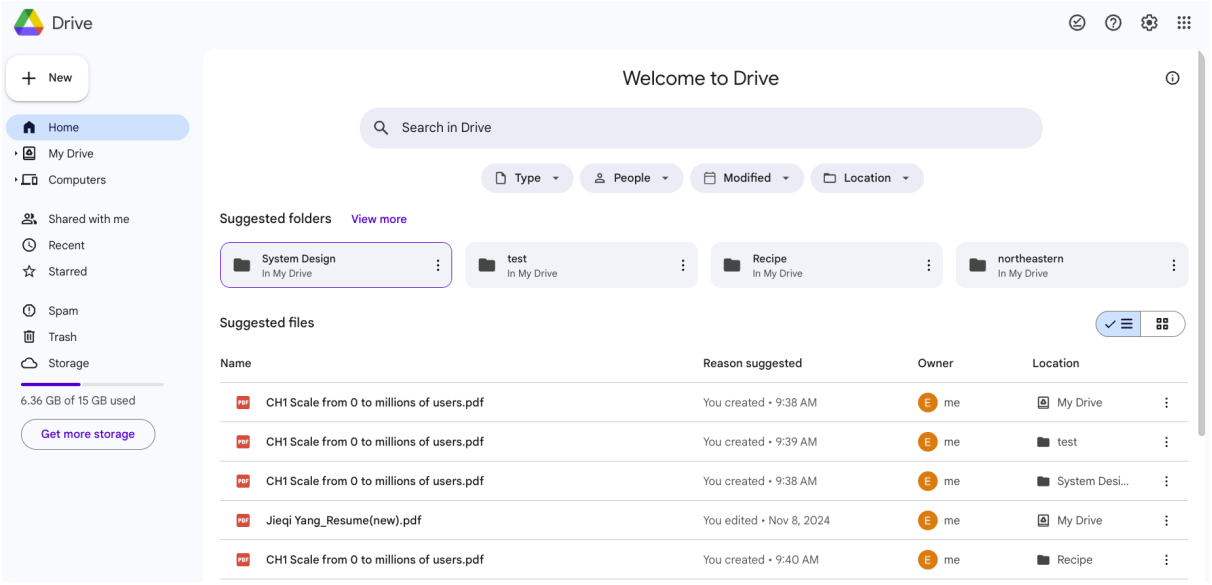
Chapter 15: Design Google Drive

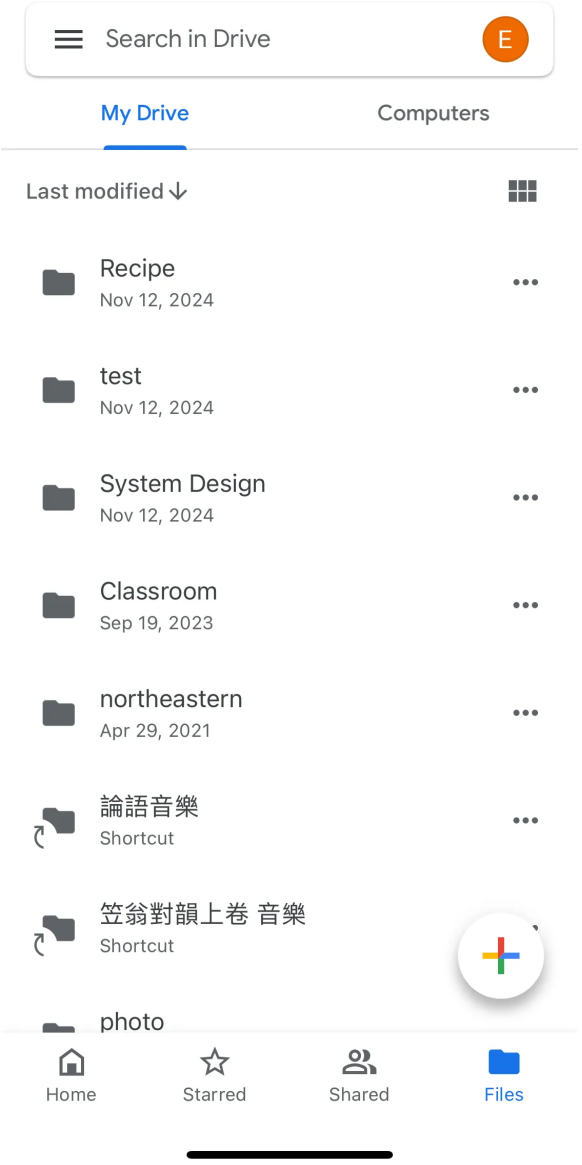
What is Google Drive?

Google Drive is a file storage and synchronization service that helps you store documents, photos, videos, and other files in the cloud. —— Alex Xu

Google Drive is a cloud-based storage service provided by Google that allows users to store, access, and share files online. It integrates with various Google services and applications, making it convenient for personal and collaborative use. —— ChatGPT

- access files from different devices(computer, smartphone, tablet) with internet connection
- allow for file sharing and collaboration with others
- automatic backup and sync
- keep a version history to track changes and recover older file versions
- Offline access allows users to view and edit files without an internet connection, with syncing when reconnected
- Protect files with encryption in transit and at rest


Drive
New
Home
My Drive
Computers
Shared with me
Recent
Starred
Spam
Trash
Storage
6.36 GB of 15 GB used
Get more storage
Welcome to Drive
Search in Drive
Type
People
Modified
Location
Suggested folders
View more
System Design
In My Drive
test
In My Drive
Recipe
In My Drive
northeastern
In My Drive
Suggested files
Name
Reason suggested
Owner
Location
CH1 Scale from 0 to millions of users.pdf
You created • 9:38 AM
me
My Drive
CH1 Scale from 0 to millions of users.pdf
You created • 9:39 AM
me
test
CH1 Scale from 0 to millions of users.pdf
You created • 9:38 AM
me
System Desi...
Jieqi Yang_Resume(new).pdf
You edited • Nov 8, 2024
me
My Drive
CH1 Scale from 0 to millions of users.pdf
You created • 9:40 AM
me
Recipe



How to design Google Drive?

Step 1 - Understand the problem and establish design scope

Candidate: What are the most important features?
Interviewer: Upload and download files, file sync, and notifications.

Candidate: Is this a mobile app, a web app, or both?
Interviewer: Both.

Candidate: What are the supported file formats?
Interviewer: Any file type.

Candidate: Do files need to be encrypted?
Interview: Yes, files in the storage must be encrypted.

Candidate: Is there a file size limit?
Interview: Yes, files must be 10 GB or smaller.

Candidate: How many users does the product have?
Interviewer: 10M DAU.

- In scope
  - Functional
    - Add files
    - Download files
    - Sync files
    - See file revisions
    - Share files with others
    - Send a notification when a file is edited, deleted, or shared with you
  - Non-functional
    - Reliability: Data loss is not acceptable
    - Fast sync to guarantee good user experience
    - Bandwidth usage
    - Scalability： handle high volumes of traffic
    - Availability: Users should still be able to use the system when some servers are offline, slowed down, or have unexpected network errors.
- Out of scope

    - multiple people editing the same document simultaneously is out of this design scope
- **Back of the envelope estimation**(Key aspects to consider and calculate for back-of-envelope-estimation)

  Overall speaking, we could consider the following aspects during back-of-envelope calculations:

    - Traffic Estimation (QPS)
    - Data Storage Requirements
    - Network Bandwidth
    - Latency Goals
    - Read/Write Patterns
    - Cache Size
    - Database Partitioning and Replication
    - Compute Resources

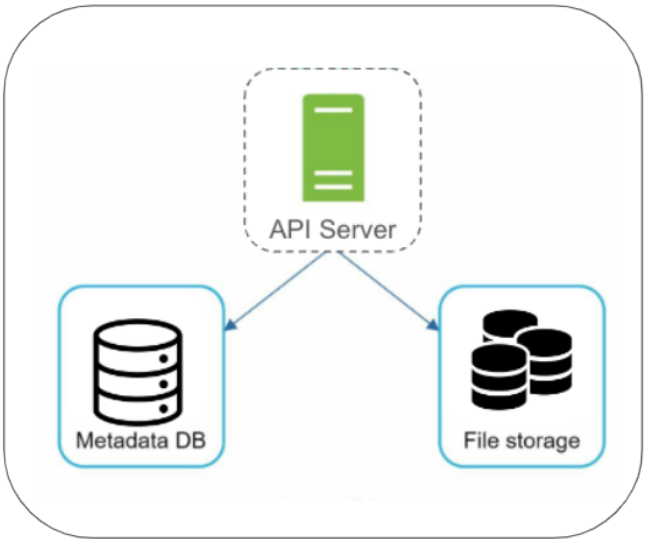
  In this design, we could talk about traffic estimation, data storage requirements, read/write patterns. For instance:

    - Assume the application has 50 million signed up users and 10 million DAU.
    - Users get 10 GB free space.
    - Total space allocated: 50 million * 10 GB = 500 Petabyte
    - Assume users upload 2 files per day. The average file size is 500 KB.
    - 1:1 read to write ratio.
    - QPS for upload API: 10 million * 2 uploads / 24 hours / 3600 seconds = ~ 240
    - Peak QPS = QPS * 2 = 480

Step 2 - Propose high-level design and get buy-in

1. A single server design
    a. Structure
        - A web server to upload and download files.

- A database to keep track of metadata like user data, login info, files info, etc.
- A storage system to store files. We allocate 1TB of storage space to store files.



Single Server

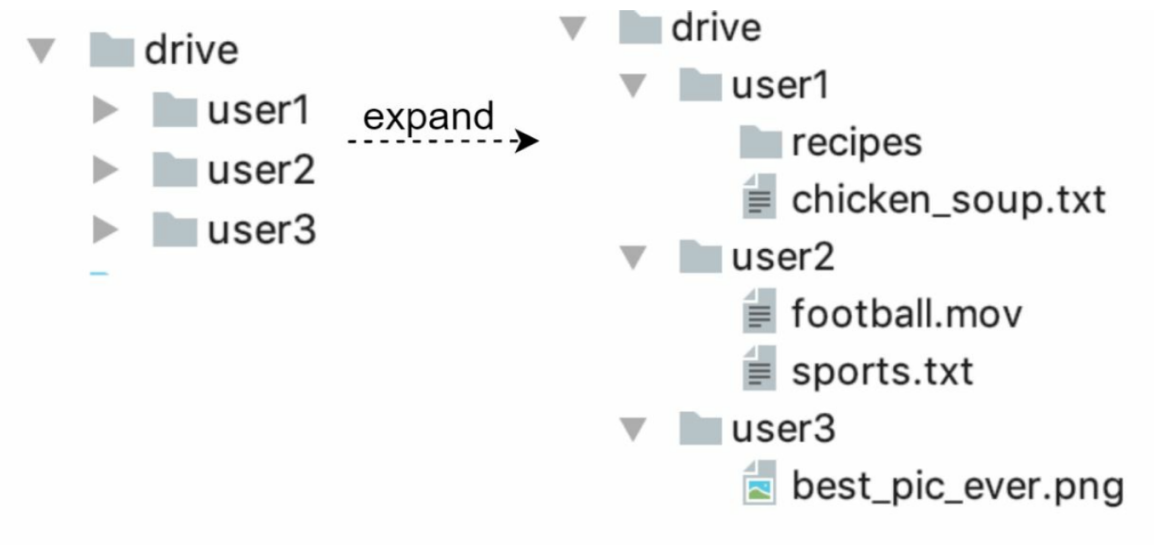
b. Schema

```
-- Users Table
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    password VARCHAR(50) NOT NULL
);

-- Files Table
CREATE TABLE files (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    file_name VARCHAR(255) NOT NULL,
    file_path VARCHAR(255) NOT NULL,
    uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

c. View



d. APIs

- Upload a file
  - Simple upload → small file size.

    $https://example.com/upload$

    Params:

    • file path: local file to be uploaded
  - Resumable upload → big file size.

    $https://api.example.com/files/upload?uploadType=resumable$

    Params:

    • uploadType=resumable

    • file path: local file to be uploaded

    A resumable upload is achieved by the following 3 steps:
    • Send the initial request to retrieve the resumable URL.
    • Upload the data and monitor upload state.
    • If upload is disturbed, resume the upload.

    For details about resumable upload, go to this link.

- Download a file

  API example: $https://api.example.com/files/download$

  Params:

  - file path: file to be downloaded

- Get file revision

  API example: $https://api.example.com/files/list\_revisions$

  Params:

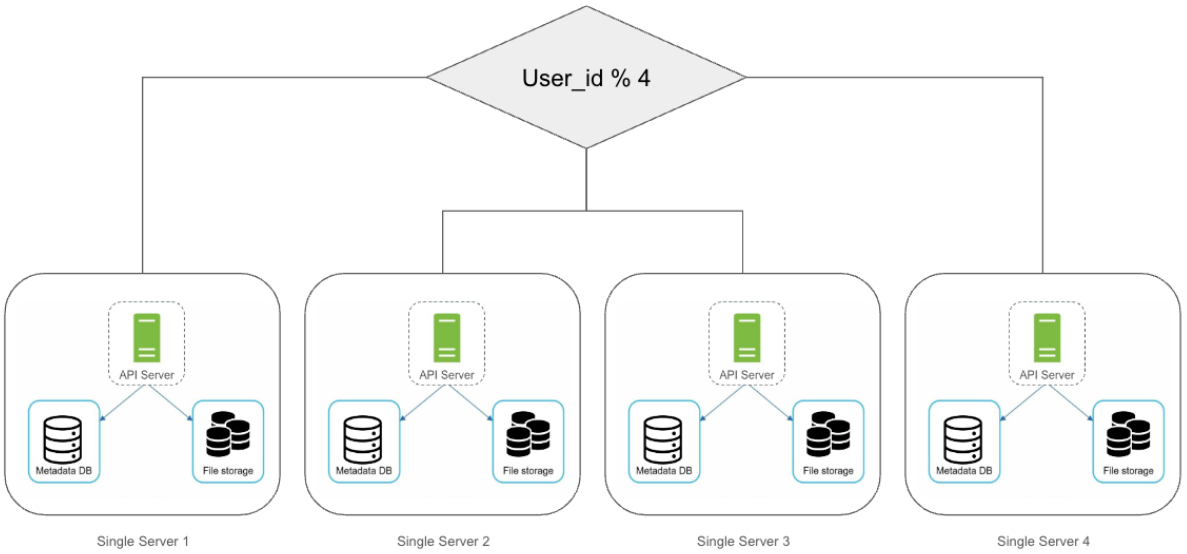
  - path: The path to the file you want to get the revision history.
  - limit: The maximum number of revisions to return.

All the APIs require user authentication and use HTTPS. Secure Sockets Layer (SSL) protects data transfer between the client and backend servers.

2. Move away from single server

Problem of single server solution:

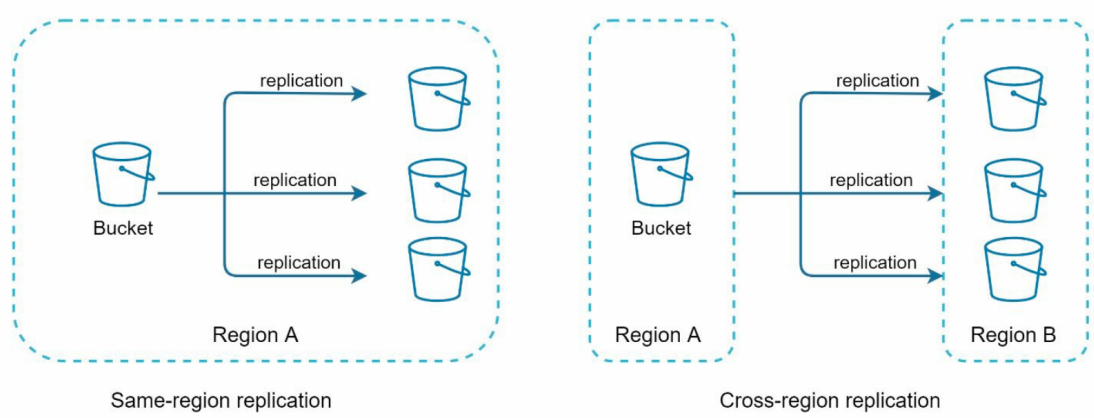
A single server is easily to run out of storage.



Problem of sharding data: potential data losses in case of storage server outage.

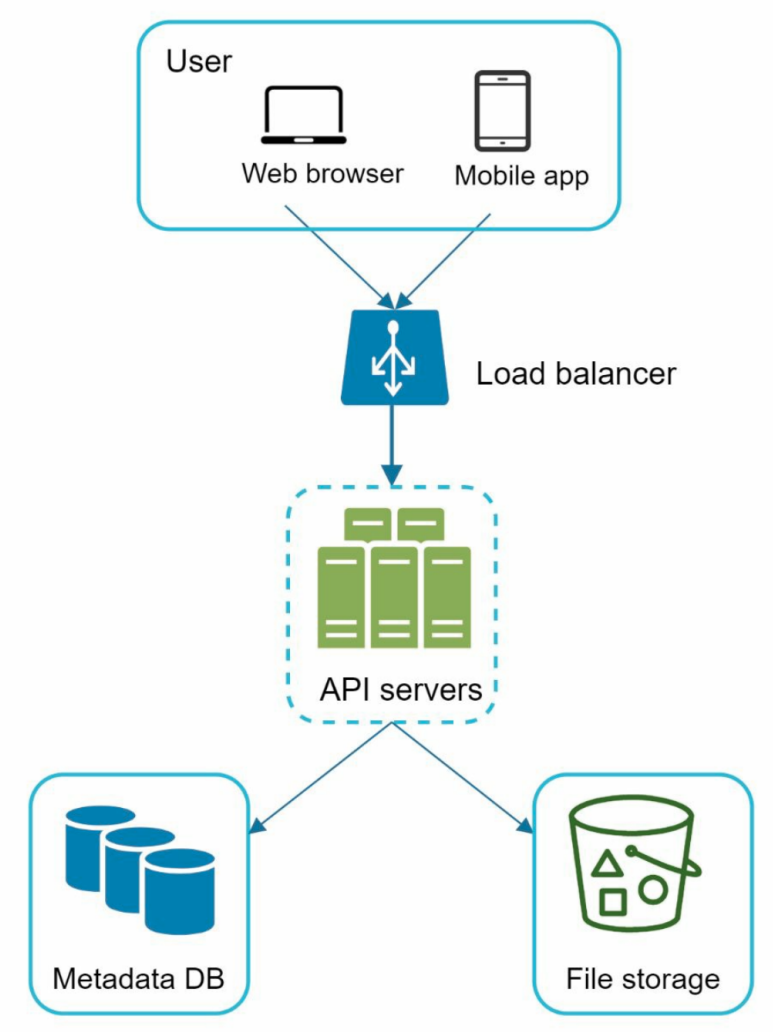
Solution: backup and auto-scaling

Amazon Simple Storage Service(S3): an object storage service that offers industry-leading scalability, data availability, security, and performance.



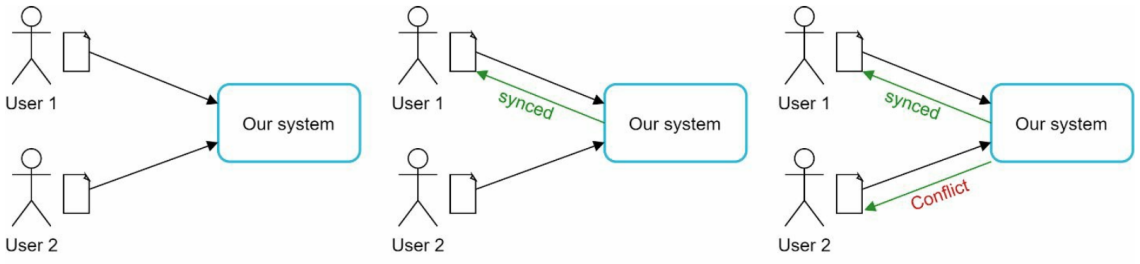
Decoupled structure after improving:

- Add a load balancer to evenly distribute network traffic
- Allow more web servers to handle large traffic
- Move both metadata DB and file storage out of the server to avoid SPOF, and add replicas for both databases



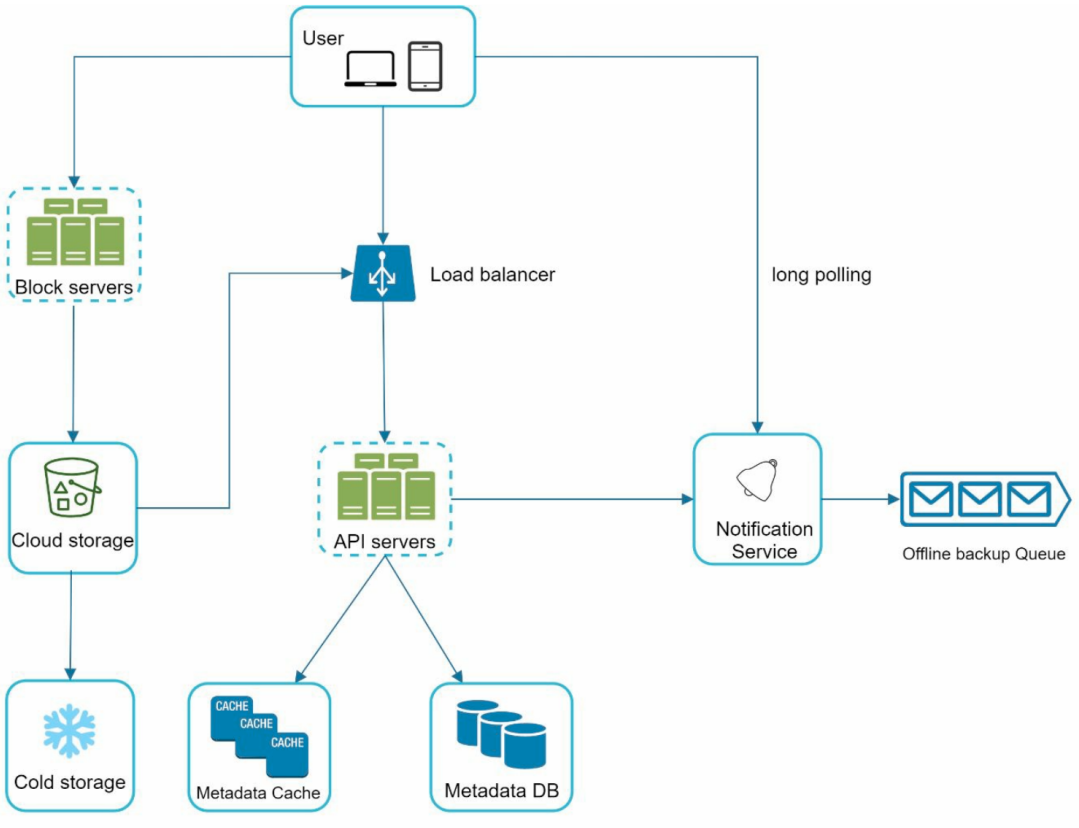
3. Sync conflicts

Strategy: the first version that gets processed wins, and the version that gets processed later receives a conflict.



How to resolve the conflict: the system presents both the local copy of the later user and the latest version from the server, so that the later user has the option to merge both copies or override one version with the other.

4. High-level Design



- Block servers
  - Split a file into blocks
  - Compress blocks
  - Encrypt blocks
  - Upload to cloud storage
  - Reconstruct blocks into a file
- Cloud storage(S3): store blocks in cloud
- Cold storage: store inactive data which are not accessed for a long time(S3 allows set up automatic transitions between different classes of storage by using S3 lifecycle policies).
  - S3 Standard
  - S3 Intellijent-Tiering

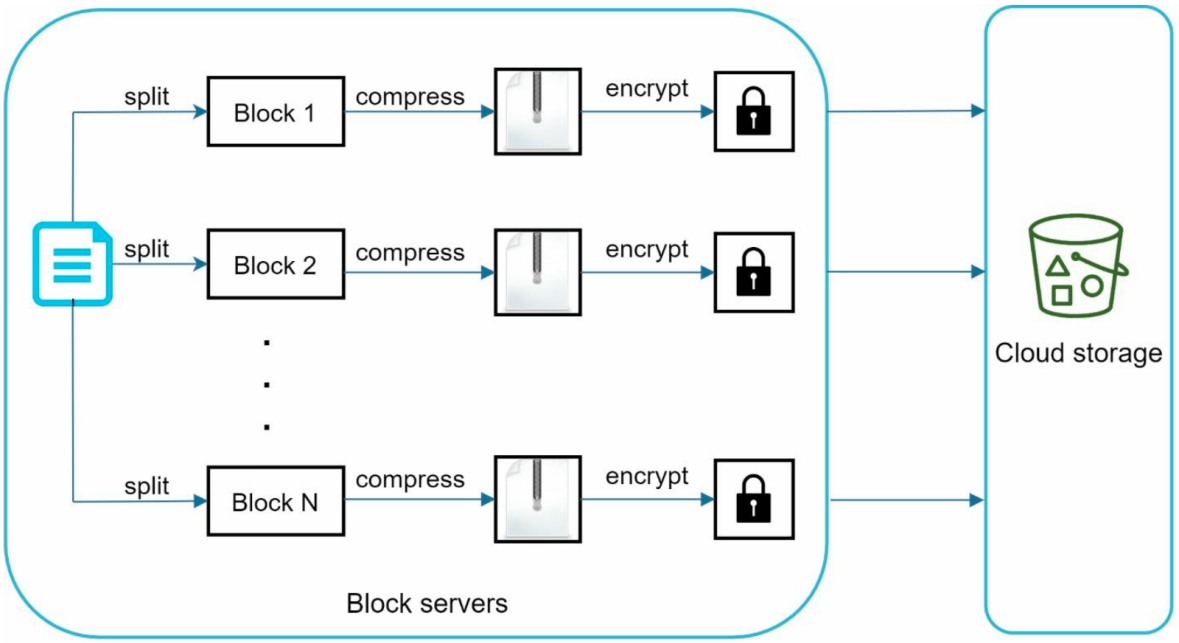
    - S3 Standard-IA(Infrequent Access)
    - S3 One Zone-IA
    - S3 Glacier and S3 Glacier Deep Archive
- Metadata database: Only stores metadata of users, files, blocks, versions, etc.
- Metadata cache: for fast retrieval
- Notification service: notifies relevant clients when a file is added/edited/removed elsewhere
- Offline backup queue: If a client is offline and cannot pull the latest file changes, the offline
  backup queue stores the info so changes will be synced when the client is online.

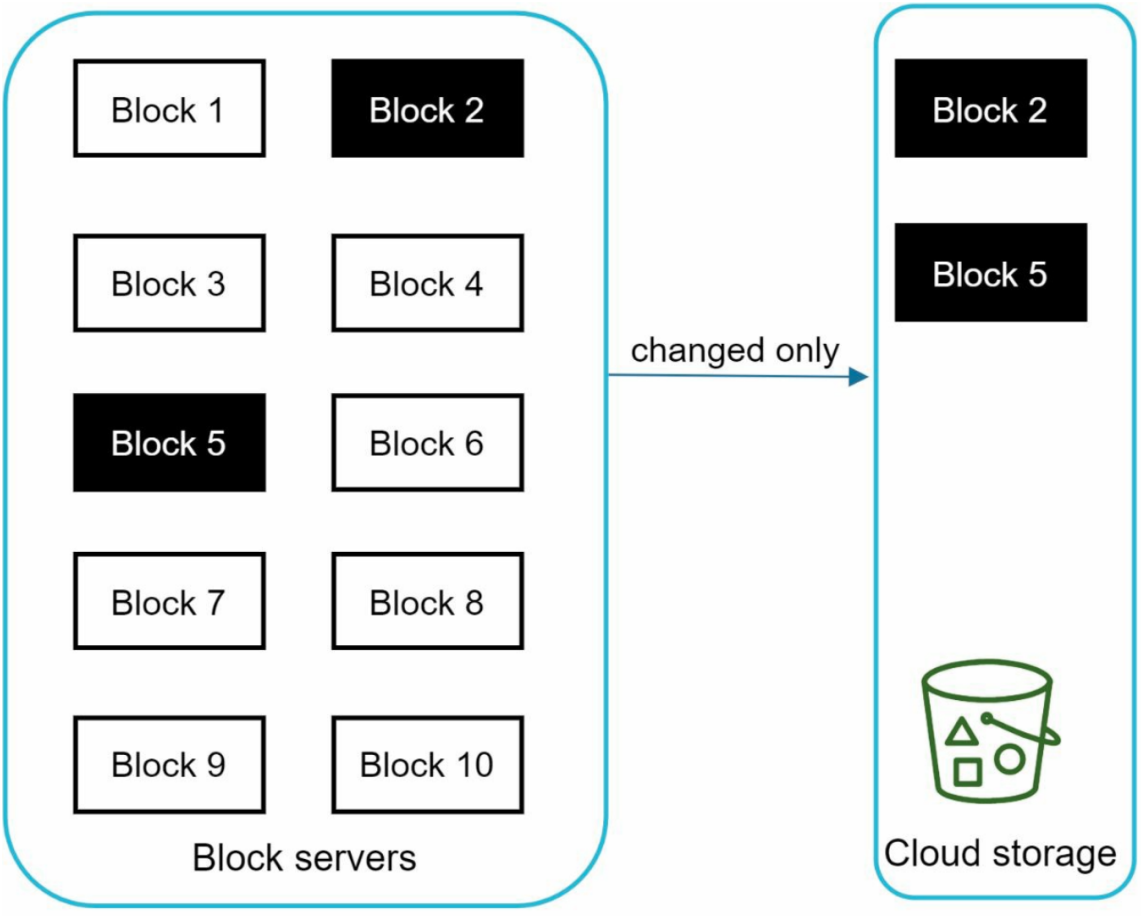
Open Questions:

- The flow on the left is uploading flow, what the downloading and sync flow will be like?
- Are block servers responsible for reconstruct blocks back to a file?
- For the offline backup queue, it stores the notification info, right? What does the updates retrieval process look like?
- Why long polling?

Step 3 - Design deep dive

- Block servers: how to minimize the amount of network traffic while updating files
    - Delta sync: only sync the modified blocks rather than the whole file
    - Compression: applying compression algorithm depending on file types





Open Questions: how to know which block is modified?

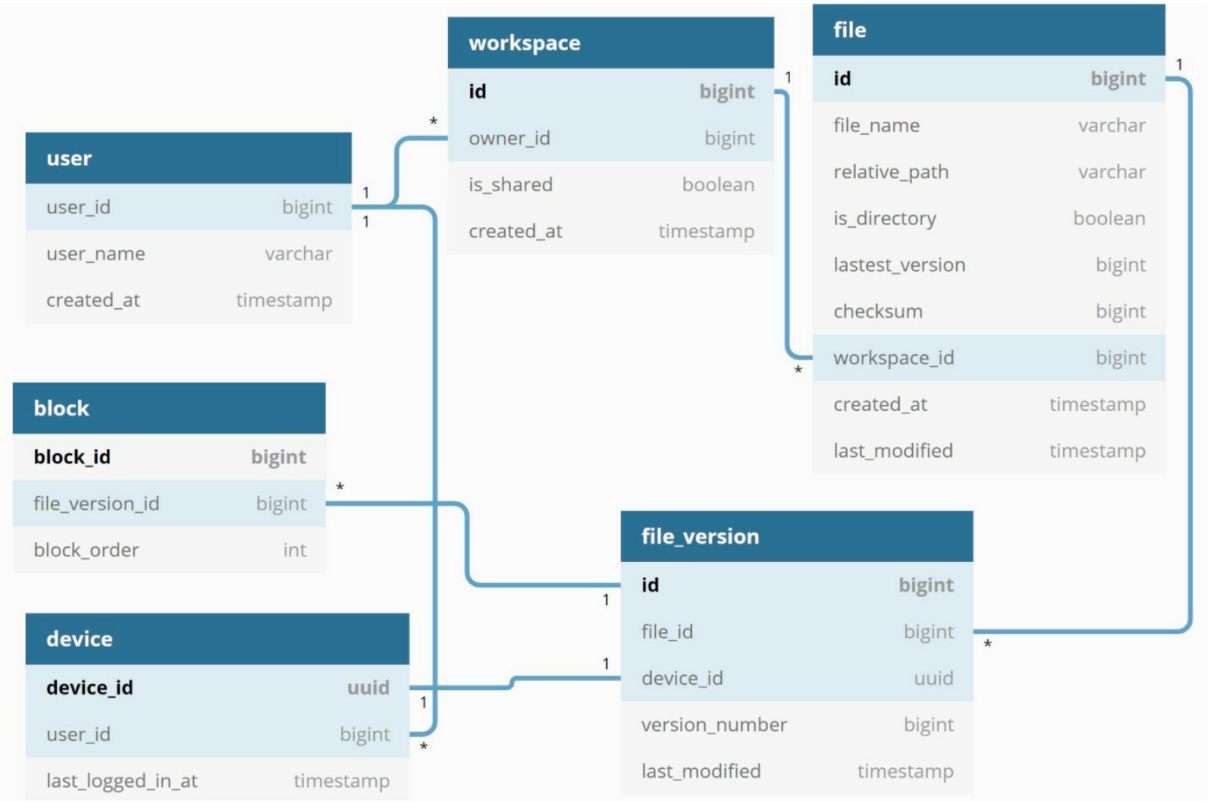
- High consistency requirement

To achieve **strong consistency**, mush ensure:

- Data in cache replicas and the master is consistent.
- Invalidate caches on database write to ensure cache and database hold the same value(Link).

In this design, we choose relational databases because the ACID(Atomicity, Consistency, Isolation, Durability) is natively supported, which makes strong consistency is easy to achieve.

- Metadata database



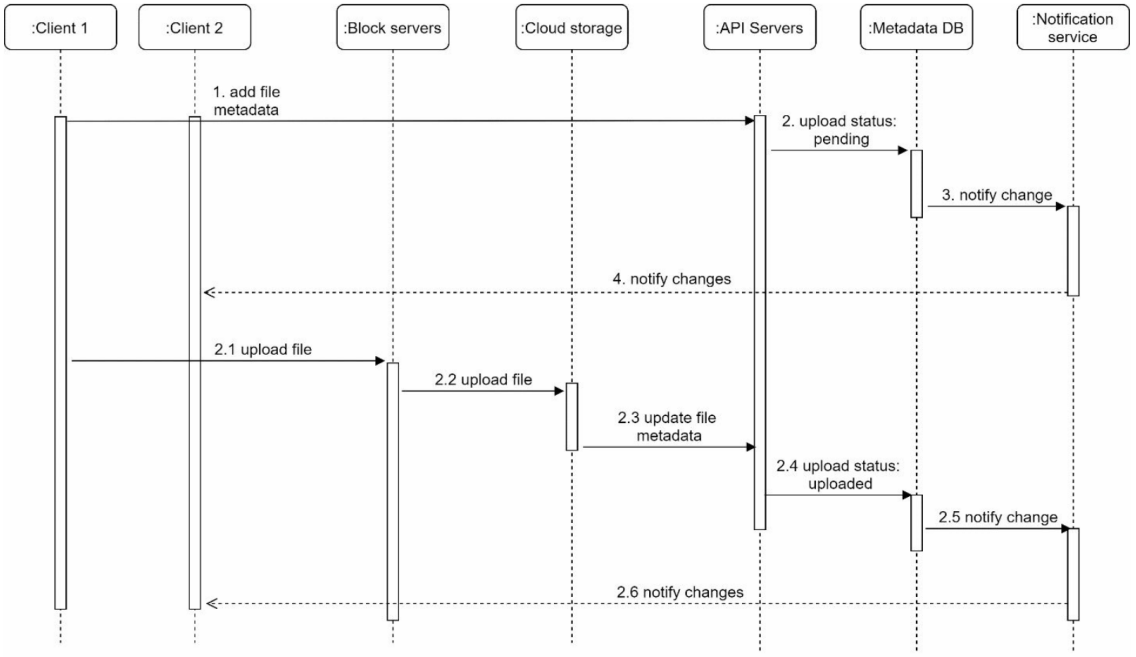
Note:

1. what is Push_id? same as device_id?
2. Rows in table file_version are read-only to keep the integrity of the file revision history
3. In the file table, there is a field named checksum. This may indicate the block comparison is based on checksum(hash).

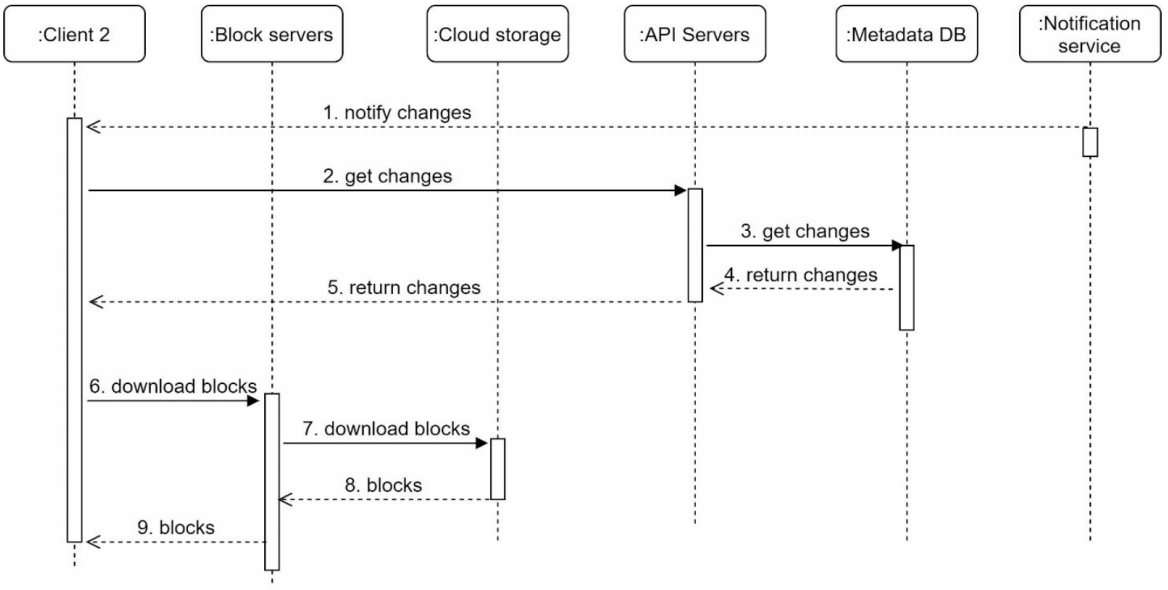
- Upload flow

  Two requests are sent in parallel:

- Add file metadata
- Upload files to cloud storage



- Download flow/Sync flow



Note

Step 2 - 5 are to get metadata on changes, like a key which is used to unlock a door

Open questions:

1. how will be the sync flow like if client is offline? It is confusing that “If client A is offline while a file is changed by another client, data will be saved to the cache.” Shouldn’t the notification of changes saved in offline backup queue?

- Notification service
  - Long polling: single direction, infrequent checking without data burst
  - WebSocket: bi-directional, real-time

  Note: The process of detecting changes with long polling: With long polling, each client establishes a long poll connection to the notification service. If changes to a file are detected, the client will close the long poll connection. Closing the connection means a client must connect to the metadata server to download the latest changes. After a response is received or connection timeout is reached, a client immediately sends a new request to keep the connection open.
- Save storage space

  To reduce storage costs due to frequent backups of all file revision:
  - De-duplicate data blocks: if two blocks have the same hash value, then remove the one of them
  - Intelligent data backup strategy
    - Set a limit: versions older than the limit will be removed
    - Keep valuable versions only: give more weight to recent versions
  - Moving infrequently used data to cold storage
- Failure handling

  • **Load balancer failure**: If a load balancer fails, the secondary would become active and pick up the traffic. Load balancers usually monitor each other using a heartbeat, a periodic signal sent between load balancers. A load balancer is considered as failed if it has not sent a heartbeat for some time.
  •

  Block server failure: If a block server fails, other servers pick up unfinished or pending jobs.
  •

Cloud storage failure: S3 buckets are replicated multiple times in different regions. If files are not available in one region, they can be fetched from different regions.

•

API server failure: It is a stateless service. If an API server fails, the traffic is redirected to other API servers by a load balancer.

•

Metadata cache failure: Metadata cache servers are replicated multiple times. If one node goes down, you can still access other nodes to fetch data. We will bring up a new cache server to replace the failed one.

•

Metadata DB failure.

• Master down: If the master is down, promote one of the slaves to act as a new master and bring up a new slave node.

• Slave down: If a slave is down, you can use another slave for read operations and
bring another database server to replace the failed one.

•

Notification service failure: Every online user keeps a long poll connection with the notification server. Thus, each notification server is connected with many users. According to the Dropbox talk in 2012 [6], over 1 million connections are open per machine. If a server goes down, all the long poll connections are lost so clients must reconnect to a different server. Even though one server can keep many open connections, it cannot reconnect all the lost connections at once. Reconnecting with all the lost clients is a relatively slow process.

•

Offline backup queue failure: Queues are replicated multiple times. If one queue fails, consumers of the queue may need to re-subscribe to the backup queue.

Note:

> The common solution to handle the above different failures are setting replicas for each of them.

For load balancer failure, it could be detected if a load balancer has not sent a heartbeat for some time.

For metadata DB failure, we could discuss two scenarios where either the master DB is down or the slave DB is down.

For the notification service failure, clients must reconnect to a different server, but not at once, which means it is a relatively slow process.

For offline backup queue failure, consumers of the queue may need to re-subscribe to the backup queue.

Step 4 - Wrap up

Other design choices:

1. Upload files directly into cloud storage
    a. Where the chucking, compression and encryption logic happen with S3?
2. Moving online/offline logic to a separate service, say presence service, out of notification servers.

Reference

1. https://nikhilgupta1.medium.com/design-dropbox-google-drive-81cd343571a8
2. https://www.pankajtanwar.in/blog/system-design-how-to-design-google-drive-dropbox-a-cloud-file-storage-service
3. https://medium.com/@murataslan1/designing-a-large-scale-distributed-file-storage-and-sharing-system-like-google-drive-a-a2e7a008c77b