Cloud Storage Benefits:

1. Availability: available on all the devices
2. Reliability: we will not lose them
3. Unlimited storage

Requirements:

Functional Requirements:

1. Upload/View/Delete Pictures or documents
2. Share pictures via link
3. Synchronization of the pictures of data on devices
4. Offline editing and sync when online
5. Should support uploading of 1GB files

Non-Functional Requirements:

1. Available
2. Reliable
3. ACID properties

Design Considerations:

1. Read: Write will be equal in this case
2. Instead of storing the entire data once, we can store the data in chunks
   1. If a chunk upload fails, we can retry only that
   2. We can only change the updated chunks instead of whole file
3. We can store the metadata of the file on the client

Number of users:

500M users

1M Active users

200 photos/ user

High Level System Design:

1. Users specify the file location to upload
   1. Mobile
   2. Laptop
   3. Desktop
   4. Once the picture is uploaded onto the cloud all the other locations need to be synced
2. Metadata
   1. File Size
   2. Location
   3. Filename

Three step process

1. Upload Picture
2. Upload Metadata of the picture
3. Sync the devices

**Upload Files:**

1. We can upload files in chunks
   1. A watcher monitors the changes to a folder
   2. When a new file gets added the watcher can notify the chunker
   3. Partition logic can partition the file into smaller chunks and generate an indexed meta data file which have the information of the chunks and the data of the chunks
   4. We can stream these smaller chunks to an app server to upload it to S3 or some object-based NoSQL database
   5. One we get the reference back we can add the reference to indexed metadata and store it locally on the client
   6. One we store this data this can be published onto to a Request queue to update the metadata on the server
   7. Once the server updates the metadata it can send an event to sync the other clients on the queue

Note: This should be Async as the clients may be offline and we don’t want to connect via WebSocket’s or http to sync the request

**Edit Files:**

1. Once we edit the file
2. We read the metadata from local and we get the index of the chunk that is being edited
3. We store the updated chunk to the app server to S3 as a new version

**Metadata:**

1. Chunks and its hashes
2. Versions for each chunk
3. User
4. Locations
5. Devices

We might need a RDBMS to maintain the consistency. If multiple users are editing the file, we need to maintain the consistency.

NoSQL has the eventual consistency, which might not work in this case, as the chunks can be misplaced. We need to programmatically implement the consistency

**Synchronization:**

1. Sync the other clients with the updates and update metadata on the client
2. When an update request is made on the data, we can check the version the user trying to update the metadata and update accordingly
3. When a client is offline, it will not receive the synchronization updates. We will need to post the message to the queue and send notifications to other clients

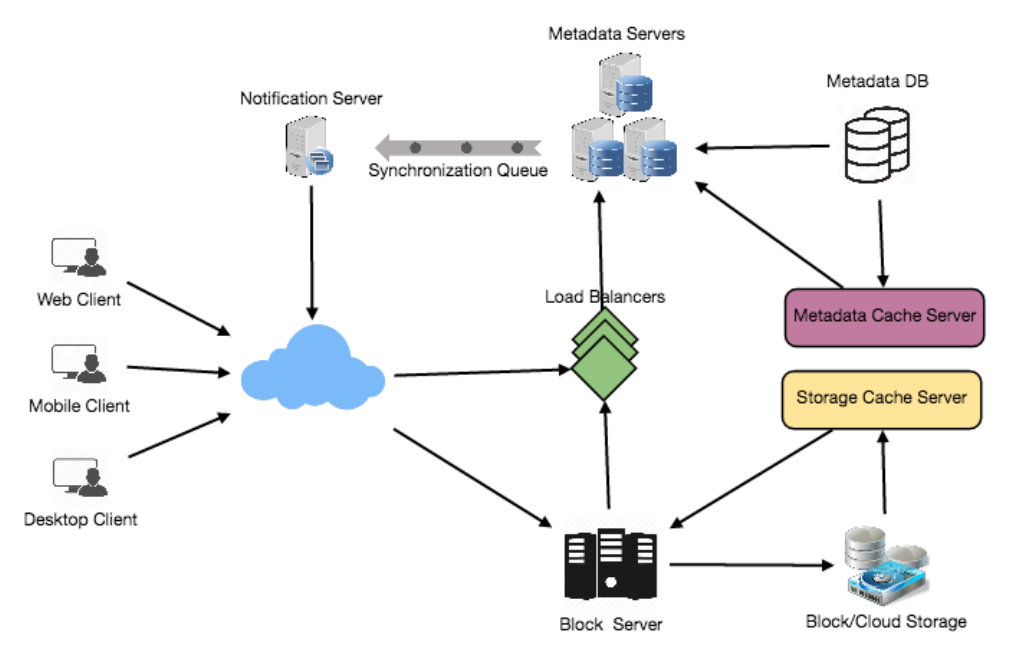
Messaging Queue:

1. One request queue to update the metadata
2. Multiple response queues per client as the message will be deleted if we use a single queue and all the clients subscribe to the same queue.

API Routes and Methods:

1. Upload
   1. File upload data stream
   2. Username
2. Set/Update meta data
   1. Metadata
   2. File location
   3. User
   4. Client
   5. Version

Block Diagram:



Identify and Resolve bottlenecks:

**Data Deduplication:**

1. Helps in efficient usage of storage space
2. Prevents storing the redundant data and prevent from transferring redundant data
   1. Post process deduplication:
      1. First stores the files on the server and then compares with existing chunks and removes the duplicates
      2. Still a waste of bandwidth and storage space for small amount of time
   2. In-line deduplication:
      1. This happens on the client as the user edits the data
      2. The metadata versioned file will have the already existing chunks hash and the changed chunk with its new hash

Data Sharding:

1. Vertical Partitioning:
   1. User information in one database
   2. Metadata related info in another
   3. Joins will be costly
   4. We cannot scale as the information stores in metadata database can grow large
2. Range based partitioning:
   1. Stores the details of file chunks based on the starting letter of the path
   2. Unbalanced servers
3. Hash based partition:
   1. Hash the id of the chunk and store it in a specific partition
   2. This will be uniform distribution

Caching:

1. We can have at the block storage server
2. We can have another at the synchronization server for metadata

Security:

1. Authenticate the user
2. One of the primary concerns users will have while storing their files in the cloud is the privacy and security of their data, especially since in our system users can share their files with other users or even make them public to share it with everyone. To handle this, we will be storing the permissions of each file in our metadata DB to reflect what files are visible or modifiable by any user.