wuyichen24 / system-design-interview Public

Code Issues 4 Pull requests Actions Security Insights

master

system-design-interview / problems / Build_Cloud_File_Storage_System.md

wuyichen24 Update Build_Cloud_File_Storage_System.md History

1 contributor

Build Cloud File Storage System

Real-life examples

- Google Drive
- Microsoft OneDrive
- Dropbox

Requirements clarification

- **Functional requirements**
  - Upload and download: Users can upload and download files.
  - Share: Users can share their files with other users.
  - Synchronization: After updating a file on one device, it should get synchronized on all devices.
- **Non-functional requirements**
  - High availability (Users can access their files whenever and wherever they like).
  - High reliability (Any file uploaded should not be lost).
  - High consistency is desirable (It should be ok for a user doesn’t see a file for a while).

Estimation

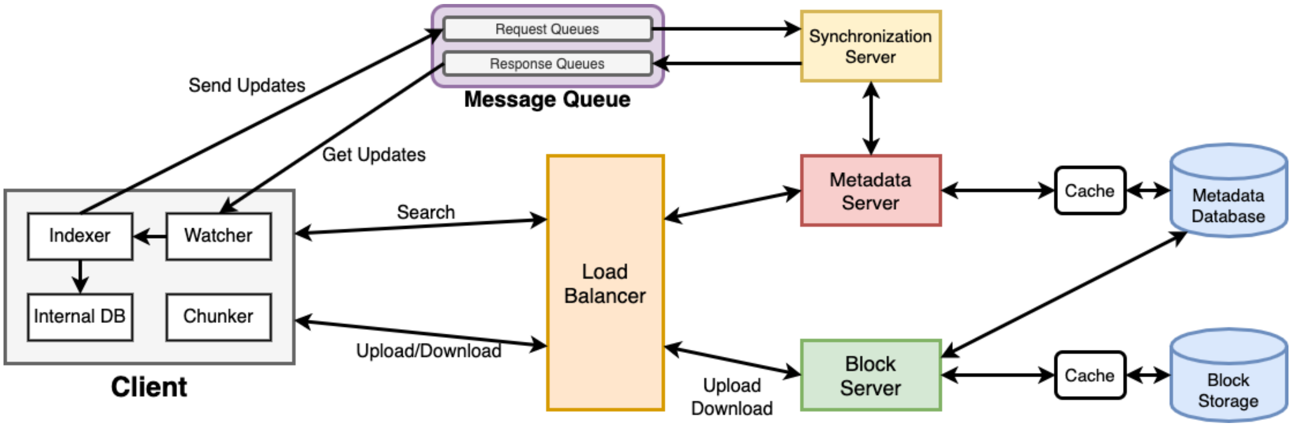
- **Traffic estimation**
  - Our system will have huge read and write volumes.
  - Read-write ratio is expected to be nearly the same.
  - Users
    - 500 million users. (Assumed)
    - 100 million daily active users. (Assumed)

    - Average file size = 100 KB (Assumed)
- **Storage estimation**
  - Total capacity needed = Number of total users x Number of files per user x Average file size = 500 million x 200 x 100 KB = 10 PB
- **Bandwidth estimation**

System interface definition

Data model definition

High-level design



- **Block Server**
  - Handle upload/download file operations.
  - Update the file metadata to the metadata database after uploading files.
- **Block Storage**
  - Store chunks of files uploaded by clients.

- **Metadata Server**
    - Handle metadata-related operation.
- **Metadata Database**
    - Maintain the versioning and metadata information about files/chunks, users, devices and workspaces (sync folders).
- **Synchronization Server**
    - Get file updates from clients.
    - Sychronize file updates to clients.
    - It is designed to transmit less data between clients and the cloud storage to achieve a better response time.
- **Message Queue**
    - A communication middleware between clients and the Synchronization Server for improving efficiency and scalability.
    - Types of queues
        - Request Queues
            - Global queue and all clients will share it.
        - Response Queues
            - Each client will have its own queue for getting updates only for itself.
- **Client**
    - Components
        - Internal DB
            - Keep track of all the files, chunks, their versions, and their location in the file system.
        - Chunker
            - Split the files into smaller chunks (for uploading).
            - Reconstruct a file from its chunks (for downloading).
        - Watcher
            - Monitor the local workspace folders and notify the Indexer of any action performed by the users.
            - Listens to any changes happening on other clients that are broadcasted by the Synchronization Server.
        - Indexer
            - Process the events received from the Watcher and update the internal DB about the chunks of the modified files.
            - Notify the file changes to the Synchronization Server for broadcasting the changes to other clients.