Storing and retrieving of large number of images for analysis

Background

The purpose of this write-up is to discuss optimal ways to store and retrieve large numbers of images for analysis. The system should take input images (with metadata and properties i.e. annotation etc) and store them somewhere. Also, the stored images might have to be retrieved for predictive analytics and restore with analytics output.

Storage Estimation

Total number of images = ~8 millions
Per image size = ~ 1MB
Total storage requirement for images = 8 millions * 1MB = 8 TB
For metadata, we can assume 10-20 GB storage might be required. So, to round up and for unexpected scenarios, we can add 2 TB which totals 10 TB of storage requirements for everything.

System Considerations

1. We can store image related metadata and analytics stuff in our database. The actual image can be stored in object storage servers. Database will contain the path of the image in object storage server and we can have some kind of caching mechanism to easily retrieve images. Proper folder structure is very important to manage images in object storage servers. Also image integrity and security is a concern for this approach. The emergence of cloud technologies nowadays has made the storage/retrieval of large numbers of images easier.

2. We also can store images in a database as BLOB type. This is good to maintain data integrity, security and everything being in one system makes it easier to maintain and program. On the other hand it is resource intensive to retrieve and insert bulk of images.

We will discuss the first option in the rest of this document.

Database Schema

PrimaryKey	Image_ID
	File_Path

Fig: Image metadata table_1

ForeignKey	Image_ID
	Annotations
	Predicted_values

Fig: Image metadata table_2

The image metadata table1 will have Image_ID (string) and File_Path (string). The other table will have Image_ID as foreign key and Annotations (serialized string) and Predicted_values (serialized string for each annotations accordingly). Separating Image_ID and File_Path from others will make the retrieval and insert more efficient in case when we will not do any prediction or annotation inserts. Depending on the frequency of operations, we can also split the prediction and annotations in separate tables. (For example, we might not require predicted values in the first inserts. In that case, it will be faster to insert annotations only and not the predictions)

There will be a system to generate unique image ids. To generate 8 million unique ids from 26 English alphabets, we will need 5 (26^5 >> 8 million) characters long strings.

We can index the database by image ids for efficient retrieval. Also, the storage can be divided using the first 1,2 or 3 characters of image_id so that all images are not stored in one bucket.

We will also shard our database by Image_ID for efficient insert and retrieval.

The system

Firstly, user request will go to a load balancer which will pass the request to a free server. Each server will have upload/download services as well as analytics service. For an insert request, the server will hit respective shard of database to insert metadata and will pass the image to respective folder/bucket of storage. For retrieval request, it will first check the cache if image is already there (or in case of prediction value retrieval it will check cache if the respective prediction value is there). If the cache is missed, it will go to respective shard to retrieve metadata and then will retrieve the image from storage.

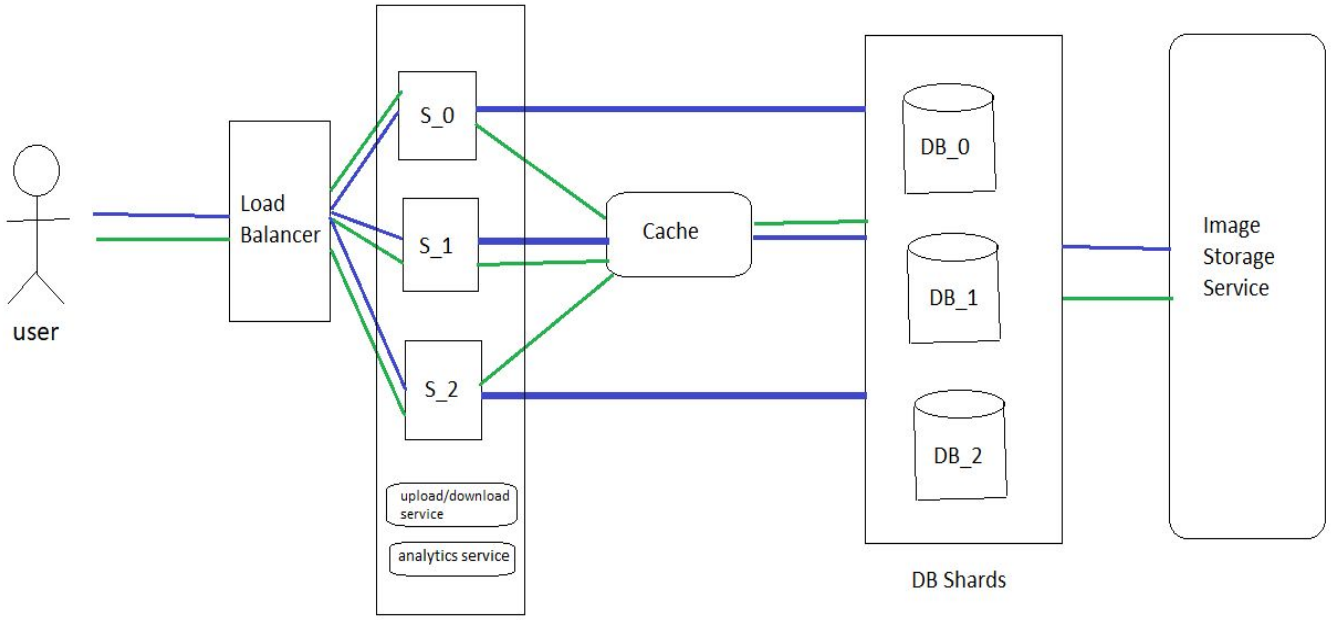

Fig: The system in a simple diagram. Blue color represents insert flow and green color represents retrieval flow.

Other Considerations

We can benchmark our database with different query types, single request , multiple requests of size n, different settings of servers to understand performance of throughput, latency, concurrency and scalability. We can also try different indexing and query optimization techniques to improve the performance of our database.