**Twitter**

**Features**

1. Tweet: User should be able to tweet to the users as fast as possible.
   1. Some users may have millions of followers so the tweet must reach them within secs
2. TimeLine:
   1. Home Timeline: Tweets from all the following users.
   2. Search Timeline: Tweets from the search keyword by the user.
   3. User Timeline: Tweets and Retweets the user made.
3. Trending: Topics which are trending around the world.

**Back of the Envelope Calculations**

1. Twitter has about 300M+ users
2. Read: 600,000 tweets/sec
3. Write: 600 tweets/sec
4. 140char / tweet

* With this, we understand that Twitter is read-heavy. For every 1000 reads, there is 1 write.
* Eventual Consistency is fine for Twitter. It's fine if a user gets a tweet after 2secs.

**Thinking Process**

1. We need a system which allows us read faster without loading the system. Also, the system should scale horizontally.
2. We immediately think of REDIS because it is a in memory DB and it scales well horizontally.
3. We cannot solely depend on REDIS only.
4. We can have User Table, Tweet Table and Follower Table

Tweets

Followers

User

1. Redis can store the data like:
   1. <USER\_ID>\_TWEETs: [1,2,3…]
   2. <USER\_ID>\_FOLLOWERS: [1,2,3…]
   3. TWEET\_ID\_TWEET: “Hey there! I am using Twitter”
2. User Timeline:
   1. Go to the user table and get the tweet IDs of your own USER\_ID\_TWEETs. From that, we can get the Tweets by TWEET\_ID\_TWEETS.
   2. We can sort this by Date and Time and then return the data.
3. Home TimeLine:
   1. We need to get the followers, get the latest tweets from them. Merge and Display in chronological order. Will this work? Go to 150-500 followers and get their latest tweets? So many queries just for a single API? This won’t scale well.
   2. We need some different strategy called FAN OUT.
   3. Fan Out: When we get a tweet, do some preprocessing, and distribute the data in such a way that when a user asks for a tweet, you get the home timeline immediately.
   4. When ever a person makes a tweet:
      1. Add an entry into the database.
      2. Add the same tweet into his User Timeline (Redis Array)
      3. **Fan out** this data to all the followers of this user. This step would add this tweet to all the followers’ home timeline (HOME not USER) (Redis Array)
      4. The Home Timeline Redis Array/Cache keeps the tweets of all the followings of a user. Now when a user loads his home timeline, he just needs to access this Redis Cache Array and he is done.
      5. We have reduced the DB query drastically now. But… will this work for everyone?
      6. Consider a case when the user is a celebrity? It is not easy task to update the Home Timeline of 50M followers with that tweet.
      7. **Tweets by celebrity:** When this guy tweets, save the tweet in DB. Add it to his user timeline. That’s it. Now when a user opens his home timeline, get the tweets from “normal people” from his redis queue and get the “celebrity tweets” from the user timeline of the celebrities. This can be done very quickly if we save the “list of celebrities” by each user. We can quickly go through this list and get the latest tweets from the celebrities’ user timeline. Basically we do NOT send the tweet to the follower’s home time line queue.
      8. Do we need to calculate the Home timeline for the users who are not even logging in from 1 month? No. We can save ourselves from that pain and save a lot of data and computation. This is one enhancement that can be done.
4. #Trending:
   1. We need to consider the volume of tweets and the timetaken to generate them.
   2. Consider 1000 tweets in 5 mins vs 10,000 tweets in 1 month. The events can be Election Results, Movie Releases, Sports Events, etc.
   3. Twitter uses stream processing framework like Apache STORM / Heron. We can also use Kafka as well

Redis DB

Count Location

Tweet

Filter

Parser

Geolocation

Rank

Count Hashtags

* 1. All the boxes are connected between with queues (maybe kafka)
  2. Filter: Filters some common hashtags (#fun, #food, etc) and do some violation check.
  3. Parser: It parses the tweets. Twitter needs to figure our by some natural language processing because a user may not have given direct hashtags. Figure out the appropriate hashtags for it.
  4. Now the tweets gets distributed into two for Geo Processing and for Trending.
  5. Geo Location: It is important because some event which is important in US .. may not be important in India. So, we basically find the location to count of hashtag calculation.
  6. Trending: Get the count of the hashtags for a specific amount of time and get the rank for a hashtag.
  7. All the data is fed into Redis and from there the front end can read the trending topics.

1. Search Timeline:
   1. Twitter uses Early Bird. It does Inverted Full text Indexing.
   2. A tweet is broken into meaningful words and hashtags.
   3. These words are indexed into a table which stores the words to tweet mapping.. so when a user searches for something, access the table and search for that word and get all the related tweets. This would be a distributed table.
   4. Scatter and Gather: For distributed computation, you have different nodes scattered across different data centers. When you get a query, send it to all the data centers, the data center replies their result. The search system collates all the results and returns to the user.

**Final Design**

1. Final System Design and Data Flow:
   1. In next Page.

Apache Storm

#hashtag

Database

Twitter Writer

Load Balancer

Zookeeper

Tweet

Redis

Fan Out

Timeline Service

Search Service

HTTP Push Websocket

1. When someone tweets,
   1. it goes to Loadbalancer and then that call hits the twitter writer.
   2. Twitter writer is responsible in writing it to DB and passing it to Storm which will process the trending #tags for you.
   3. A copy of tweet is sent to fanout service which will update the tweet into the followers timelines (home and user timeline) in Redis.
   4. A copy of tweet is sent to search service. These are indexed to make it searchable.
2. When user searches:
   1. It goes to timeline service and it forwards to search service.
   2. It will scatters the request and gather the result from the different data centers.
3. When user asks for home/user timeline:
   1. It also hits LB and then it gets forwarded to timeline service. It goes to Redis and returns the necessary data.
4. HTTP PUSH Websocket:
   1. It handles the realtime, persistent connections to mobile applications. It should handle millions of connections at any point of time.
5. Zookeeper:
   1. It is coordination service for distributed components. Say it has 1000 nodes for redis (very big cluster of redis), then for managing it we have zookeeper.
   2. It helps keeping configuration for each and every node in the redis cluster.
   3. It helps elect a master in the redis cluster.
   4. It tracks which nodes are online and offline.
6. DB:
   1. Twitter uses Geysar which is built on top of MysQL and Inno DB. Most number of queries are not going to DB, so we need not worry a lot about it. But it is the place where all the tweets are kept.