16. 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

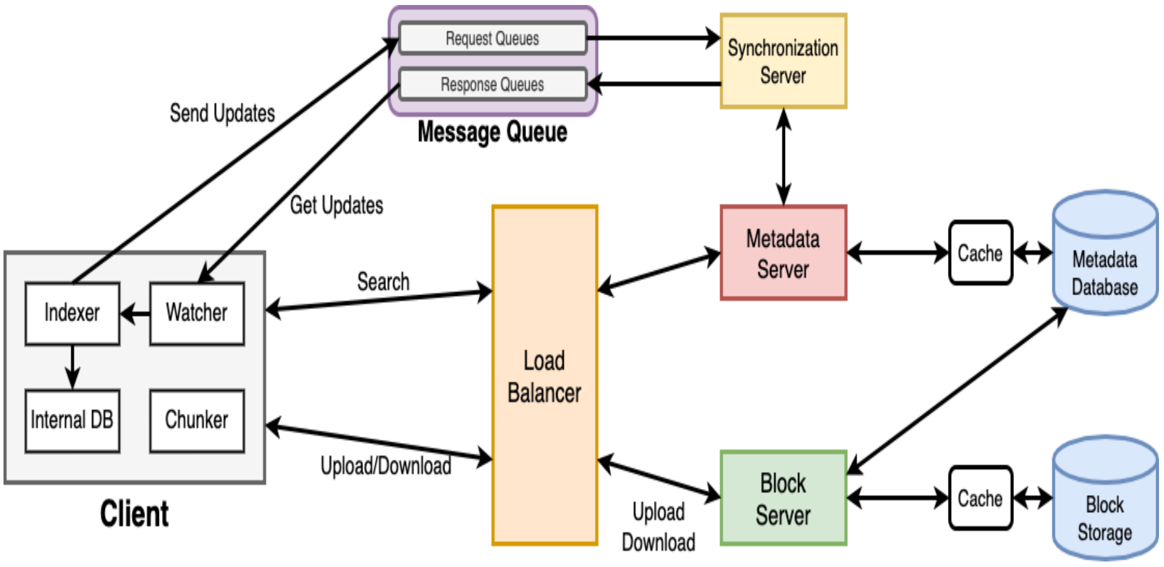
- **Traffic estimation**
  - ▪ Our system will have huge read and write volumes.
  - ▪ Read-write ratio is expected to be nearly the same.
  - ▪ Users
    - ✓ 500 million users. (Assumed)
    - ✓ 100 million daily active users. (Assumed)
  - ▪ Files
    - ✓ Each user has 200 files. (Assumed)
    - ✓ 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.

Object Storage System

- Real-life examples
- Requirements clarification
- Estimation
- System interface definition
- Data model definition
- High-level design
- Detailed Design
  - Data Stores
    - ✓ High Level Design
    - ✓ Data Persistent Flow
    - ✓ How data is Organized
  - Metadata Data Model
  - Object Versioning
  - Optimizing uploads of large files

Real-life examples

- AWS S3

Requirements clarification

- **Functional requirements**
  - Bucket creation: Users can create buckets.
  - Object uploading and downloading: Users can upload or download objects to/from buckets.
  - Object versioning: One object can have multiple versions.
  - Listing objects in a bucket: User can see all object information of a bucket.
- **Non-functional requirements**

    - High availability
    - Storage efficiency (Reduce storage costs while maintaining a high degree of reliability and performance).

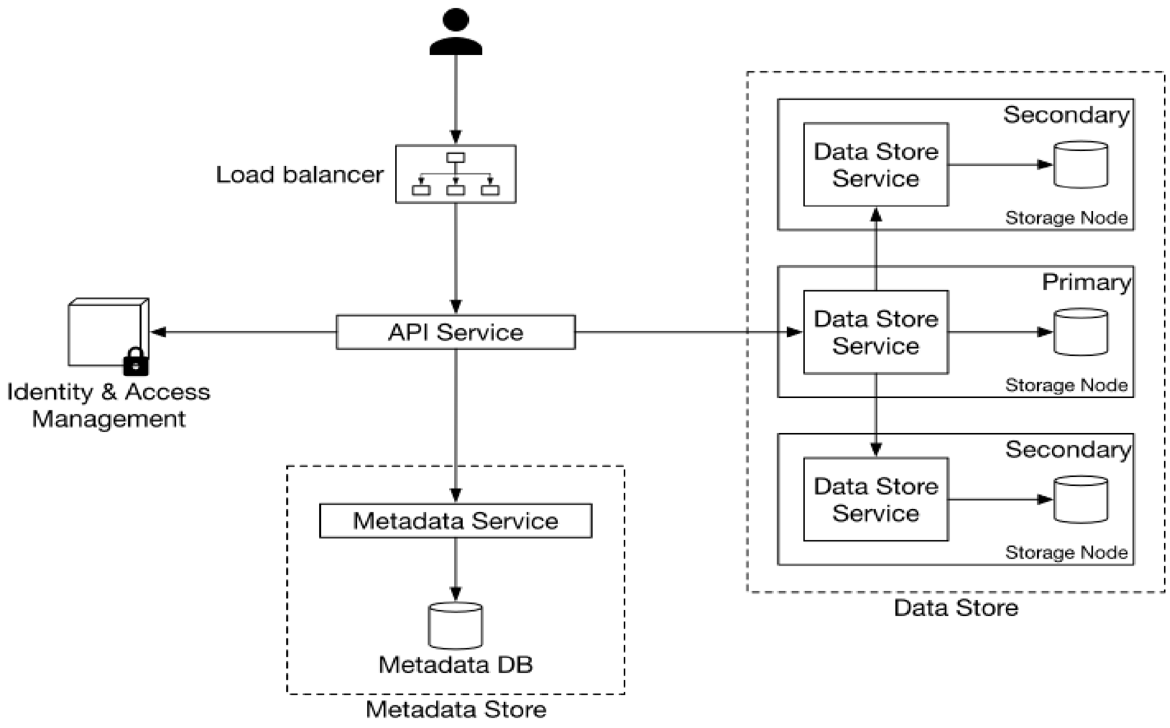
Estimation

System interface definition

- **Upload object**
    - PUT /bucket_name/object_name
- **Download object**
    - GET /bucket_name/object_name

Data model definition

High-level design

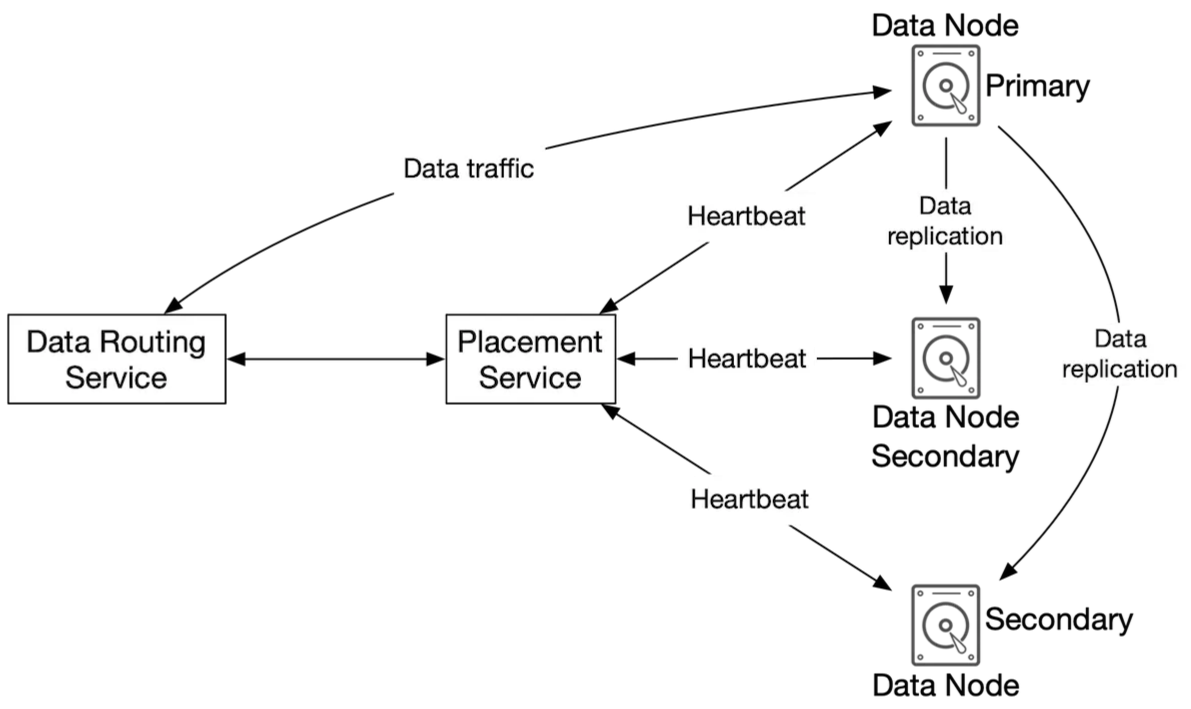


- **Load balancer**
  - Distributes RESTful API requests across a number of API servers.
- **API service**
  - Orchestrates remote procedure calls to the identity and access management service, metadata service, and storage stores.
- **Identity and access management (IAM)**
  - Handles authentication, authorization, and access control.
- **Metadata store**
  - Stores the metadata of the objects.
- **Data store**
  - Stores and retrieves the actual data.

Detailed design

Data store

High-level design

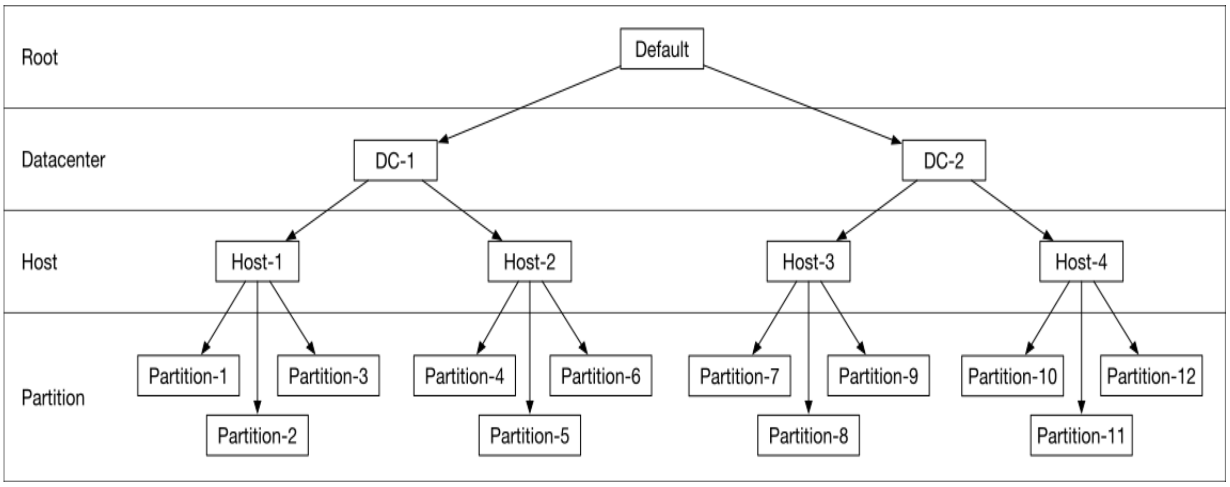


Data Routing Service

- Provides RESTful or gRPC APIs to access the data node cluster.
- Queries the placement service to get the best data node to store data.
- Reads data from data nodes and return it to the API service.
- Writes data to data nodes.

Placement Service

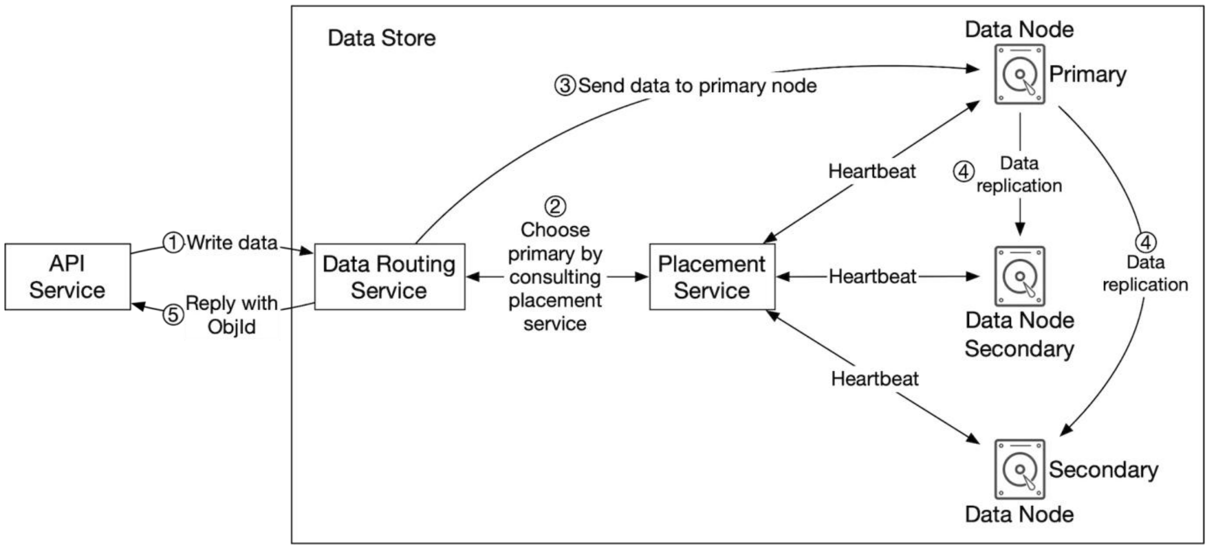
- Determines which data nodes (primary and replicas) should be chosen to store an object.
- Maintains a virtual cluster map, which provides the physical topology of the cluster.
- Continuously monitors all data nodes through heartbeats.



Data node

- Stores the actual object data.
- Ensures reliability and durability by replicating data to multiple data nodes, also called a replication group.

Data persistence flow



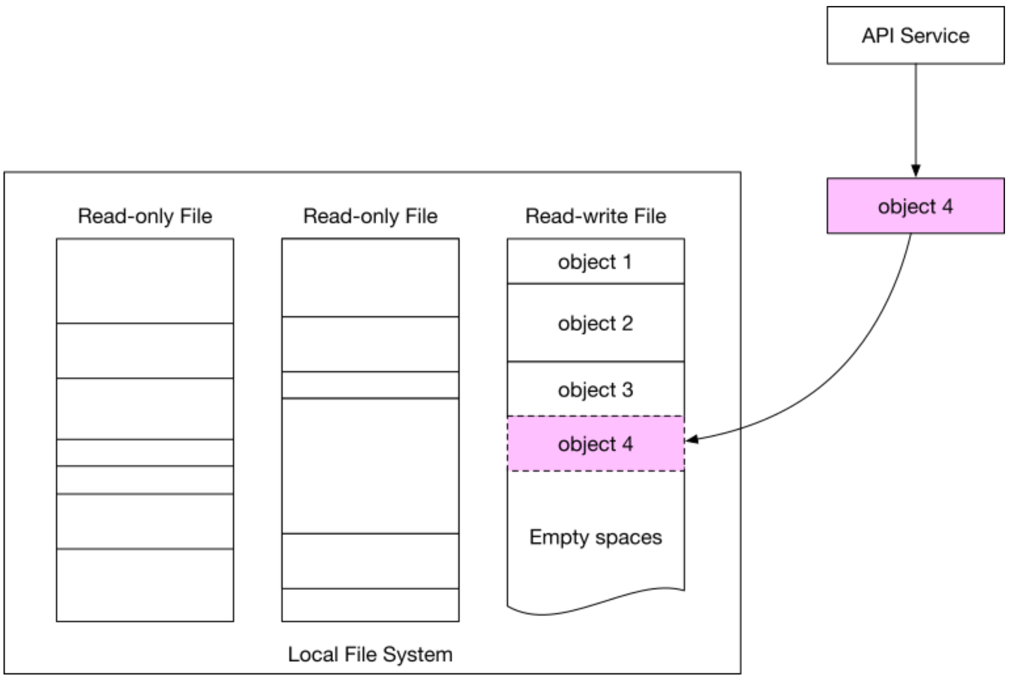
- Step 1: The API service forwards the object data to the data store.
- Step 2: The data routing service generates a UUID for this object and queries the placement service for the data node to store this object. The placement service checks the virtual cluster map and returns the primary data node.

- Step 3: The data routing service sends data directly to the primary data node, together with its UUID.
- Step 4: The primary data node saves the data locally and replicates it to two secondary data nodes. The primary node responds to the data routing service when data is successfully replicated to all secondary nodes.
- Step 5: The UUID of the object (ObjId) is returned to the API service.

How data is organized

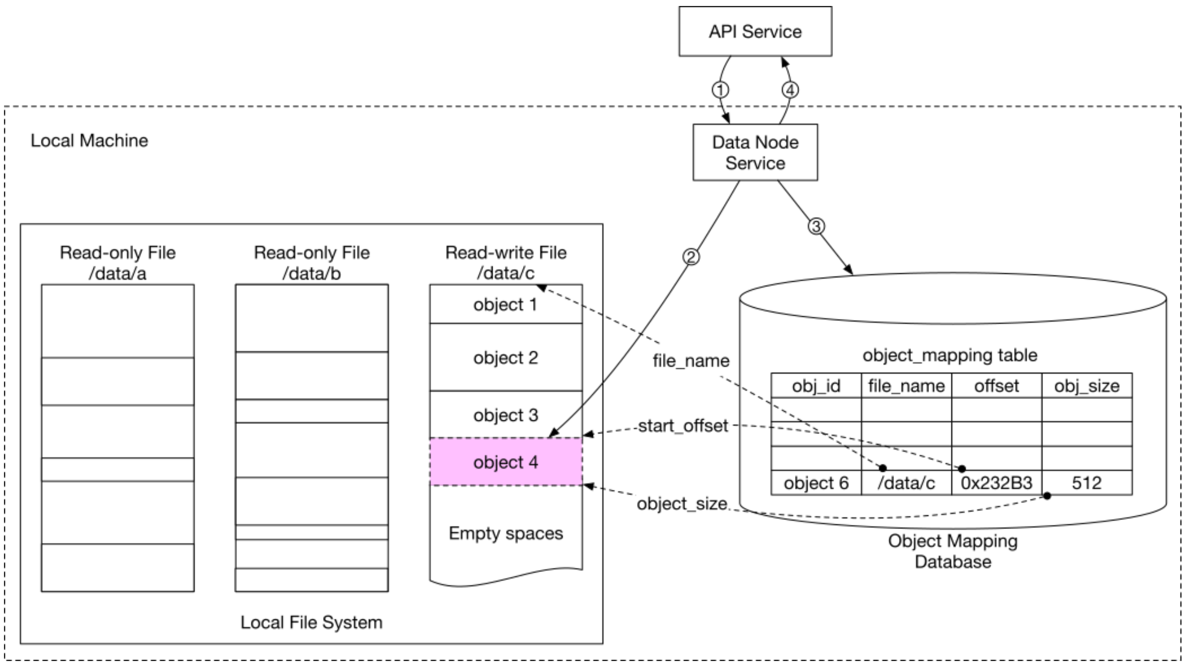
Store many small objects into a larger file:

- When we save an object, it is appended to an existing read-write file.
- When the read-write file reaches its capacity threshold, the read-write file is marked as read-only, and a new read-write file is created to receive new objects.
- Once a file is marked as read-only, it can only serve read requests.



Use the object_mapping table to store the relationship between object and file

Field	Description
object_id	UUID of the object
file_name	The name of the file that contains the object.
start_offset	Beginning address of the object in the file.
object_size	The number of bytes in the object.



Metadata data model

Table : Objects

Field	Description
object_name	The name of the object.
object_id	The UUID of the object.
bucket_id	Which bucket the object belongs to.
object_version	The version number (TIMEUUID) of the object.

Table: Buckets

Field	Description
bucket_name	The name of the bucket.
bucket_id	The UUID of the bucket.
owner_id	The owner of the bucket.
enable_versioning	The bucket enabled versioning or not.

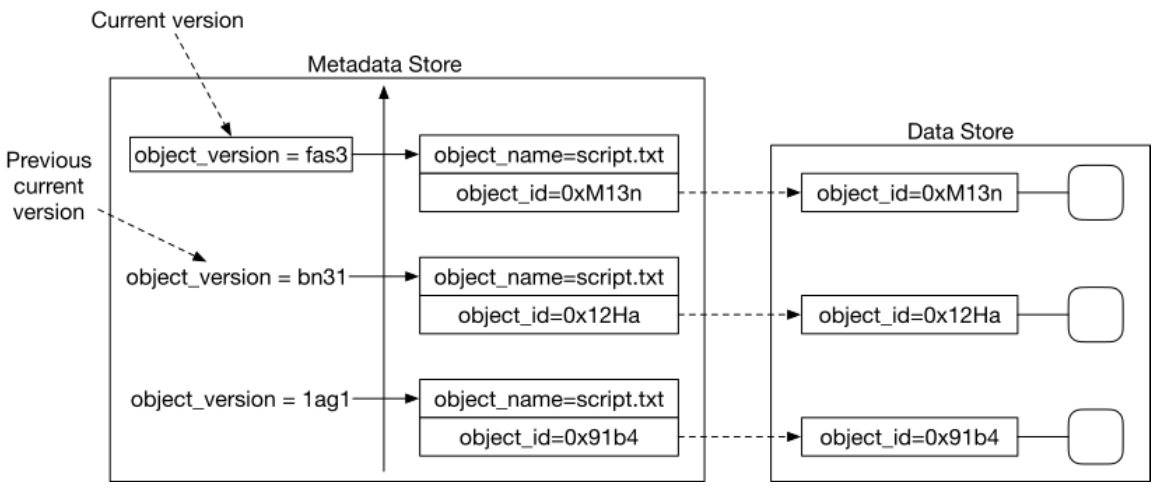
Sharding key

- The hash of the combination of bucket_name and object_name.

Object versioning

If an existing object is modified

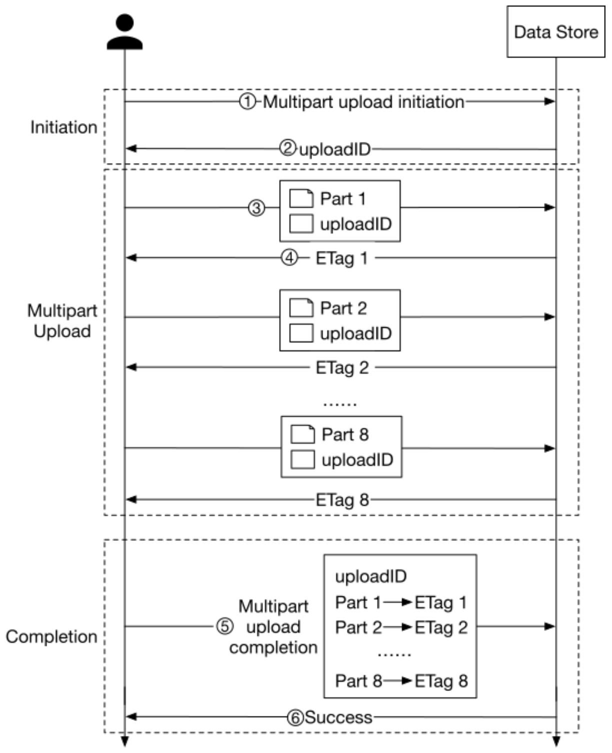
- Insert a new record in the objects table.
  - Same bucket_id and object_name as the old record.
  - Generate new object_id (UUID) and object_version (TIMEUUID) for the new record.
  - The current version has the largest TIMEUUID of all the entries with the same object_name.



Optimizing uploads of large files

Solution

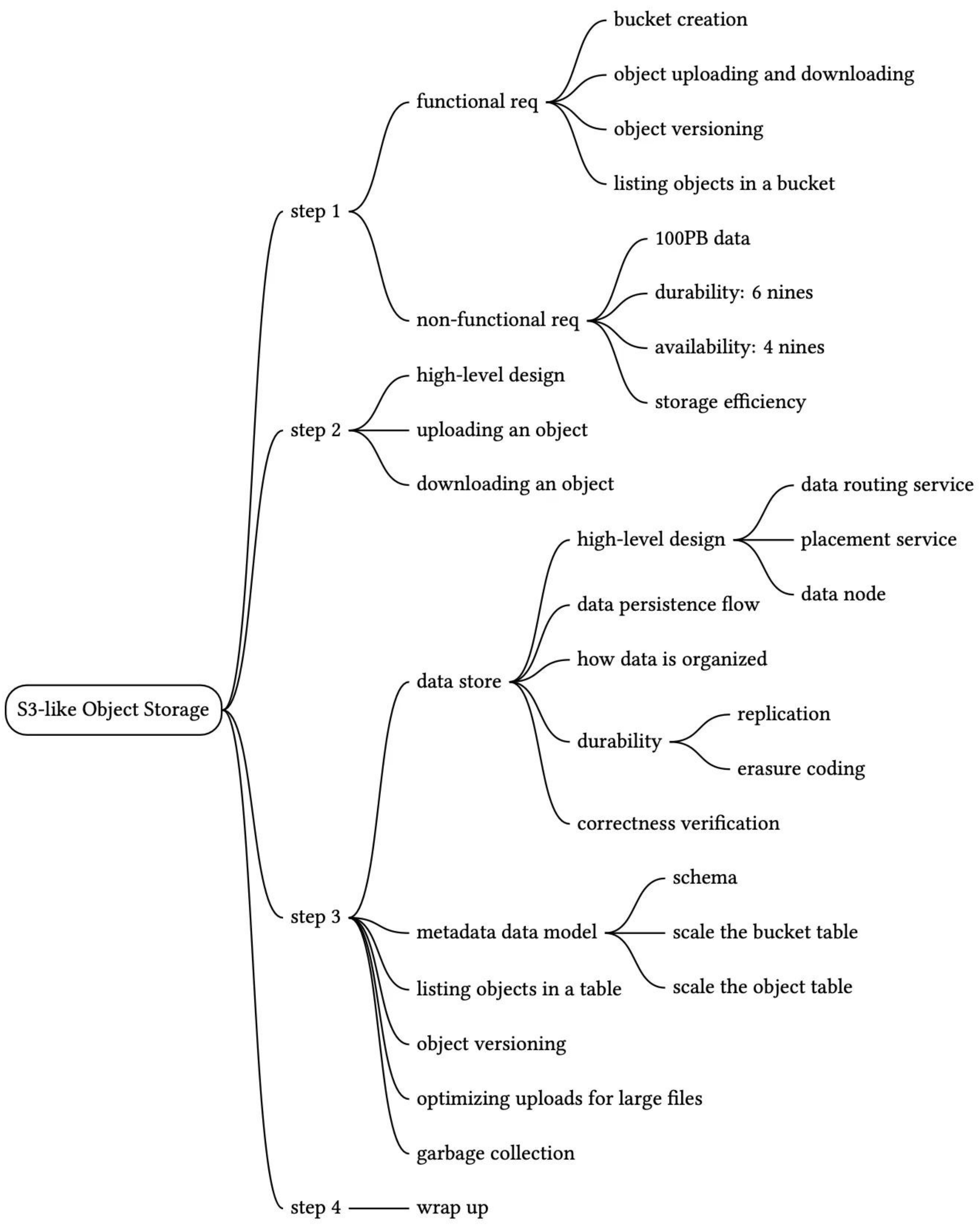
- Slice a large object into smaller parts and upload them independently.
- After all the parts are uploaded, the object store re-assembles the object from the parts.



Key points

- The data store does not store the name of the object and it only supports object operations via object_id (UUID).
- Multiple small objects will be stored in a single file. Use the object_mapping table to locate the object in a certain file.
- Same data will be replicated to different data nodes, also replicated to different Availability Zones.
- For supporting multiple versions of an object, the object_name should be same but object_id (UUID) should be new.

Summary


S3-like Object Storage
step 1
functional req
bucket creation
object uploading and downloading
object versioning
listing objects in a bucket
non-functional req
100PB data
durability: 6 nines
availability: 4 nines
storage efficiency
step 2
high-level design
uploading an object
downloading an object
step 3
data store
high-level design
data routing service
placement service
data node
data persistence flow
how data is organized
durability
replication
erasure coding
correctness verification
metadata data model
schema
scale the bucket table
scale the object table
listing objects in a table
object versioning
optimizing uploads for large files
garbage collection
step 4
wrap up