Design a news feed system

Practice on: 19 jan 2025

Time Taken: 95 mins (1 Hour 35 min)

Questions for the Interviewer

1. What is this news feed for? social media application, news application etc?
2. Which content is added to feed? Friends we follow, personality we follow, pages we follow
3. What can be part of feed? text post, photos, videos
4. Can user perform any action on the feed post? like, comment, share
5. Is high consistency needed as a person post something? need to show on other people feed in realtime?

Requirements

Functional

1. User should be able to post
2. User should be able to see his feed

Non-functional

1. Scalable: system should be able to handle large number of users
2. Latency: feed load time should be as low as possible (~200 ms)

Capacity Estimation

Assumptions:
1. Active users: 1 Billion
2. DAU: 100 million
3. Avg post size: 500 kb | Avg video size: 2 MB
4. photo:video post > 10:1
5. people post:not post > 1:100

Throughput(WPS & RPS)
WPS > number of people post > 1/100 * 100 million(10^6) = 100 * 10^4 = 10^6 => 1 million post a day
=> 1 million / 24*60*60 ~ 12 post/sec
RPS > 12 * 100 = 1200 rps

Storage
=> 1 million post a day
=> photos been posted in a day => 1 million * 9/10 = 9 * 10^5 photos
=> videos been posted in a day => 1 million * 1/10 = 1 * 10^5 video
=> storage for photos => .9*10^6 * 500*10^3 ~ 500*10^9 ~ 500 GB
=> storage for video => .1*10^6 * 2*10^6 ~ .2 * 10^12 ~ 200*10^9 ~ 200 GB
=> total storage = 700 GB / day

Bandwidth
Assume write speed is 500kb/s and read speed is 1mb/s
Write bandwidth = 12 post/sec * 500kb/s = 6000 kb/s = 6 MB/s
Write bandwidth = 1200 rps * 1 mb/s = 1200 mb/s = 1.2 GB/s

Core Entities

1. User
2. post
3. feed
4. User connection (people he/she follow, pages he follow)

APIs

User posting a post

REQUEST:
HTTP POST /post
body {
    auth_token: "*****",
    caption: "happy new year 2025...",
    attachment: "location"
}

CHECK LATER: HOW AN ATTACHMENT IS SENT IN A POST REQUEST??

RESPONSE {
    message: "posted, get back in sometime"
}

User get his news feed

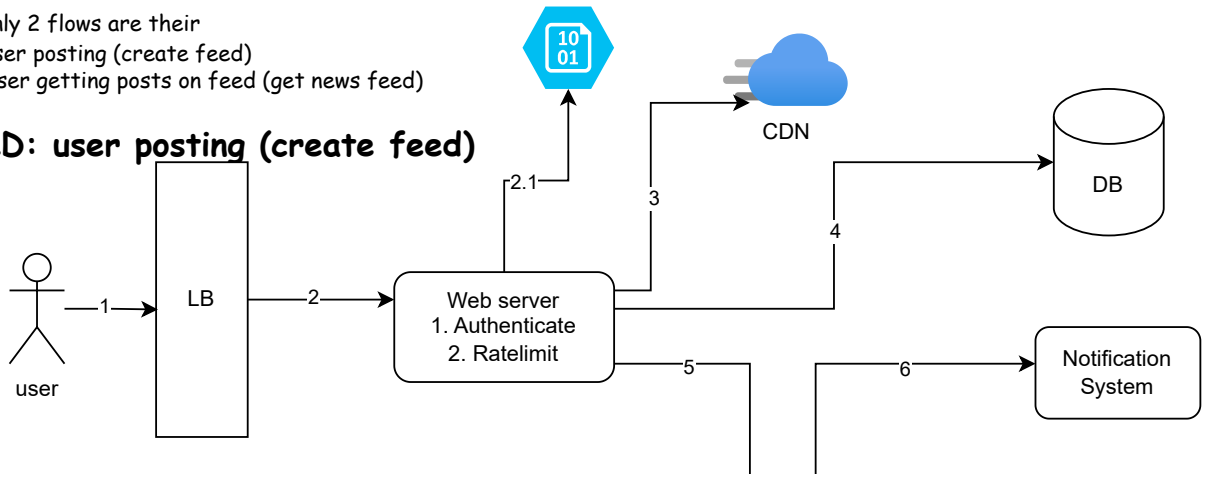
REQUEST:
HTTP GET /feed

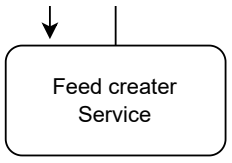
RESPONSE {
JSON with a list of all the post
}

High level Design

mainly 2 flows are their
1) user posting (create feed)
2) user getting posts on feed (get news feed)

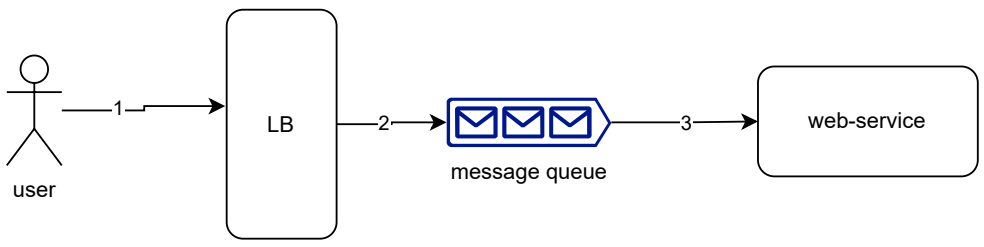
HLD: user posting (create feed)





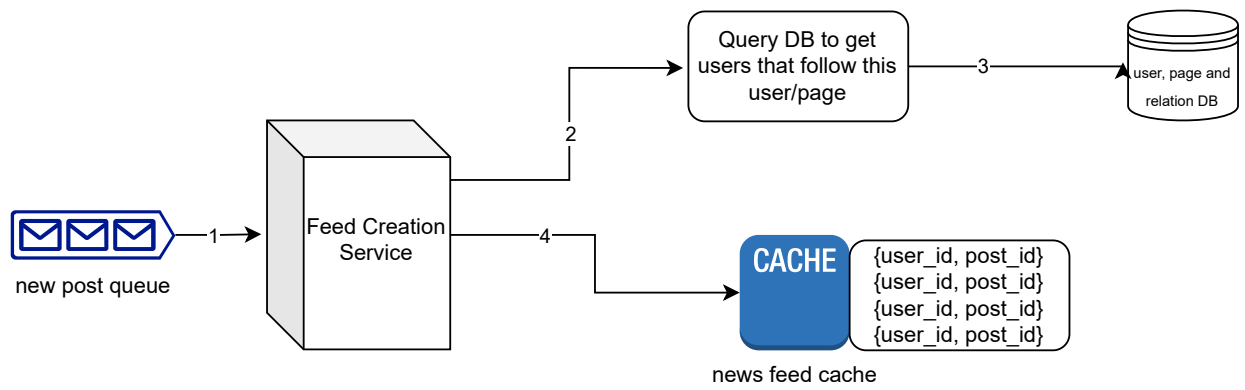
1. User requested to post a photo, request goes to LB
2. LB is needed for scale and as their can be many instances of web service. LB send request to web service
3. Web service authenticate the user and post. It will also rate limit the user, so user can post a particular number of posts under a min
4. The photo/video will be stored to a blob stoareg like s3 and link to the storage is return to the webservice
5. The photo/video should also be stored to CDN for quick access
6. The post details are been stored in the database, details for user and post will be added.
7. To create a feed, request is sent to feed creation service.

As we are uploading photo and video it will be better is requests are stored in a queue before web-service, so that the message is never missed. In fact we can put a queue in front of Feed creation Service as well. (THIS NEED TO CHECKED)



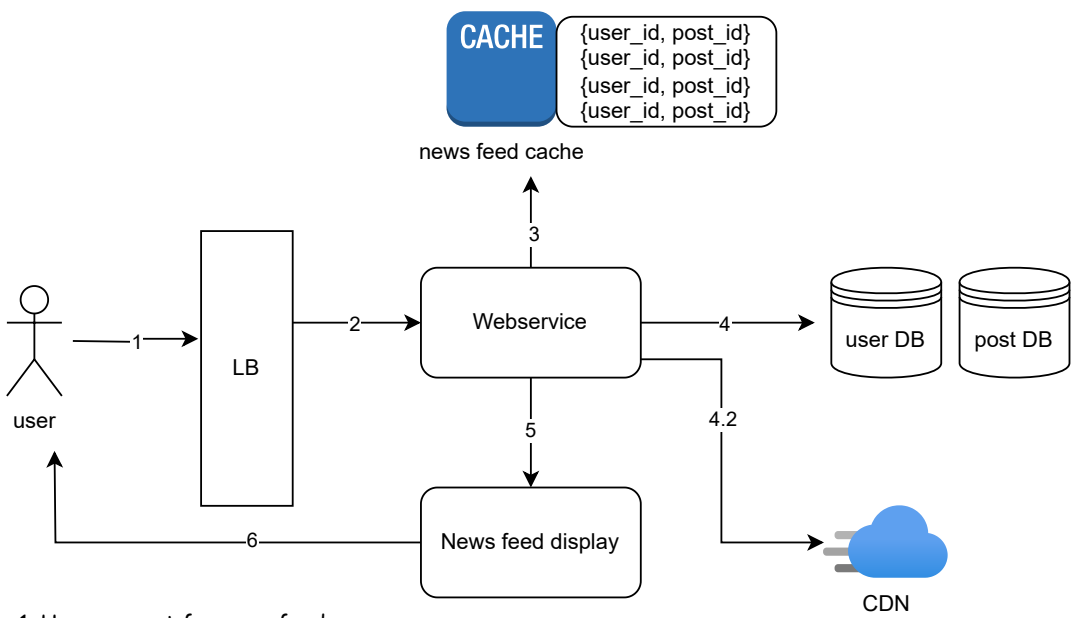
News Feed Creation Service

1. When a post is created by a user A, that post need to shown to all the friends / followers of user A.
2. When a post is created by a page B, that post needs to be shown to all the followers of that page.
3. When a user A create a post > we can query his friends / people who follow him/her > show them user A's post in their feed
4. When a page B create a post > we can query people who follow that page > show then page's B post in their feed



HLD: user getting posts on feed (get news feed)

Once user login/open his account, /get/feed API is been requested that calls the News Feed Cache and Create the news feed page for the user in chronological order.



1. User request for news feed
2. News feed GET service get {user_id, post_id} mapping for this user(user_id) from the cache
3. get post info from post DB with user who did the post from user DB
4. if post have photo/video get the link from CDN
5. show post on the feed page of the user

Design Deep Dive

1. Will keep CDN to store the photos and videos - this will decrease Latency
2. In front of DB will use cache - This will help in decrease Latency
3. For each service to remove SPF we will have multiple instances(Horizontal Scaling) > will keep load balancer in front of the web service
4. To query mySQL is good but it doesn't scale well, cane have nosql with graph for user relationship
5. SQL vs NoSQL > for query SQL is better, but scaling is an issue (SEE LATER: HOW TO SCALE SQL)
6. For CAP, we can have availability as the primary requirement over consistency

Design Review By ChatGPT

Strengths:

1. **Clear Requirements:** You clarified functional and non-functional requirements effectively, which helped guide the design.
2. **Capacity Estimation:** You included thorough calculations for storage, bandwidth, and throughput, showcasing a good understanding of system scale.
3. **API Design:** Your API specifications for creating and fetching news feed posts are concise and practical.
4. **Modular High-Level Design:**
    - Separation of responsibilities between services (e.g., feed creation, feed retrieval).
    - Use of queues for handling asynchronous operations like post creation.
5. **Performance Optimization:**
    - Use of CDNs for media delivery.
    - Cache for reducing database load and improving response time.
    - Horizontal scaling with load balancers to ensure scalability.

- Horizontal scaling with load balancers to ensure scalability.

Improvement Suggestions:

1. **Feed Ranking Logic:**
   - Add details on how posts are ranked in the feed (e.g., chronological, engagement-based, or algorithmic recommendations).
   - Include considerations for personalization (e.g., ML-based ranking).
2. **Consistency vs. Availability:**
   - You mentioned prioritizing availability over consistency but didn't elaborate on how inconsistencies (like delayed feeds) might be handled.
   - Discuss strategies like eventual consistency or conflict resolution.
3. **Database Design:**
   - Expand on the schema for the post database and relationships database.
   - For scalability, consider mentioning partitioning/sharding strategies for SQL or the use of graph databases like Neo4j for user relationships.
4. **Failure Handling and Monitoring:**
   - Elaborate on strategies for retrying failed requests in queues.
   - Consider implementing monitoring tools for real-time performance insights and fault detection.
5. **Security Concerns:**
   - Include mechanisms for ensuring secure API calls (e.g., authentication, rate limiting for abuse prevention).
   - Discuss handling sensitive data, like encrypted media uploads.
6. **Scaling Feed Creation:**
   - You mentioned querying followers for feed creation. At large scales, this might result in significant delays.
   - Discuss alternate approaches, like precomputing feeds for users or maintaining fan-out data structures (push-based).
7. **Latency Estimation:**
   - You estimated a latency goal of 200 ms but did not provide calculations or breakdowns for different components (e.g., cache retrieval, database queries).