Requirements

PC, Mobile, Tablet all should be in sync(ACID) once we upload from any device.

Upload/Download Files

Support Large Storage

Sharing

Offline Editing

Consistency

Design Considerations

Huge Read and Writes with ratio almost 1:1

Files can be in GBs

Break the files in chunks of say 4MB each

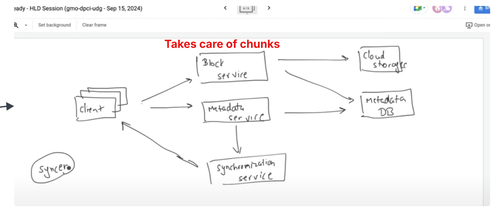
Store file and chunks in metadata

Client can be intelligent to calculate diff in case of failure and upload only affected chunks, not full files.

Capacity Estimation & Services

500 Mn total users , 100 Mn DAU. Assume 200 files are there. Total files = 500Mn*200 = 100Billon.
One file- 100kb, so total = 100*10^12 * 100 *10^3 = 10 PB

ClientService
Upload /Download Service
MetaData Service
Synchronisation Service



Component Design

Client : 1. Upload and Download .
2. Detect File Changes
3. Handle conflicts due to concurrent changes
4. Break files to chunks

MetaData : 1. Having Local metadataDb allows offline editing. i.e. Local mai bhi ek rakhlo so that sync krte time server par call marne kizarurat ni vo locally bhi utha skta hai.
2. Save Round trips

→ How can clients get latest updates?
→ Push → Websockets, HTTP long Polling
→ Pull → HTTP (interval)

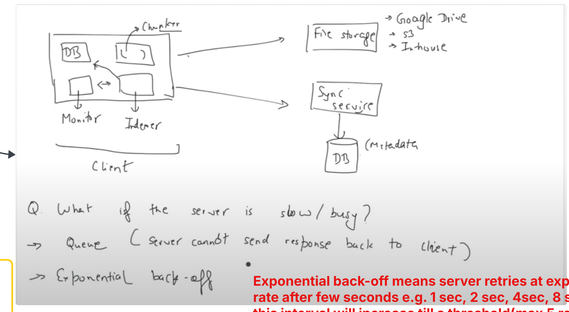
CLIENT service takes of these things

Internal Metadata DB

Chunker (Split Files)

Monitor (monitor changes in workspace)

get event from monitor & update internal DB. Sync internal & remote DB.

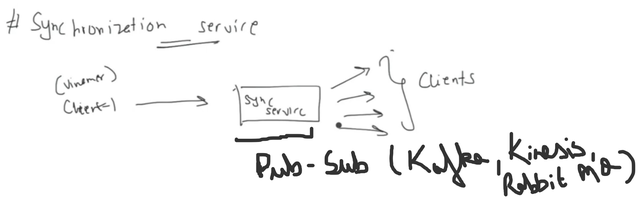


Exponential back-off means server retries at exponential rate after few seconds e.g. 1 sec, 2 sec, 4sec, 8 sec..... like this interval will increase till a threshold(max 5 retries)

Synchronisation Service

Like Whiteboard say, I make some changes and share it to others, so other clients should get notified my whiteboard.

It can also update only the updated part of the file. Queuing system can be used for sync service.



MetaData DB

UserId WorkSpace Devices Filename Size Chunks

SQL will best suite as ACID properties is needed.

Capacity Estimation = 500 M total users=>100M DAU
One user has 200 files so = 100Bn files... Above entities are around 500 bytes so total = 100* 10^9 * 500 = 50 TB
and say DB server capacity is 2 TB => 50/2 = 25 dB partitioning.

Partitioning can be done on userId or FileId(MDS/SHA).. and fileId will make more sense, because syncing doesn't need userId to present. for userId, it can be uneven for some people.

File Permission DB

We can use a FilePermissionDB →
fileId, userId, Permission

WORK-FLOW

⇒ Workflow
→ Client A uploads chunks to cloud storage
→ Client A updates metadata DB
→ Client A receives confirmation
→ Sync service sends notifications to client B & C
→ client B & C request metadata update
→ client B & C download updated chunks.

Data Deduplication (Chunks mai kabkaise bheje)

Post Process - deduplication happens on server side.
Advantage - Client doesn't have to wait.
Disadvantage - Server overhead, Bandwidth wastage i.e. pehle vo chunk bhejega and dekhega ye already uploaded hai ki ni, fir reject krega agar hai toh update ke time, so bhejna to pd hi rha hai at first place so bandwidth waste ho rha h.

Inline Process - deduplication happens on client side.
Calculate chunk hash and compare with server hash . Upload only if hash doesn't match.
Advantage - Client doesn't have to wait.
Disadvantage - Client has to wait,

When Gb's of data is uploaded, bandwidth matters a lot.Then go for Inline else go with Post PRocess for User experience. If Interviewer says it depends on us go for Inline then we can go for Async running and run this in background.

Caching & Load Balancer

Deploy caching for Hot chunks i.e. being read multiple times. Out of Application and Global Cache we can use Global Cache here. and Eviction stratergy can be LRU cache.

From Client-> Block Server (Round Robin LB)
From Client -> Metadata Server (Round Robin LB)
From Metadata Server -> Metadata DB(Consistent Hashing)