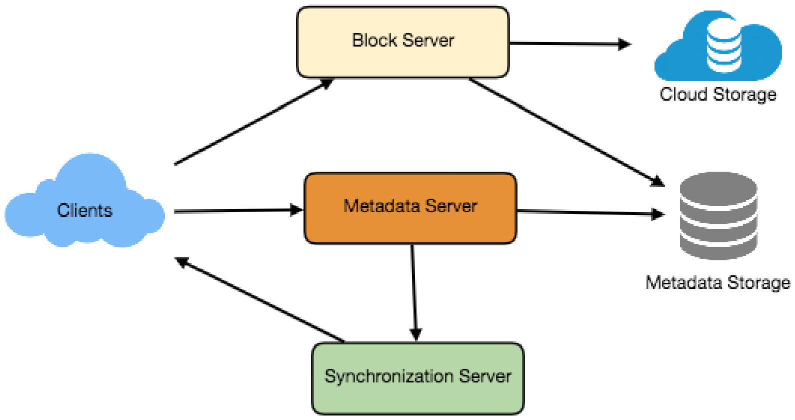
I. Designing Dropbox

1. Why cloud storage?
   a. availability : Must be able to access files from any device, anytime, anywhere
   b. reliability and durability : Uploaded files should not be lost
   c. scalability : Unlimited storage
2. requirements and goals of the system
   a. Upload / download / share files from any device
   b. Automatic synchronization between devices
   c. Large file support (up to 1GB)
   d. ACID-ity support-ensures that transactions are performed safely
      i. atomicity: all or nothing
      ii. consistency: integer cannot be a string
      iii. isolation: No other work can intervene between transactions
      iv. durability (Delay): After deposit, the deposit history should not disappear
   e. Offline editing: Add / delete / modify files offline (synchronize later when online)
   f. File version management (data snapshot support)
3. design considerations
   a. Heavy usage: read / write huge number of files
   b. read/write ratio : 1 : 1
   c. Within the system, files are stored as chunks (ex) 4mb
      i. Transmission failure / update Retransmit only one chunk
      ii. Eliminate duplicate checks: save bandwidth and storage space
      iii. Saving copies of metadata on the client-saving data exchange
      iv. If the change is very small (it's too much to send a chunk): upload diff only

4. capacity estimations and constraints
    a. See spread sheet
5. high level design (11.jpg)



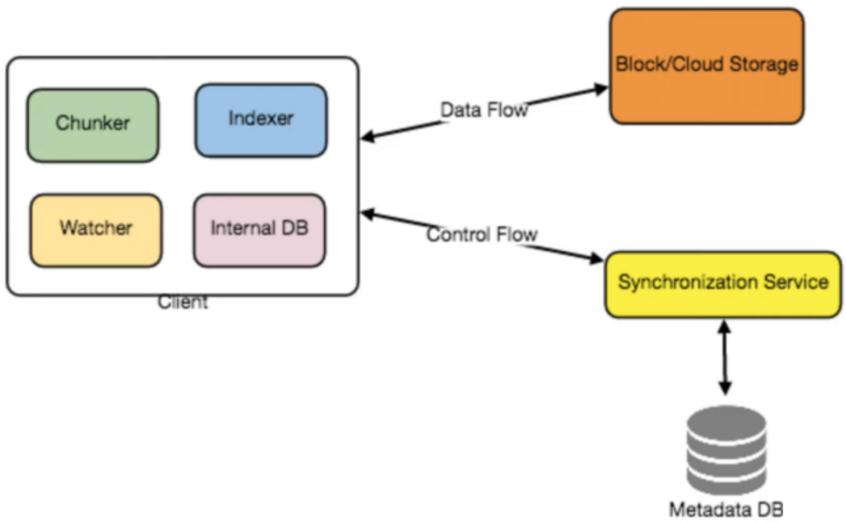
    a. device specific folder-> cloud and sync
    b. Automatic synchronization between devices: Changes made in one device are automatically corrected in all other devices
    c. Save file, metadata, and file sharers-A file upload / download server (block server) and a metadata server are required, and a mechanism for notifying file sync from all clients when an update occurs
    d. component
        i. block server : file upload/download
        ii. metadata server: keep the metadata on the client up to date
        iii. sync server: sync all clients
        iv. storage
            - cloud storage : object storage (ex) amazon s3
            - metadata storage : RDBMS (ex) MySQL, MS-SQL, Oracle
6. component design
    a. client
        i. client application
            - Synced folder monitoring
            - remote cloud storage with sync
            - file upload/download/update

ii. file transfer
   - Divide the file into chunks-only update / transfer failed chunks
   - chunk size: Optimizes space utilization and IOPS (Input Output operations Per Second)
   - Information such as chunk size, network bandwidth, and average storage file size is stored in metadata

iii. Does the client need to keep a copy of the metadata?
   - Offline update can be performed
   - Save round-trip time to update remote metadata

iv. What are the ways clients can check for updates?
   - Client periodically checks
     - ✓ Reflection (server-> client) delay occurs
     - ✓ Server mostly returns empty response: waste of bandwidth and increase of server usage
   - http long polling
     - ✓ Server keeps requests open
     - ✓ Respond to clients immediately if new information is available

v. client component (12.jpg)



- internal metadata DB
  - ✓ Track all files, chunks, versions and locations in the file system
- chunker
  - ✓ File-> divide by chunk
  - ✓ chunk-> file reconstruction
  - ✓ chunk algorithm: Detect only the modified part of the file-> Send only the relevant part to the cloud storage-Save bandwidth / sync time
- watcher
  - ✓ Local workspace (sync folder) monitoring
  - ✓ User notifies indexer when file / folder is created / deleted / updated
- indexer
  - ✓ Event handling received from watcher
  - ✓ Save updated file chunk information in metadata DB
  - ✓ Upload / download the chunks to cloud storage
  - ✓ After communicating with the remote sync service, update the metadata DB after broadcasting the update to other client devices (ipad, mobile, pc, etc)
- How is the client handled when the server is slow?
  - ✓ client increases retry interval exponentially
- In case of mobile device (using wireless internet), do I need to sync server changes immediately?
  - ✓ no. Only upon user request (to save bandwidth and device storage)

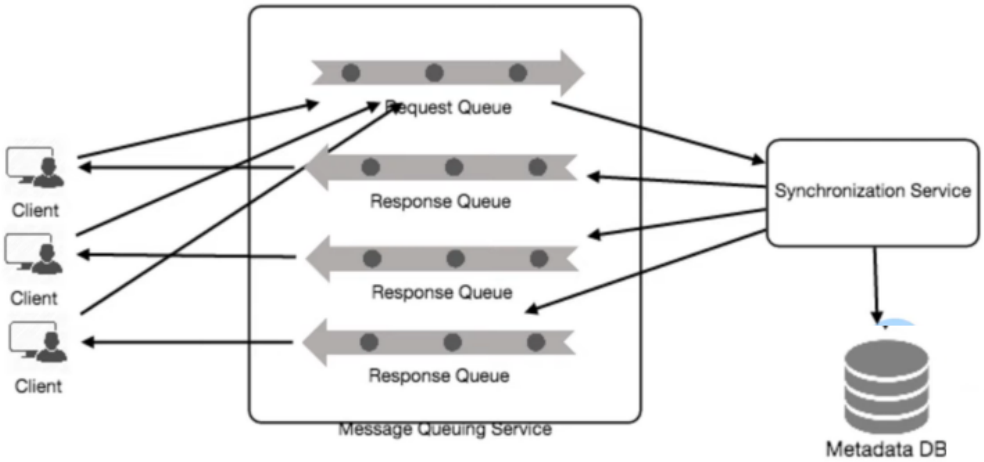
b. metadata database
    i. Maintain version and metadata for file / chunk, client and workspace (sync folder)
    ii. When implementing NoSQL (ACID-ity is not supported for scalability and performance), ACID-ity must be implemented programmatically.
    iii. Store information about the following objects: chunk, file, client, device, workspace (sync folder)
c. sync service
    i. If there is a client update, another client sync after file update processing
    ii. sync client local (metadata) DB and server metadata DB
    iii. When the client is offline for a certain period of time, a long polling request is made as soon as it is online (for updating)-> sync service is updated after checking the consistency in the metadata DB

    iv. optimization
        - Designed to transmit less data between client and cloud storage (ex) using differencing algorithm
        - Transfer only the differences between the two versions (parts of the changed file)
        - If the server already has a chunk with the same hash (even for other users), cancel the upload and use the existing chunk
        - Efficient and scalable sync protocol
            ✓ Using communication middleware (between client and sync server)
            ✓ The messaging middleware must provide scalable message queuing and update notifications that can support a large

number of clients with a pull / push strategy.

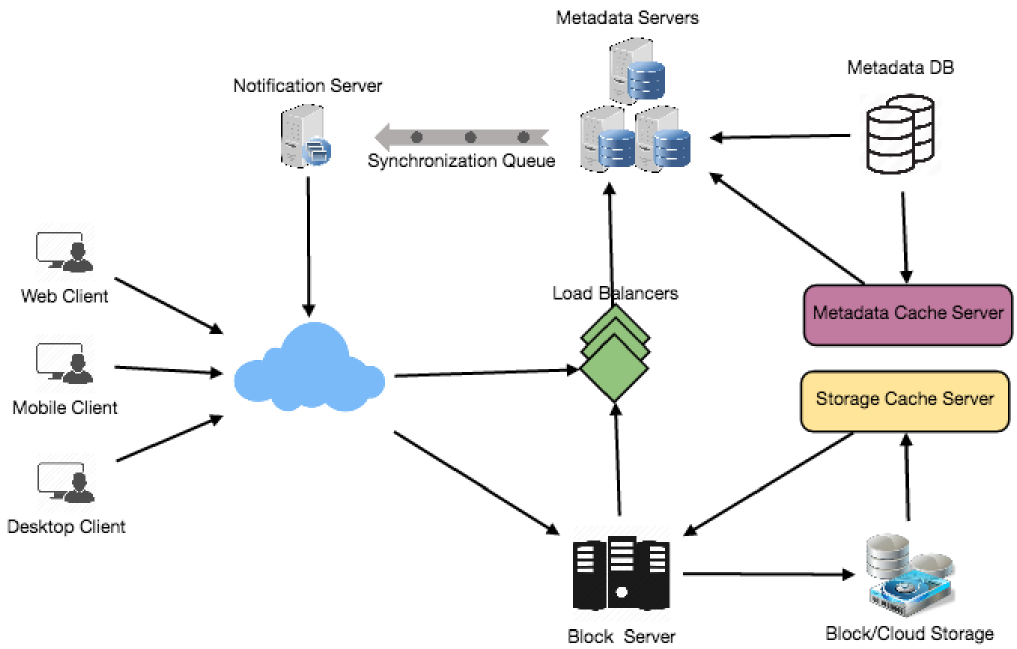
✓ Multiple sync services can receive requests in the global request queue, and communication middleware can adjust the balance

d. message queue service (13.jpg)



i. messaging middleware
ii. async message based communication between client and sync service
iii. Asynchronous loose connection message-based communication between distributed servers
iv. Implementation of two queues
   - request queue
     ✓ global queue-shared by all clients
     ✓ Client sends metadata DB update request to request queue-> sync service is metadata update
   - response queue
     ✓ Send update message to each client
     ✓ Since the message received by the client is immediately deleted from the queue, if you need to share the updated message for each device, you can create a separate response queue.

e. cloud/block storage (14.jpg)



   i. Save user update file
   ii. Client interacts directly with storage to send and receive objects in storage

7. file processing workflow
   a. Client 1 uploads chunk to cloud storage
   b. Client2 updates metadata and commits changes
   c. Client 1 confirms and sends notification of changes to clients 2 and 3.
   d. Clients 2 and 3 download the updated chunk after receiving metadata changes
8. data deduplication
   a. data deduplication
      i. Delete duplicate data to increase storage utilization
      ii. Also applicable for network data transfer
   b. post-process deduplication
      i. Find and delete duplicate files after saving a new chunk
      ii. Pros: No storage degradation
   c. in-line deduplication

        i. Real-time deduplication hash calculation when saving
        ii. When a previously stored chunk is identified, only a reference to the chunk is added to metadata
        iii. Provides minimal (optimal) network and storage usage

9. metadata partitioning-partitioning schema (split and store data in another DB)
    a. vertical partitioning
        i. Store tables related to one specific function on one server (ex) Store user related tables in one DB and file / chunk related tables in another DB
        ii. Disadvantages
            - No future expansion method
            - Joining each table in two DBs causes performance and consistency problems (ex) Joining user and file tables
    b. range based partitioning
        i. Based on the first letter of the file path
        ii. Static partitioning scheme: files can be saved / searched in a predictable way
        iii. Disadvantages
            - Inconsistent distribution of servers (ex) Too many file paths starting with e
    c. hash based partitioning
        i. Get the hash of the object and select the DB
        ii. Disadvantages
            - The distribution of servers is not uniform-> solved by consistent hashing
10. cache
    a. Block storage cache for hot file / chunk processing
    b. memcached: store the entire chunk with the corresponding id / hash and block server

    c. cache eviction policy : LRU (Least Recently Used)

11. load balancer
    a. Location
        i. Between client and block server
        ii. Between client and metadata service
    b. Initial Policy: round-robin method
        i. Advantages
            - simple, no overhead
            - Do not send traffic to a non-working sever
        ii. Disadvantages
            - Failed to check server load-Traffic is transmitted even to a heavily loaded server
        iii. Solution
            - Use intelligent LB with server load check
12. security, permissions and file sharing
    a. Save each file permission in metadata DB