Build Twitter

Requirements clarification

- **Functional requirements**
  - Post: Users can post tweets.
    - Tweets 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 tweets from all the people the user follows.
    - User timeline: All the tweets a particular user has sent.
    - Home timeline: A temporal merge of all the user timelines of the people are you are following.
- **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 tweet for a while).

Estimation

- **Traffic estimation**
  - Our system will be read-heavy.
  - Users
    - 1 billion users. (Assumed)
    - 200 million daily active users. (Assumed)
    - 100 million new tweets every day. (Assumed)
    - Each user follows 200 people on average. (Assumed)
  - Number of tweets 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 tweets. (Assumed)
    - Number of tweets will be viewed per day = 200 million x ((2 + 5) x 20 tweets) = 28 billion
- **Storage estimation**
  - Types
    - Data: Yes
    - File: Yes
  - Capacity
    - Capacity for text
      - Each tweet
        - Has 140 characters. (Assumed)
        - Need 30 bytes to store the metadata. (Assumed)
      - Each character needs 2 bytes.
      - Total size for storing new tweets per day = 100 million x (140 x 2 bytes + 30 bytes) = 30 GB
    - Capacity for photo and video
      - 20% tweets has a photo and 10% tweets 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
- **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.
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: Tweet
    - Description
      - Store the information of each tweet.
    - Columns

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

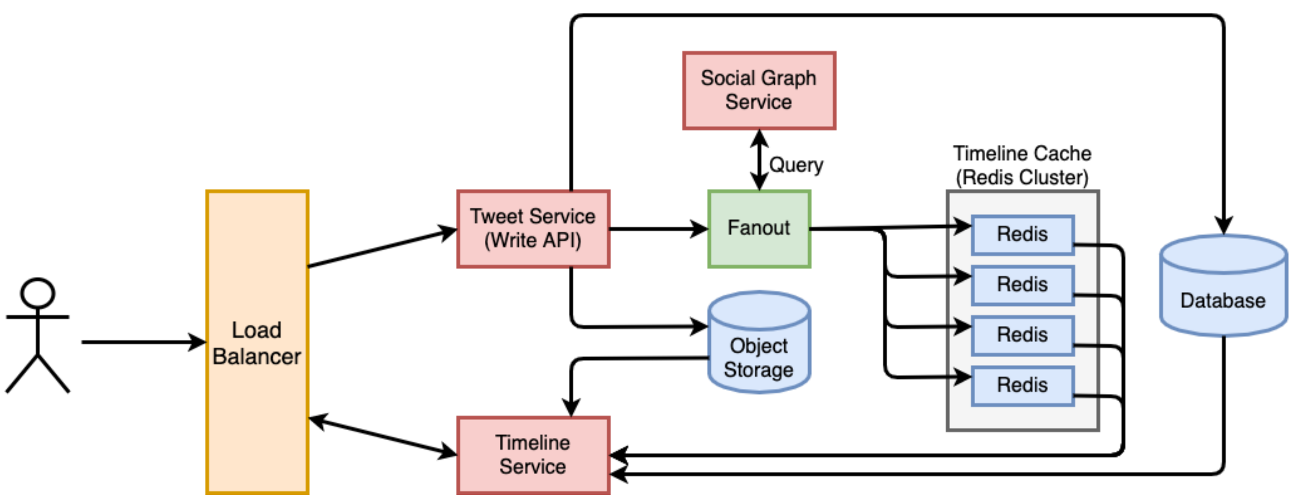
  - Table 3: UserFollow

- 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



- **Tweet Service**
  - Handle all the new tweets sent from users.
- **Fanout**
  - Take all the new tweets 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 tweet 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 tweets for each active user.
- **Timeline Service**
    - Generate timeline for users.
- **Object Storage**
    - Store photos and video for tweets.

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 TweetID	Store tweets based on tweet ID.	Load is distributed evenly.	Have to query all the servers for timeline generation.
By Tweet creation time	Store tweets 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).

Option	Description	Pros	Cons
By both tweetID and tweet creation time	Store tweets based on the new tweet ID (Epoch seconds + Auto-incrementing sequence)	Reads and writes will be substantially quicker than the original tweet ID solution. ▪ While writing, we don't have any secondary index on tweet creation time. ▪ While reading, we don’t need to filter on tweet creation time as our primary key has epoch time.	Have to query all the servers for timeline generation.

References

- https://www.infoq.com/presentations/Twitter-Timeline-Scalability/
- http://highscalability.com/blog/2013/7/8/the-architecture-twitter-uses-to-deal-with-150m-active-users.html