1. Facebook newsfeed in which a user could upload posts, photos, videos, and update status.

2. To improve upon, try to include features like ‘likes’ from users, user can follow a page or join a group which could all be displayed on a scrollable format.

3. This design to be similar or be updated to other social media applications like Twitter, Quora or Instagram.

Feature Requirements

Primary features:

1. Posts, pages, and groups followed by a user must be populated on the newsfeed.
2. Consider designing the newsfeed to accommodate large user base, for example, a user might follow many pages or groups or could have many friends.
3. Newsfeed may not only be restricted to text, include images and videos in your design.
4. Your design must be able to periodically update the newsfeed to include new posts as and when the user refreshes.

Secondary Requirements:

1. The design must be able to refresh instantaneously, limit the delay to be within 2s.
2. A post should not take more than 5s to make it to a user’s feed assuming a new newsfeed request comes in.

Capacity Estimation and Constraints

Let us assume on average a user has 500 friends and follows 200 pages.

Traffic estimates:

Total users per day = 1 billion

Total News Feed per day = 1 * 5 = 5 billion = 57870 request per day

Storage estimates:

Average user post in feed = 300

Average Post size = 1 mb for Image/ 10 Mb For Video/ 100 kb for Text

Assuming a post contains all the three

= 10 + 1 + 10 = 11 MB

Total Storage for Data = 5 billion * 11 Mb = 55 PB

If a server can hold 5 TB of data in one server .we might need 1000 such machine to store all the 300 posts for given user.

System APIs

get Newsfeed (User_id, since , total count , max item count)

Database Design

User

Column	Type
User id	Integer
Username	String
Email	String
Last login	Timestamp
Created date	timestamp
Password	Timestamp

Entity (Page/Group)

Column	Type
Page/Group ID	Integer
Page/Group Name	String
Description	String
Creation date	Timestamp
Type	String
Email	String

Feed

Column	Type
feed id	Integer
Number of likes	Integer
Contents	String
Latitude	Integer
Longitude	integer
Last updated	Datetime
Creation date	Datetime

Media

Column	**Type**
Media id	Integer
Type	Integer
Path	String
Created date	Datetime
Longitude	Integer
latitude	integer

User followers

Column	Type
Follower id	Integer

Page/Friend/Group ID	Integer

Entity User

Column	Type
User id	Integer
Entity id	integer

User Feed

Column	Type
User id	Integer
Entity id	integer
Feed id	Integer
Media id	integer

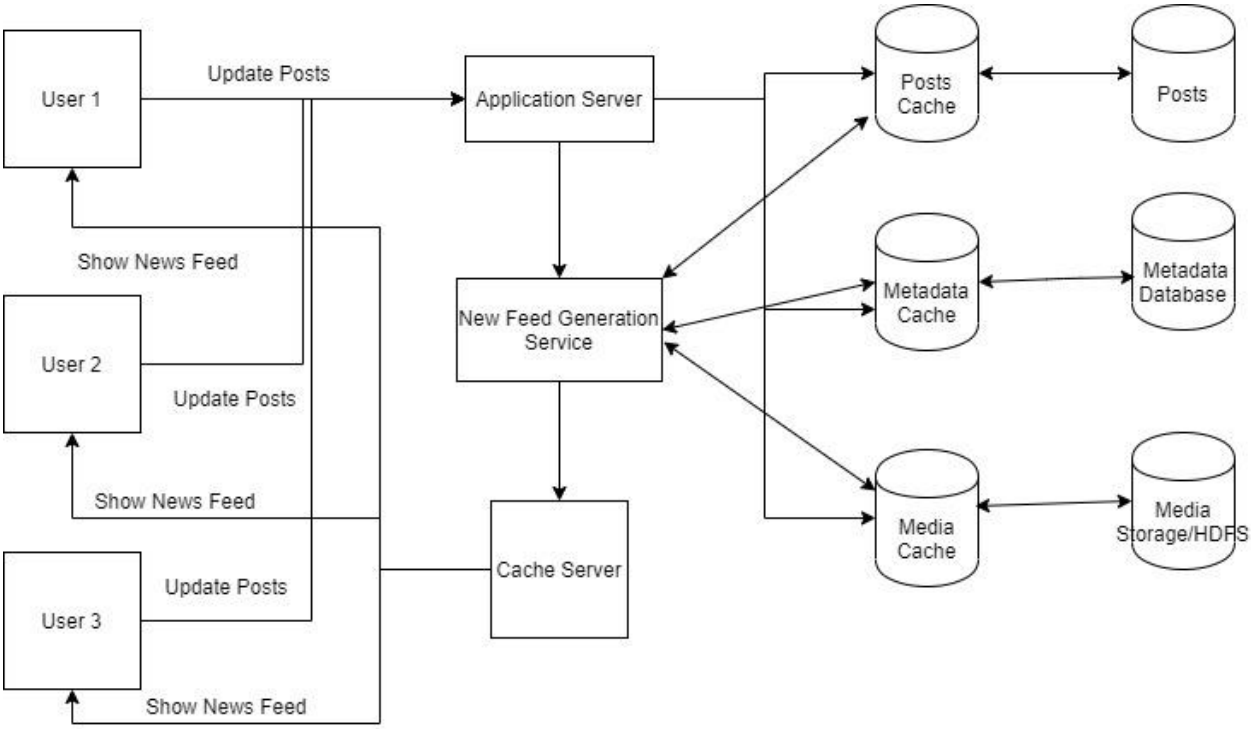
A straightforward approach for storing the above schema would be to use an RDBMS like MySQL since we require joins. But relational databases come with their challenges, especially when we need to scale them.

We can store media in a distributed file storage like HDFS or S3.

We can store the above schema in a distributed key-value store to enjoy the benefits offered by NoSQL. All the metadata related to media can go to a table where the 'key' would be the 'Media Id' and the 'value' would be an object containing Creation Timestamp, etc.

We need to store relationships between entity and photos, to know which entity owns which photo. Also, we need to store user-entity relationship. We also need to store the entity/friends of users. For all of these tables, we can use a wide column datastore like Cassandra.

High Level System Design



Detailed Component Design

News feed generation We need to fetch latest posts of user followers/page/groups to create and show news feed for any user.

Our application server will first get a list of people the user follows/page/groups and then fetch metadata info of latest 100 friends/entity from each user. In the final step, the server will submit all these tweets to our ranking algorithm which will determine the top 100 tweets (based on recency, likeness, etc.) and return them to the user. A possible problem with this approach would be higher latency as we must query multiple tables and perform sorting/merging/ranking on the results. To improve the efficiency, we can pre-generate the News Feed and store it in a separate table.

Pre-generating the News Feed: We can have dedicated servers that are continuously generating users' News Feeds and storing them in a 'User Feed' table or Redis Datastore. For this part ,we can use In Memory database such as Redis/Memcache So, whenever any user needs the latest photos/posts/videos for their News Feed, we will simply query this User Feed/Redis/Memcache and return the results to the user.

Whenever these servers need to generate the News Feed of a user, they will first query the User News Feed table to find the last time the News Feed was generated for that user. Then, new News Feed data will be generated from that time onwards

We have following approaches to send News Feed contents to the user

1. **Pull:** Clients can pull the News Feed contents from the server on a regular basis or manually whenever they need it. Possible problems with this approach are a) New data might not be shown to the users until clients issue a pull request b) Most of the time pull requests will result in an empty response if there is no new data.

2. **Push**: Servers can push new data to the users as soon as it is available. To efficiently manage this, users must maintain a Long Poll request with the server for receiving the updates. A possible problem with this approach is, a user who follows a lot of people or a celebrity user who has millions of followers; in this case, the server must push updates quite frequently.

3. **Hybrid**: We can adopt a hybrid approach. We can move all the users who has a greater number of followers to a pull-based model and only push data to those users who have a few hundred (or thousand) follows. Another approach could be that the server pushes updates to all the users not more than a certain frequency, letting users with a lot of follows/updates to regularly pull data.

.**Data Partitioning**

a. Sharding posts and metadata Since we have a huge number of new posts every day and our read load is extremely high too, we need to distribute our data onto multiple machines such that we can read/write it efficiently.
There are two ways to do that

User id – Sharding based on user id is highly effective in retrieval. In this ,we need to store both User_id and shared id in the system .The Feed_id has to be generated auto increment sequence and then combined with ShardID But it has downsides as it can leading to uneven distribution of loads on application and database shards

Also following question need to

- How would we handle hot users? Several people follow such hot users and a lot of other people see any photo they upload.
- Some users have infrequent post compared to others, thus making a non-uniform distribution of storage.
- What if we cannot store all post of a user on one shard? Distribute onto multiple shards will it cause higher latencies?
- Storing all posts of a user on one shard can cause issues like unavailability of all the user's data if that shard is down or higher latency if it is serving high load etc.

B Feed Id

If we can generate unique Fedics first and then find a shard number through “Feed_id % 100”, the above problems will have been solved. We would not need to append ShardID with Feed_id in this case as Feed_id will itself be unique throughout the system

b. Sharding feed data We can partition feed data based on user id. We can try storing all the data of a user on one server. However, this data should be more than 300 Feeds.

For querying feeds from set of servers, we can always Consistent Hashing. Also, it works well for scalability and growth of our system.

Caching

Since we are generating huge amount of data every day ,Distributed/In-Memory caching server can be of immense benefit for our system. These servers will be storing news feed for given user. The feed generation keep generating feeds and store them into database .Also replica of the feed will be stored in Caching Server. For this ,we propose to utilize Redis for our system. It is used as in memory distributed cache.

Redis has good support for all kind of data structures.

For storing and retrieving can used User_id ad key and feed object as value

Load Balancing

Since our system are both read and write heavy, we need to install load balancers between following entities

- Users and web server
- Web server and feed generation service
- Application server and cache server
- Cache server and database server