33. SOCIAL MEDIA SYSTEM

Follower Feed System

Requirements clarification

1. Functional requirements

- Post: Users can post feeds.
  - feeds can contain photos and videos.
- Follow: Users can follow other users.
- Timeline: System should be able to create and display a user's timeline consisting of top feeds from all the people the user follows.
  - User timeline: All the feeds a particular user has sent.
  - Home timeline: A temporal merge of all the user timelines of the people are you are following.

2. Non-functional requirements

- High availability.
- Acceptable latency of generating timeline is 200ms.
- High consistency is desirable (It should be ok for a user doesn't see a feed for a while).

Estimation

1. Traffic estimation

- Our system will be read-heavy.
- Users
  - 1 billion users. (Assumed)

- 200 million daily active users. (Assumed)
- 100 million new feeds every day. (Assumed)
- Each user follows 200 people on average. (Assumed)

- Number of feeds will be viewed per day
  - A user visits their timeline 2 times and visits five other people's pages every day. (Assumed)
  - Each page has 20 feeds. (Assumed)
  - Number of feeds will be viewed per day = 200 million x ((2 + 5) x 20 feeds) = 28 billion

2. Storage estimation

- Types
  - Data: Yes
  - File: Yes
- Capacity
  - Capacity for text
    - Each feed
    - Has 140 characters. (Assumed)
    - Need 30 bytes to store the metadata. (Assumed)
    - Each character needs 2 bytes.
    - Total size for storing new feeds per day = 100 million x (140 x 2 bytes + 30 bytes) = 30 GB
  - Capacity for photo and video
    - 20% feeds has a photo and 10% feeds has a video.

- Photo size is 200 KB and Video size is 2 MB
- Total size for storing new photos and videos per day = (100 million x 20% x 200 KB) + (100 million x 10% x 2MB) = 24 TB

3. Bandwidth estimation

- Text bandwidth = (28 billion x 280 bytes) / (24 hours x 3600 seconds) = 93 MB/s
- Phtot bandwidth = (28 billion x 20% x 200 KB) / (24 hours x 3600 seconds) = 13 GB/s
- Video bandwidth = (28 billion x 10% x 30% x 2 MB) / (24 hours x 3600 seconds) = 22 GB/s (Assume users only open to see 30% of videos in their timeline)
- Total bandwidth = 93 MB/s + 13 GB/s + 22 GB/s = 35 GB/s

System interface definition

Data model definition

Schema

- **Table 1: User**
  - Description
    - Store user accounts.
      - Columns

Column Name	Column Type	PK	Description
UserId	int	PK	The user ID.

Column Name	Column Type	PK	Description
Name	string		The name of the user.
Email	string		The email of the user.
Location	string		The location of the user.
LastLogin	datetime		The last login time of the user.

- **Table 2: Feed**
  - Description
    - Store the information of each feed.
      - Columns

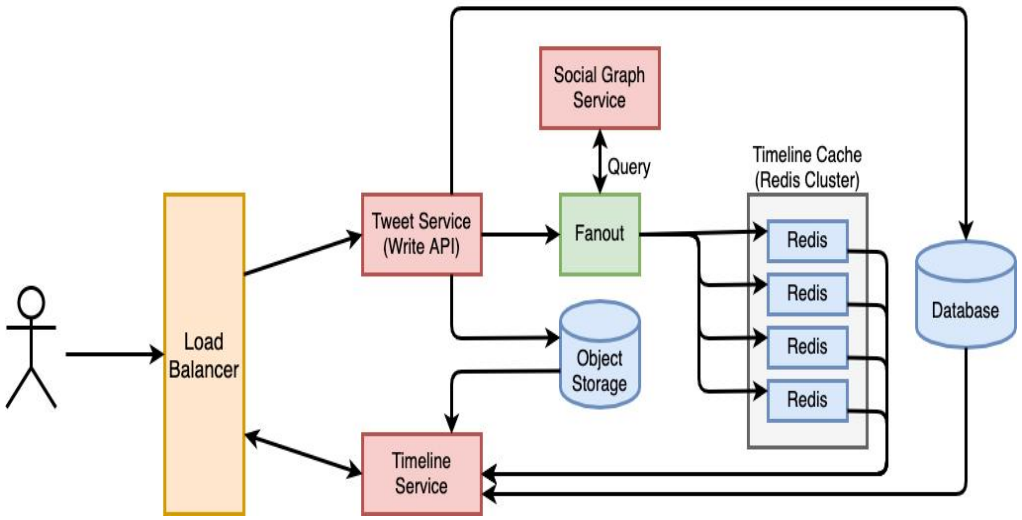
Column Name	Column Type	PK	Description
FeedId	int	PK	The feed ID.
UserId	int		The user ID of the user who created the feed.
Content	string		The text content of the feed.
Location	string		The location of the feed was published.
CreateTime	datetime		The time of the feed was published.
Path	string		The URL to access the photo or video of the feed in distributed file system.

- **Table 3: UserFollow**
  - Decription
    - Store the user following relationship.
      - Columns

Column Name	Column Type	PK	Description
FollowerUserId	int		The follower user ID.
FolloweeUserId	int		The user ID who has been followed.

- Data storage
  - Database
    - SQL database (We need to do join operations).
  - File storage
    - HDFS
    - Amazon S3
    - GlusterFS

High-level design



Feed Service

- Handle all the new feeds sent from users.

Fanout

- Take all the new feeds come in and place them into the Timeline Cache (A massive Redis cluster).
- Query the Social Graph Service to get the user following relationship.
- A single feed might be written into multiple Redis instances for improving read performance.

Social Graph Service

- Hold all the following relationship information between users.

Timeline Cache

- A massive reds cluster to store all the new feeds for each active user.

Timeline Service

- Generate timeline for users.

Object Storage

- Store photos and video for feeds.

Detailed design

Data sharding

Options

Option	Description	Pros	Cons
By UserID	Store all the data of a user on one server.		Load is not distributed evenly (The server holding a hot user will have a very high load comparing to the servers holding normal users).
By FeedID	Store feeds based on feed ID.	Load is distributed evenly.	Have to query all the servers for timeline generation.
By Feed creation time	Store feeds based on creation time.	Only have to query a very small set of servers for timeline generation.	Load is not distributed evenly (The server holding the latest data will have a very high load comparing to the servers holding old data).
By both feedID and feed creation time	Store feeds based on the new feed ID (Epoch seconds + Auto-incrementing sequence)	Reads and writes will be substantially quicker than the original feed ID solution. While writing, we don't have any secondary index on feed creation time. While reading, we don’t need to filter on feed creation time as our primary key has epoch time.	Have to query all the servers for timeline generation.

Friend Feed System

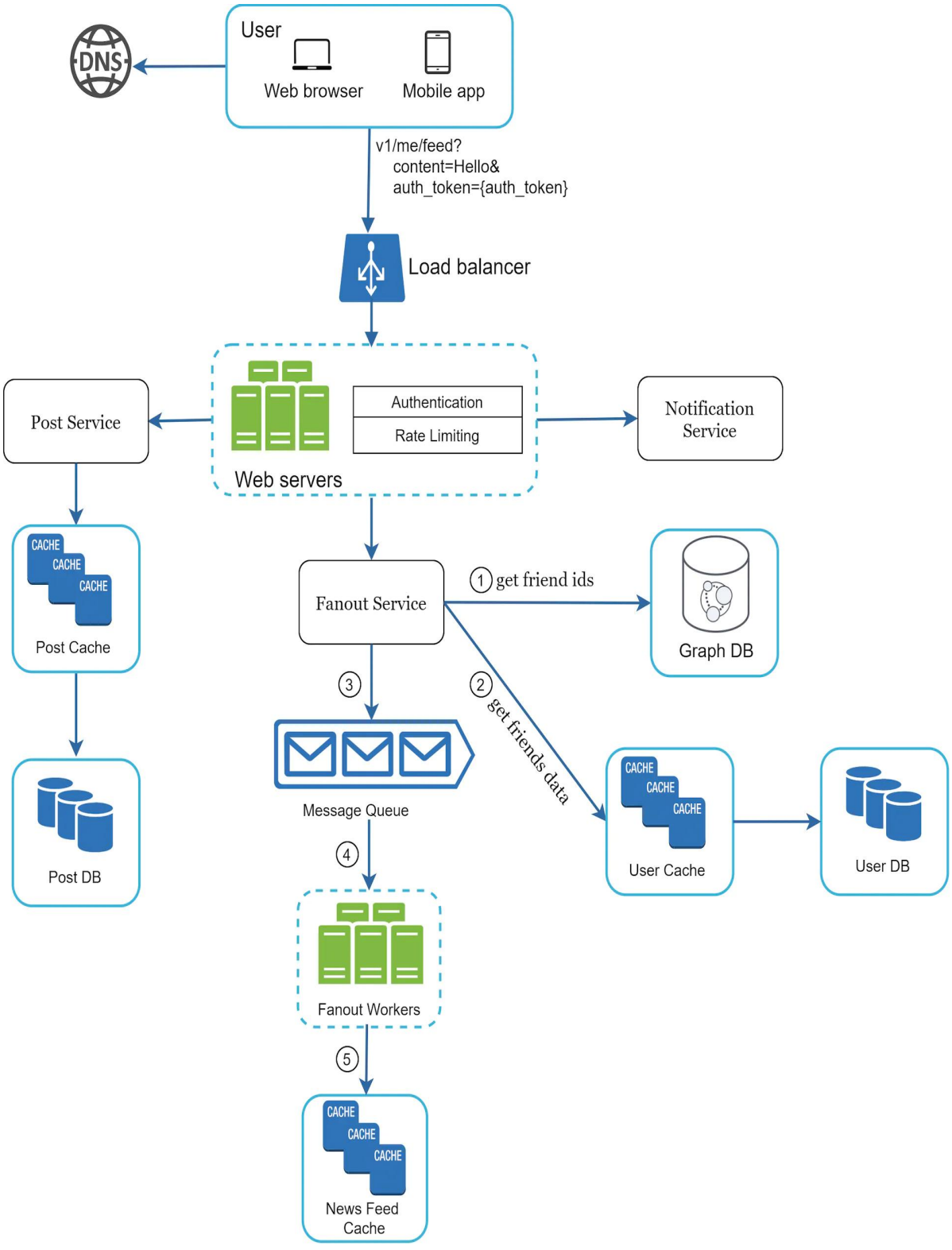
Real-life examples

- Facebook

Requirements clarification

- **Functional requirements**
  - A user can publish a post.
  - A user can see friends' post on the feed page.
  - A post can contain media files, including both images and videos.
- **Non-functional requirements**

High-level design



Web servers

- Enforces authentication and rate-limiting.

Fanout service

- Delivers a post to all friends
- Fetch friend IDs from the graph database and get friends info from the user cache.
- Send friends list and new post ID to the message queue.

Fanout workers

- Fetches data from the message queue and store news feed data in the news feed cache.
- Inserts <post_id, user_id> into news feed cache.

Graph DB

- Manages friend relationship and friend recommendations.

News Feed Cache

- Stores <post_id, user_id>

Messaging System

Real-life examples

- Facebook Chat
- Whatapp
- Slack
- Discord

Requirements clarification

Functional requirements

- Messaging: One user can send a message to another user or a group of users.
- Status: Shows online/offline statuses of uesrs.
- Images and videos uploading: User can upload images and videos in addition to text messages.
- Push notifications: Offline users can receive a push notification when there are new messages.
- Read receipt: Senders can get a receipt when receivers read messages they sent.

Non-functional requirements

- Low latency (real-time messaging).
- High consistency (users should be able to see the same chat history on all their devices).
- High availability is desirable (base on CAP theorem).

Data model definition

Schema

- **Table 1: User**
  - Description
    - Store user accounts
      - Columns

Column Name	Column Type	PK	Description
UserId	int	PK	The user ID.
Name	string		The name of the user.
LastActive	datetime		The timestamp of when the user is online (support online/offline status).

- **Table 2: Message**
  - Descrition
    - Store messages and their metadata.
      - Columns

Column Name	Column Type	PK	Description
MessageId	int	PK	The message ID.
SenderUserId	int		The user ID of the sender.
ConversationId	int		Identify the message belongs to which conversation.
Text	string		The text message of the message

Column Name	Column Type	PK	Description
MediaUrl	string		The url to access the media files attached to the message.

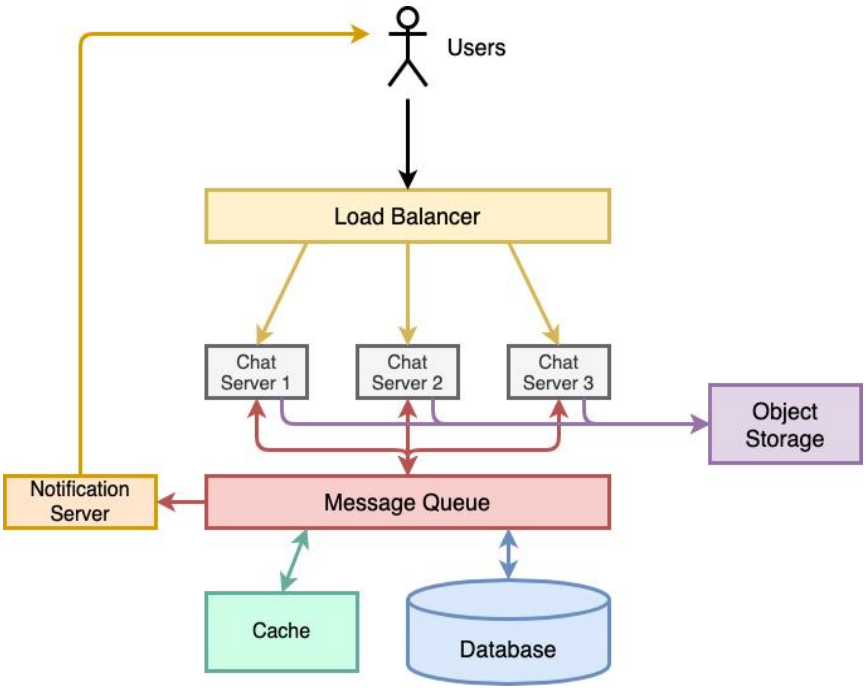
- **Table 3: Conversation**
  - Description
    - Store conversation information
      - Columns

Column Name	Column Type	PK	Description
ConversationId	int	PK	The conversation ID.
name	string		The name of the conversation (like channel name in Slack).

- **Table 4: ConversationUsers**
  - Description
    - Store the relationship about which user is a part of the converation.
      - Columns

Column Name	Column Type	PK	Description
ConversationId	int		The conversation ID.
UserId	int		The user ID of each user belongs to this conversation.

High-level design



Chat server

- Orchestrate all the communications between users (The direct connection between 2 users is not reliable).

Message Queue

- Handle the communication between chat servers.
- Each chat server will have a channel for receiving messages for that chat.

Object Storage

- Store media files.

Detailed design

- Users
  - Considerations
    - Consideration 1: Users receive new messages
      - Ideas

- It cannot be server initiated, it must be client initiated.

Solutions for receiving messages

- **Solution 1: HTTP polling (poor)**
  - Description: Users can repeatedly ask the server if there are any new messages for them.
  - Cons: Users will send lots of unnecessary to the server.
- **Solution 2: HTTP long polling (acceptable)**
  - Description
  - A user send one request to the server.
  - The server will hold the request, wait and response only if there is a new message for the user.
  - Cons: The server need to maintain lots of open connections.
- **Solution 3: WebSocket (good)**
  - Pros
    - Full duplex: Users and the server can send data at the same time.
    - Connections can keep open for the duration of the session.
- **Solution 4: BOSH - Bidirectional-streams Over Synchronous HTTP**
  - Key points
    - Use WebSocket for clients to get new messages.