Step 1: Requirements

Functional requirements:
- Users can upload and download files from any device
- Support automatic synchronization between devices
- Support large files up to a few GBs
- ACID is required (atomicity, consistency, isolation and durability)

Step 2: Back of Envelope Calculations

- 500M total users, 100M Active daily users
- On average each user connects with 3 devices
- On average each user has 200 files/photos → 100B total files
- Average file size is 100KB → 10PB storage
- 1M active connections per minute

Step 3: System Interface Definition

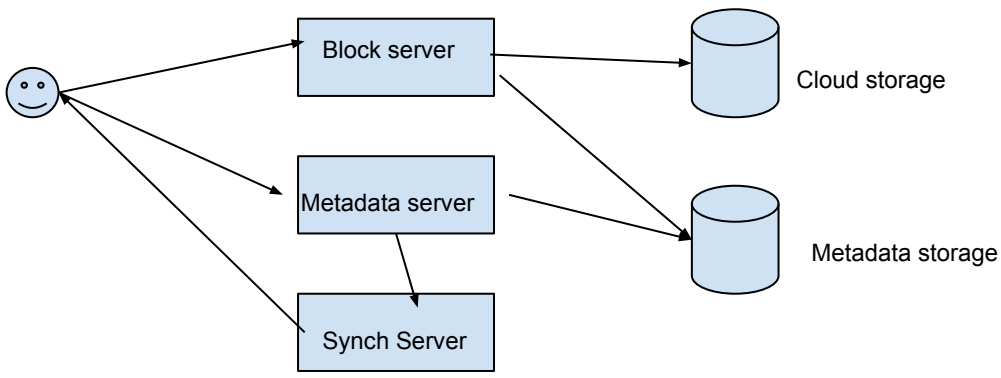
Skipped

Step 4: Define Data Model

Skipped

Step 5: High-Level Design

- Block servers to upload/download files from cloud storage and metadata storage
- Metadata server to keep metadata of files updated in a database
- Synchronization servers to handle notifying clients about changes for synchronization



Step 6: Detailed Design

Client
- Client app monitors workspace folder on user's machine and syncs files/folders with cloud storage
- Only transfer chunks of files that are modified not entire file
- Assume we divide each file into 4MN chunks, then we can calculate optimal chunk size based on:
  a. IOPS for cloud storage
  b. Network bandwidth
  c. Average file size in storage
- Metadata should contain record of each file and chunks that make it up
- Use HTTP long polling so server holds requests open and waits for new info to become available
  a. Once info available, server sends response to client; client can immediately reissue request

Metadata Database
- Maintains versioning and metadata about file/chunks, users and workspaces
- To guarantee ACID-compliance either:
  - Use SQL
  - Use NoSQL and guarantee ACID programmatically via logic in synchronization service

Synchronization Service
- Processes file updates made by clients and applies these changes to other subscribed clients
- Clients poll this service to check for updates
- Synch service checks with metadata database for consistency and proceeds with update
- Notification sent to all subscribed users or devices to report file update
- Scalable message queueing between synch service and clients

Cloud/Block Storage
- Stores chunks of files uploaded by users
- Clients directly interact with storage to send and receive objects from it

Metadata Partitioning
- Vertical partitioning: store tables related to one feature on one server
- Range-based partitioning: store files/chunks in separate partitions based on filepath (may lead to unbalancer servers)
- Hash-based partitioning: based on file ID

Caching
- Cache for block storage to handle hot files/chunks

Load Balancing
- Between client and block servers
- Between client and metadata servers
- Round robin probably okay or we can add additional logic based on expected load of requests