Solutions for Assignment 2

Rodrigues, Diogo
MTK 03770446
diogo.rodrigues@tum.de

TUM – Cloud-Based Data Processing 2022/23
16th November 2022

My answers for Assignment 2.

1. **List the main functional requirements of your system.**
   Keep in mind the assumptions in the above section (Designing Instagram), and focus on the browsing/posting experience. Feel free to make any assumptions of your own. (hint: start with around four requirements and prioritize them)
   A:
   - Persistent storage of photos, likes and followers.
   - A user can see the top posts of the users they follow, sorted by likes.
   - A user can follow another user.
   - A user can see the posts of another user.
   - A user can publish a post.
2. **List the main non-functional requirements of your system.**
   Hint: start with around four requirements and prioritize them.
   A:
   - The first 3 photos in a user feed should be provided in at most 100 ms.
   - All other photos after the 3 first photos should be available instantly when user scrolls down news feed.
   - Photo upload can be slower than download, but not too slow (1000 ms per photo). When a photo is posted, it is not important if some users see the post and others do not, as long as there is eventual consistency over a short time span (30 s) under normal system operation.
   - Newer posts should be favored against older posts in terms of performance, because users are expected to access newer posts much more frequently than older posts.
   - The system should be scalable.
   - Security: make sure only authenticated users can post with their account.
3. **How will your system stand in terms of the CAP theorem and why?**
   A: The system stands in AP, since eventual consistency is acceptable. Therefore, we can sacrifice strong consistency (C) in favor of high availability (A) and network partitioning resistance (P).
4. **What type of workloads should we expect?**
   A: We expect mostly transactional workloads, where users make small but very frequent updates to the system data (upload and download photos, like photos and follow other users, which are simple operations that involve small amounts of data, but they are very frequent and should be handled with as least latency as possible).
5. **Capacity Estimations**
   Assume that we have 500M total users, and 2M photos posted every day. If the average photo size is 200KB, how much storage will we need:
   (a) For 1 day of photos? **A:** 400 GB

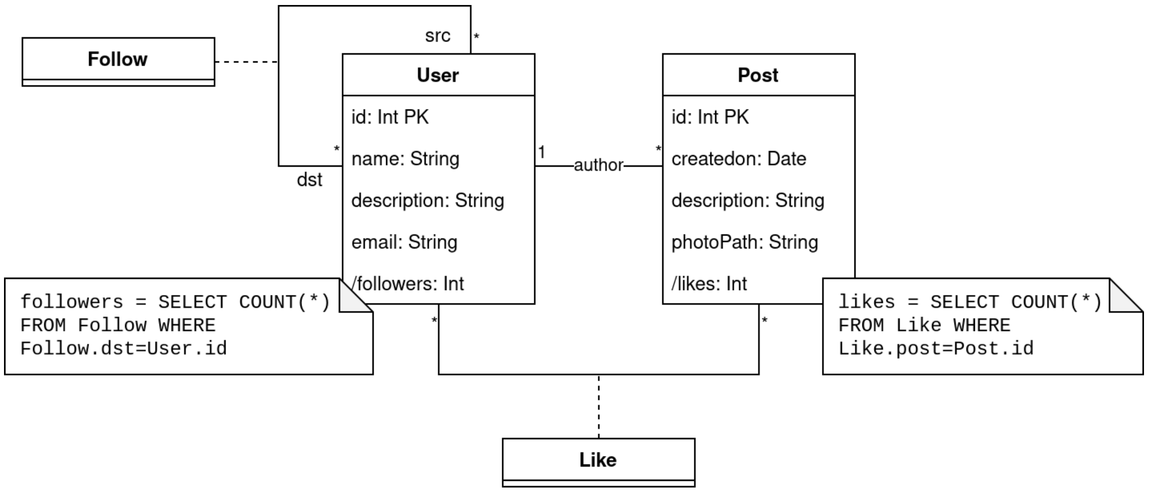
(b) For 10 years? **A:** $1460\,\mathrm{TB}$

With 23 photos uploaded every second, how much bandwidth should the whole system support? **A:** $37\,\mathrm{Mbit/s}$

6. **Define the system's API.**
Define the most important operations in terms of their input parameters and their return types and values. If it helps thinking about something concrete, you can assume building a REST API. There is no need to do any background research on REST APIs.
A:

   - `PUT /user`: Create a new user.
   - `GET /user`: Get information of user.
   - `GET /photo/{photoID}`: Get a photo.
   - `GET /post/{postID}`: Get a post. Returns the `photoID`, the `userID` of the user that posted it and the number of likes.
   - `GET /post/{postID}/like`: Get list of users that liked post `postID`. Returns `userID[]`
   - `PUT /post/{postID}/like/{userID}`: User `userID` likes post `postID`
   - `GET /post/{postID}/like/{userID}`: Check if user `userID` liked post `postID`. Returns boolean.
   - `GET /user/{userID}/feed`: Get feed for user. Returns `postID[]`.
   - `PUT /user/{userID}/post`: Create a post (and upload a photo). Returns `postID`
   - `GET /user/{userID}/posts`: Get posts made by user.

7. **Database Schema Design**
Provide a simple database schema. You can use UML if you know it, otherwise, any format is sufficient.
A:



8. **What kind of database would you choose to store data and why?**
Hint: it might be better to store different types of data in different components
A: Our data can be split into two main types: *photos*, which are the main content of our platform and should be the majority of the data we will be storing; and *metadata*, which includes user information, followers and likes. The metadata can be stored using a relational database that can meet our requirements of availability over consistency. The photos can be stored in a filesystem, and made available through a CDN.

9. **Handling read/write concurrency**
Photo uploads (writes) can be quite slow as they have to go to the disk, whereas reads are faster, especially if they are served from cache. How can we ensure that reads continue being served quickly, even in the case of multiple write requests?
A: By allowing reads even in a scenario where parallel writes are being executed in the system. This can be performed using a system that guarantees high availability with eventual consistency and automatic conflict resolution.

10. **Handling resiliency.** Losing files is not an option for our service. How will you ensure that even if a storage server dies, no data will be lost?
    A: The only way to decrease the chance data is lost is to duplicate data in many nodes that do not share any factors that could cause them to fail simulaneously; so, we want replication in nodes in different regions/availability zones, so their failures are as independent as possible. This also helps at serving data faster because the node that is geographically closest to a user can serve him.

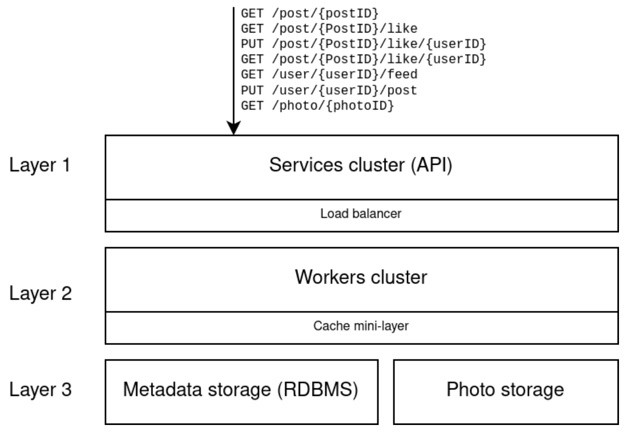
11. **Give a partitioning for your database scheme.**
    Specify whether you will use hash-partitioning or range partitioning, and explain your partitioning scheme.
    A: I would use hash partitioning with respect to the user ID. Therefore, all posts of a user are stored together with the user information. This partitioning scheme reduces the latency of obtaining a list of posts of a user, although it may overload the databases that contain certain users with a lot of followers (in which case a solution with replication can be implemented for better performance).

12. **(Optional) How can we plan for the future growth of our system (in terms of partitioning)?**
    A: in terms of partitioning, we could macro-partition data by range using the date, and send older posts to long-term storage that we expect will not be frequently accessed (and will also be cheaper), and have the most recent posts in special storage that expects a lot of accesses. Even then, inside each macro-partition, data should be hash partitioned to distribute the load evenly over all nodes.

13. **Which cache eviction policy would you use in your system?**
    A: LFU (Least Frequently Accessed), because we want to keep the most accessed posts in cache to obtain the most speed gains.

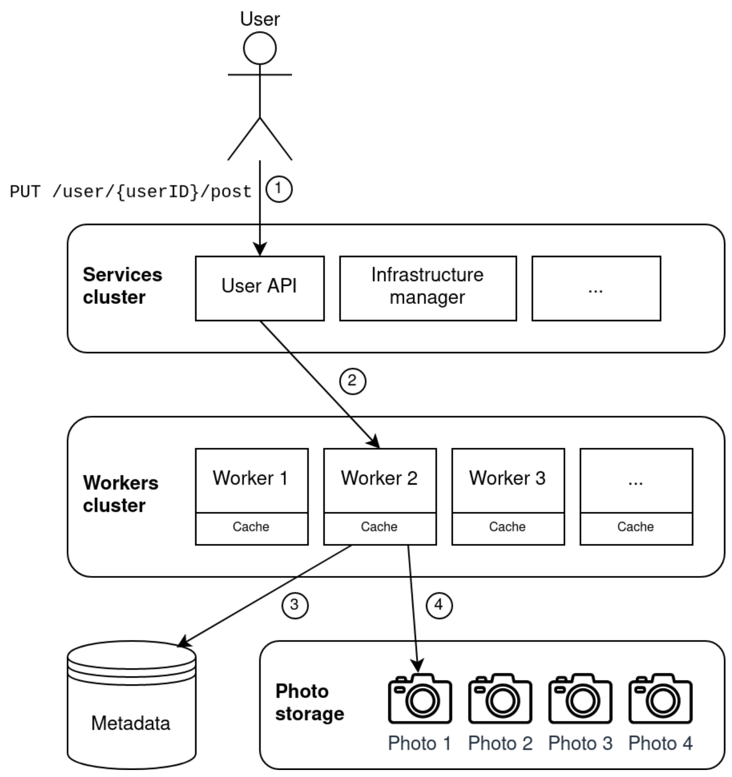
14. **How can we build a more intelligent cache?**
    Hint: 80-20 rule
    A: We can build a more intelligent cache by using the 80-20 rule, where we say that 80% of the requests refer to posts of the 20% most popular users (users with most followers). This allows us to know which items to place in cache beforehand.

15. **Draw a high level diagram of your design, including all components.**
    A:



```
GET /post/{postID}
GET /post/{PostID}/like
PUT /post/{PostID}/like/{userID}
GET /post/{PostID}/like/{userID}
GET /user/{userID}/feed
PUT /user/{userID}/post
GET /photo/{photoID}
```

16. **Draw a diagram visualizing how a photo upload is handled by your system.**
    A:


User
PUT /user/{userID}/post
1
Services cluster
User API
Infrastructure manager
...
2
Workers cluster
Worker 1
Cache
Worker 2
Cache
Worker 3
Cache
...
Cache
3
4
Metadata
Photo storage
Photo 1
Photo 2
Photo 3
Photo 4