Google Doc:

1. How the system should be used? Use Cases
   1. File storage
   2. Online editing and formatting
   3. Collaboration (Concurrency)
      1. There are solutions like (OT) Operational Transformation: The basic idea is to transform each person’s mutation based on its revision and revisions from other collaborators. Examples: ‘abc’: ‘xabc’; ‘ab’🡪’xab’
   4. Access Control (owner, read-only, allow comment etc.)
2. Scaling and Constraints

When scaling system to millions of users, there can be a lot of issues?

* 1. Speed (example: update permission of a folder, the update should be propagated to all its children. Speed can be a concern here.)
  2. Consistency. (Multiple replications…)
  3. Propagation

Twitter

1. Use Cases:
   1. Send Tweets
   2. Following
      1. For example, there are features like suggestions about who to follow.

This is a core feature that plays an important role in user onboarding and engagement.

* 1. Serve feeds
  2. Search feature
     1. Also involves with ranking algorithm:

The most straightforward approach is to give each feature/signal a weight and then compute a ranking score for each tweet. Then we can just rank them by the score. Features can include reply/retweet/favorite numbers, relevance, freshness, users popularity etc.

* 1. Trending topic
     1. Ranking of topic based on frequency, reply/retweet/favorite numbers

1. Data Modeling - use a relational database like MySQL
   1. User object
   2. Feed Object
2. How to scale the system when there are millions/billions of users?
3. How to evaluate the system?
4. How to design the same feature for Facebook (bi-directional relationship)

Instagram (Photo Sharing app)

Two major objects: (Relational database)

* User object

User name, email, registration date

Relations: user follow relations (not a bi-directional relation/user-picture relation

* Picture object

Potential scale issues (what if we have millions of users)

1. Response time: one costly operation is to render users feed

* One approach: upgrade the ranking algorithm (ranking by date and only read the top n most recent pictures with the infinite scroll feature)
* accelerating the picture fetching and ranking is the core

1. scale architecture

For example, we can have database separate from web apps (in different servers) with [load balancers](https://en.wikipedia.org/wiki/Load_balancing_(computing)).

1. Scale database

For this specific problem, we can either do the [vertical splitting (partitioning)](https://en.wikipedia.org/wiki/Partition_(database)) by splitting the database into sub-databases like user database, comment database etc. or [horizontal splitting (sharding)](https://en.wikipedia.org/wiki/Shard_(database_architecture)) by splitting based on attributes like US users, European users.

1. Feed Ranking

* come up with a scoring mechanism, rank everything in chronological order, like/comment numbers, whether the user has liked many photos of the owner and so on. A [linear combination](https://en.wikipedia.org/wiki/Linear_regression) can be used as a starting point due to simplicity.
* Later, more advanced machine learning algorithms like [collaborative filtering](https://en.wikipedia.org/wiki/Collaborative_filtering) is worth to try.

1. Image compression

[Amazon S3](https://aws.amazon.com/s3/)

The point is that images are usually of large size and seldom get updated. So a separate system for image storage has a lot of advantages. For instance, cache and replication can be much simpler when files are static.

**Trending algorithm**

General idea: To represent a topic, use a **term** (i.e. a word, a hashtag, a phrase….etc.)

If a term has a huge volume in recent tweets (compare to past volume : this is important, there are word like weather, Monday…which has no point to be selected as a trending word), this term will be identified as **popular.**

Infrastructure:

Offline pipelines: keep several pipelines running in the offline that **calculates the ratio of each term**  (volume within last few hours / volume within last X days) **and output the results to some storage system**. The pipelines may refresh every several hours assuming there’s no big difference between a short period of time

About the term volume:

 (volume within last few hours / volume within last X days) : This can potentially give problems. 🡪  *volume within last few hours / (volume within last X days + 10000)🡪* This can dilute the small volume. Or adding the absolute volume as another indicator.

Influencers: Those who have high volume of followers

The tweets from influencers can be counted \* a parameter > 1

Personalization:

calculate a relevance score between each topic and the user based on signals including his previous tweets, who he has followed and what tweets he has favorited etc..

Location should also be a valuable signal. We may even calculate trending topics for each location (maybe city level).

**Facebook Chat (Messaging App)**

Chat server that acts as the core of the whole system. When a message comes, it won’t be sent to the receiver directly. Instead, it goes to the chat server and is stored there first. And then, based on the receiver’s status, the server may send the message immediately to him or send a push notification.

Real-Time: [HTTP persistent connection](https://en.wikipedia.org/wiki/HTTP_persistent_connection)

showing online friends: It improves user experience tremendously…

**Design a Recommendation System**

Machine learning (ML) is commonly used in building recommendation systems, it doesn’t mean it’s the only solution.

Heuristic solution:

For instance, based on videos a user has watched, we can simply suggest videos from same authors. We can also suggest videos with similar titles or labels. If we use the popularity (number of comments, shares) as another signal, the recommendation system can work pretty well as a baseline.

Collaborative filtering (CF)

In a nutshell, to recommend videos for a user, I can provide videos liked by similar users. For instance, if user A and B have watched a bunch of same videos, it’s highly likely that user A will like videos liked by B. Of course, there are many ways to define what “similar” means here. It could be two users have liked same videos, it could also mean that they share the same location.

The above algorithm is called user-based collaborative filtering. Another version is called item-based collaborative filtering, which means to recommend videos (items) that are similar to videos a user has watched.

Features that can be used to build the recommendation system

explicit and implicit features…

* Freshness can be a very important factor. We should figure out how to recommend fresh content.
* Eval is an essential component of recommendation system, which allows us to understand how well the system works.
* To train the collaborative filtering system, we may also include video position signals. Usually, videos ranked on top have much higher chance to be clicked.

Design a Cache system:

In short, a cache system stores common used resources (maybe in memory) and when next time someone requests the same resource, the system can return immediately. It increases the system efficiency by consuming more storage space.

LRU Cache

Concurrency Issue with Cache:

An alternative is to use commit logs. To update the cache, we can store all the mutations into logs rather than update immediately. And then some background processes will execute all the logs asynchronously. This strategy is commonly adopted in database design.

Design eCommerce Website

First of all, building an eCommerce website requires things like database design, system availability, concurrency consideration and so on so forth. All of them are extremely important in today’s distributed systems.

Three major objects: *Product*, *User* and *Order*.

*Product*defines the basic model for a product in the shopping cart. Some important fields include price, the amount left, name, description, and the category. Category can be tricky here.