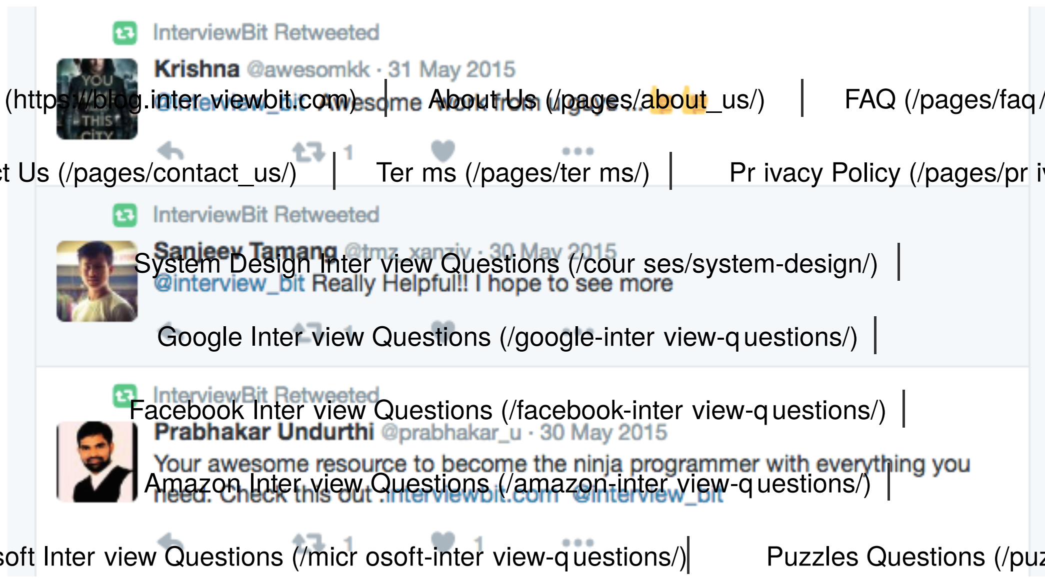
System Design (/cour ses/system-design) / System Design Inter view Questions (/cour ses/system-design/topics/inter view-q uestions/) / Design Twitter

Show Notes

Design Twitter

Bookmark

Let's design a Twitter like system.



Features:

> *This is the first part of any system design interview, coming up with the features which the system should support. As an interviewee, you should try to list down all the features you can think of which our system should support. Try to spend around 2 minutes for this section in the interview. You can use the notes section alongside to remember what you wrote.*

Q: What ar e some of the Twitter featur es we should suppor t?
A: Let's assume that we ar e looking at posting tweets, following people and favor iting tweets. A user should also be able to see a feed of tweets of his/her follower s.

Q: Do we need to suppor t r eplies to tweets / gr ouping tweets by conver sations?
A: Let's assume we don't need to for this case.

Q: How about pr ivacy contr ols ar ound each tweet?
A: Not r equir ed. Let's assume for this case that all tweets ar e public.

Q: Do we need to suppor t tr ending tweets? If so, do we need to suppor t localization and per sonalization?
A: For this case, lets just assume we ar e not focussing on building the tr ending tweets featur e.

Q: How about Dir ect messaging ?
A: No. Lets leave that out for this question. That could be another question by itself.

Q: How about mentions / tagging?
A: Let's assume we don't need to suppor t mentions/tagging.

Q: Do we need to suppor t a notification system?
A: For the pur pose of this question, no.

- **Estimation:**

> *This is usually the second part of a design interview, coming up with the estimated numbers of how scalable our system should be. Important parameters to remember for this section is the number of queries per second and the data which the system will be required to handle.*
> *Try to spend around 5 minutes for this section in the interview.*

Lets estimate the volume of tweets. Assume that our system would be the second most popular tweeting ser vice after Twitter .

Q: What is the number of user s and tr affic that we expect the system to handle?

A: Twitter does ar ound 500 million tweets per day with 100 million daily active user s. Lets assume similar number s.

()

Q: How many follower s does ever y user have?

A: The behavior should be similar to Twitter her e. Each user has on aver age 200 follower s, with cer tain hot user s having a lot mor e follower s. For example, user s like Justin Bieber would have millions of follower s.

()

Q: How many times is a tweet favor ited?

A: Assuming the same behavior as Twitter , we can assume that each tweet is favor ited twice. However , in this case as well, ther e will be outlier s. Ther e ar e cer tain tweets which might be favor ited by millions of people.

()

Q: Assuming the networ k of user s, how many user to follower edge would exist?

A: We had 100 million active user s w th 200 follower s on aver age. This means 100M x 200 edges = 20B edges

()

- **Design Goals:**

> "
> **Latency** - Is this pr oblem ver y latency sensitive (Or in other wor ds, Ar e r eq uests with high latency and a failing r eq uest, eq ually bad?). For example, sear ch typeahead suggestions ar e useless if they take mor e than a second.
> **Consistency** - Does this pr oblem r eq uir e tight consistency? Or is it okay if things ar e eventually consistent?
> **Availability** - Does this pr oblem r eq uir e 100% availability?
>
> *There could be more goals depending on the problem. It's possible that all parameters might be important, and some of them might conflict. In that case, you'd need to prioritize one over the other.* "

Q: Is Latency a ver y impor tant metr ic for us?

A: Yes. A twitter like system needs to be fast, especially when you ar e competing with Twitter .

()

Q: How impor tant is Consistency for us?

A: Not r eally. Assuming a lot of activity on this system, if I miss out on a tweet of a per son I am following ever y now and then, its not the end of the wor ld. Compar e this to dir ect messaging wher e consistency is extr emely impor tant.

()

Q: How impor tant is Availability for us?

A: Yes. If Twitter becomes unavailable, it becomes a news. As a pr oduct, it needs to be highly available.

()

- ## Skeleton of the design:

> *The next step in most cases is to come up with the barebone design of your system, both in terms of API and the overall workflow of a read and write request. Workflow of read/write request here refers to specifying the important components and how they interact. Try to spend around 5 minutes for this section in the interview.*
> ***Important** : Try to gather feedback from the interviewer here to indicate if you are headed in the right direction.*

As discussed befor e, ther e ar e 4 major oper ations we need to suppor t:

Posting new tweets
Following a user
Favor iting a tweet
Get the feed for a user

Q: What would the API look like for the client?

Q: What data would need with ever y Tweet we fetch?

()

A: We should have the content of the tweet, the per son who posted the tweet, the timestamp when tweet was cr eated and number of favor ites for the tweet.

Q: Would we need all the user pr ofles of user s who have favor ited a tweet?

()

A: Given thats a lot of data to fetch, we can be mor e intelligent about it and just fetch top 2 people in the list. In this scheme, we would show ever y tweet as 200 favor ites which on hover shows Favor ited by X, Y and 198 other s

Q: How many tweets should we fetch at a time?

()

A: At a time, only a cer tain number of tweets will be in the viewpor t (lets say 20). Lets call it a page of tweets. For a user , we would only want to fetch a page of tweets at a time.
Gotcha: Would the page size r emain constant acr oss differ ent situations? Pr obably not. The page size would be differ ent acr oss clients based on scr een size and r esolution. For example, a mobile's page size might be lower than that of a web br owser 's.

A: The fir st 3 oper ations end up doing a wr ite to the database. The last oper ation does a r ead. Following is an example of how the API might look like :

```
Posting new tweets : addTweet(userId, tweetContent, timestamp)

Following a user : followUser(userId, toFollowUserId)

Favorite a tweet : favoriteTweet(userId, tweetId)

TweetResult getUserFeed(user, pageNumber, pageSize,
lastUpdatedTimestamp)

where TweetResult has the following fields :

TweetResult {

    List(Tweets) tweets,

    boolean isDeltaUpdate

}

Tweet {

    userId,
```

```
    content,
    timestamp,
    numFavorites,
    sampleFavoriteNames
 }
```

Ther e could be other APIs as well which would help us fetch the most r ecent tweets of a user , or fetch the follower s for a tweet.

()

Q: How would a typical wr ite q uer y (addTweet) look like?

A: Components:
Client (Mobile app / Br owser , etc) which calls addTweet(user Id, tweetContent, timestamp)
Application ser ver which inter pr ets the API call and tr ies to append the tweet to user 's tweet with the timestamp in the database layer .
Database ser ver which appends the tweet

()

Q: How would a typical r ead q uer y (getUser Feed) look like?

A: Components:
Client (Mobile app/Br owser , etc) which calls getUser Feed
Application ser ver which inter pr ets the API call and q uer ies the database for the top user feed.
Database ser ver which looks up the follower s' tweet to get the r esult.

()

- **Deep Dive:**

> *Lets dig deeper into every component one by one. Discussion for this section will take majority of the interview time(20-30 minutes).*

Lets dig deeper into ever y component one by one.

- **Application layer:**

Think about all details/gotchas yourself before beginning.

Q: How would you take car e of application layer fault toler ance?

Q: How do we handle the case wher e our application ser ver dies?

()

A: The simplest thing that could be done her e is to have multiple application ser ver . They do not stor e any data (stateless) and all of them behave the exact same way when up. So, if one of them goes down, we still have other application ser ver s who would keep the site r unning.

Q: How does our client know which application ser ver s to talk to. How does it know which application ser ver s have gone down and which ones ar e still wor king?

()

A: We intr oduce load balancer s. Load balancer s ar e a set of machines (an or der of magnitude lower in number) which tr ack the set of application ser ver s which ar e active (not gone down). Client can send r equest to any of the load balancer s who then for war d the r equest to one of the wor king application ser ver s r andomly.

A: If we have only one application ser ver machine, our whole ser vice would become unavailable. Machines will fail and so will networ k. So, we need to plan for those events. Multiple application ser ver machines along with load balancer is the way to go.

()

- **Database layer:**

This is the hear t of the question. In the skeleton design, we assumed that the database is a black box which can magically stor e or r etr ieve anything efficiently. Lets dig into how we will build that magic black box.

Q: What data do we need to stor e ?

A:
For ever y tweet, we need to stor e content, timestamp and owner ID.
For ever y user , we need to stor e some per sonal infor mation (Name, age, bir thdate, etc.)
We need to stor e all u1->u2 follower r elations.
We need to stor e all user _ids against a tweet of user s who have favor ited the tweet.

()

Q: RDBMS or NoSQL?

Q: Ar e joins r equir ed?

()

A: NoSQL databases ar e inefficient for joins or handling r elations. As such, NoSQL databases stor e ever ything in a denor malized fashion. In this case, we do have r elations like
user -> follower s
tweets -> favor ited by user s

SQL seems to win on this par ameter on ease of use.

Q: How much data would we have to store?

()

A: If the size of the data is so small that it fits on a single machine's main memory, SQL is a clear winner. SQL on a single machine has next to zero maintenance overhead and has great performance with right index built. If your index can fit into RAM, its best to go with a SQL solution. Lets analyze our current case :
Size of tweets :
Number of tweets per day : 500 million
Maximum size of a tweet : 140 chars + 1 byte for timestamp + 1 byte for userId = 142 bytes
Provisioning for : 5 years = 365 * 5 days
Space required : 142bytes * 500M * 365 * 5 = 129.5TB

Size of user - follower relation :
Assuming total of 1 Billion users and every user has 200 followers on average, we end up with 200B total connections. To store it, we would need 200B * 2 bytes (size of 2 user IDs) = 400G.

Size of tweet to favorites relation :
Average number of favorites per tweet : 2 (Ref. Estimations section)
Total number of tweets daily : 500M
Provisioning for : 5 years = 365 * 5 days
Space required : (2 bytes + 1 byte for tweetId) * 500M * 365* 5 = 2.7TB
So, total space required is close to 130TB. That'd definitely not fit on a single machine's hard disk.

Q: How important is technology maturity?

()

A: SQL DBs like MySQL have been around for a long time and have hence been iterated enough to be very stable. However, most NoSQL databases are not mature enough yet. Quoting an article from PInterest Engineering Blog :

> “ We intentionally ran away from auto-scaling newer technology like MongoDB, Cassandra and Membase, because their maturity was simply not far enough along (and they were crashing in spectacular ways on us!).”

A: Things to consider :
Are joins required?
Size of the DB
Technology Maturity

In practice, the score is equal for both RDBMS or NoSQL for this one. In theory, NoSQL would be a better fit.
We can choose either to proceed further. Lets go with a relational DB like MySQL for this one.

()

Q: What would the database schema look like?

A: Always be prepared for this question in cases where the schema might be a bit more elaborate.
We have two main entities: users and tweets. There could be two tables for them. Table users would have personal information about the user. A sample table schema for that could look like the following :

Table **users**

ID (id) - primary key

username (username) - unique key

First Name (first_name)

Last Name (last_name)

password related fields like hash and salt (password_hash & password_salt)

created at (created_at)

Last updated at (updated_at)

Date of Birth (dob)

description (description)

Tweets should be slightly simpler :

Table **tweets**

ID (id) - primary key

content (content)

date of creation (created_at)

user ID of author (user_id)

Now, lets look at the r elations that we need to model :

Follower r elation (User A follows another user B)

Table **connections**

ID of user that follows (follower_id)

ID of user that is followed (followee_id)

date of creation (created_at)

Favor ite : A user can favor ite a tweet.

Table **favories**

ID of user that favorited (user_id)

ID of favorited tweet (tweet_id)

date of creation (created_at)

Now, based on the r ead API quer ies, we can look at the additional index we would need :

Get the feed of a user - This would r equir e us to quickly lookup the user Ids a user follows, get their top tweets and for each tweet get user s who have favor ited the tweet.

This means we need to following index :

An index on follower _id in connections to quickly lookup the user Ids a user follows

An index on user _id, cr eated_at in tweets to get the top tweets for a user (wher e user _id = x sor t by cr eated_at desc)
An index on tweet_id in favor ites

()

Q: Now the bigger question, How would we do shar ding?

Question: Approach1: How can we shar d on user s?

()

A:
Detail : Whats stor ed in each table :
user s : par t of the table with user _ids which belong to the shar d
tweets : par t of the table with author _ids which belong to the shar d (Or in other wor ds, tweets by the user s in the cur r ent shar d)
connections : All entr ies wher e follower _id belongs to the cur r ent shar d
favor ites : All entr ies wher e tweet_id belongs to the tweets table in this shar d

Pros:
Equal load distr ibution
Cheap wr ites : All of the wr ite quer ies ar e simple and r ely on just one shar d (Assuming tweet favor ite API encodes the tweet owner ID in the tweet ID when sending r equest).

Cons:
While looking up the user Ids a user follows is easy on the machine, getting the top tweet for each of those user Ids would r equir e quer ying differ ent shar ds.
Even when we need to favor ite a tweet, finding the tweet would r equir e us to quer y all the shar ds. We can wor k ar ound it, however , by encoding owner _id with the tweet_id fr om the client.

Question: Approach2: Can we shar d on r ecency(timestamp)?

()

A: Shar d on r ecency(timestamp) -

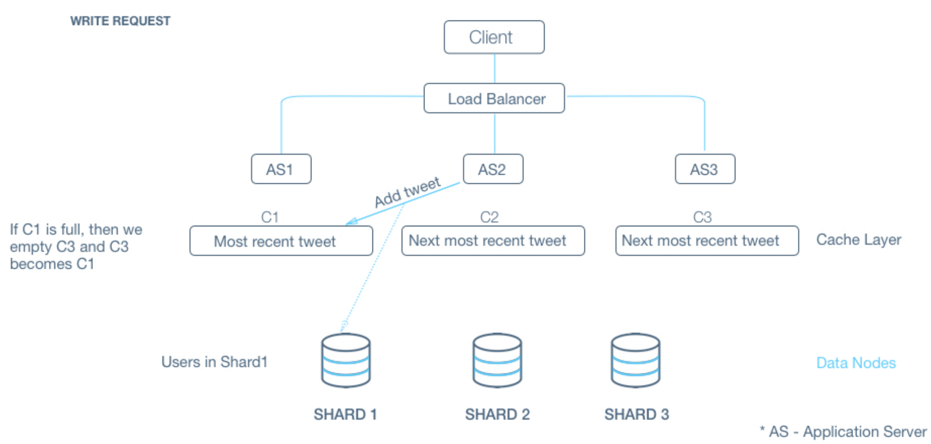
Most recent tweets in the most recent shard. The idea is that most of the time we are only working with most recent tweets. Its rare to dig up tweets which are more than a few weeks old. New tweets are requested most frequently.
users : All of users table resides in one shard separately.
tweets : This table is sharded across shards by recency. When the most recent shard starts getting full, we create a new shard and assign the new incoming tweets to the newly created shard.
connections : All of the table resides in the shard with users.
favorites: Stored with the tweets in their shard

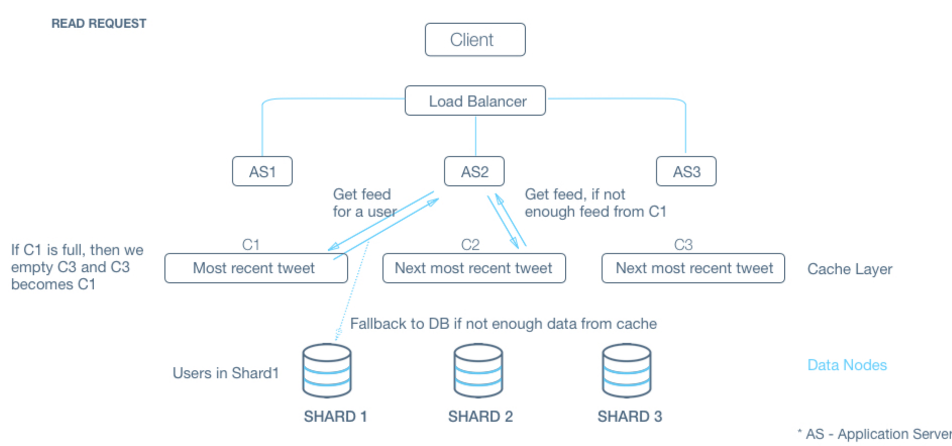
Pros:
Fetching the user feeds requires just querying 2 shards (users and tweets).
More reliable and has low latency. Most of the queries would only interact with 2 shards.

Cons:
Load imbalance : The most recent tweet shard will be handling almost all of the traffic while the other shards will remain idle.
Huge maintenance overhead : Every time, you need to create a new shard, you'll need to allocate new machines, setup replication and make things switch almost instantly so that no downtime is induced. All in all, a nightmare for DBAs at that scale.

A: We have already established earlier that we would need to shard as data would not fit into a single machine. The read query we are optimizing :
Get the feed of a user - This would require us to quickly lookup the user Ids a user follows, get their top tweets and for each tweet get users who have favorited the tweet.
We can model our data with two basic approaches, sharding based on recency of tweet or based on users, we will call these approaches Approach1 and Approach2 respectively. Our answer consists of a hybrid approach of Approach1 and Approach2, so we highly recommend you to go through hints which explains both of them in detail.
Both solutions have their downsides and don't seem ideal. What is a clean solution then? Lets go back to our design goals. We want low latency and high

availability. Consistency is not a big deal for us (If I miss a tweet once in a while in the feed, its not the end of the wor ld).
With that in mind, we can look for a hybr id model. We will definitely need to heavily cache.
We can have a cache which simulates the r ecent shar d in Appr oach 2 and a DB which stor es stuff as in Appr oach 1. The idea being that most r eads will be ser ved by the cache itself and it has the collection of all r ecent tweets. In the r ar e case of not so r ecent tweet, we will go to the DB and in such cases, latency outlier s ar e alr ight. Notice that the DB wr ites would be cheap as discussed in Appr oach 1.





()

Q: Do we need special handling for spiky cases (User s with unusual number of follower s / Tweets with likelihood of getting unusually high number of favor ites) ? Think about the case when Katy Per r y (with mor e than 70M follower s) tweets.

A: Lets look at both cases one by one.
Let's say Katy Per r y tweets. Following is what happens :
We wr ite the tweet to the shar d wher e Katy Per r y belongs. Not a pr oblem.
We add it to the r ecent tweets cache. Again, not a pr oblem.
As all follower s get their feed update by specifically r equesting for an update, the r esultant change is that a lot of followees will get an update when they r equest for it. This should manifest as an uptick in the upload bandwidth. In the wor st case, assuming that 30% of the follower s ar e online at a time, we would need ar ound 3G of upload bandwidth which is a r eally small number for a datacenter .

Lets look at r eally popular tweets now. They'll have an unusually high r ate of being favor ited (The highest being 3M total favor ites). This means a r eally high r ate of wr ite to the shar d which can cause deadlocks. We can add some optimizations her e if r equir ed in ter ms of batching the updates to favor ites table in a queue befor e flushing them. Nitty Gr itty : Would the queue be per sistent? If not, what happens if the machine dies. That would cause data loss. If yes, wher e does the queue r eside? How do you mer ge the quer y r esults?

()

Q: How would we handle a DB machine going down?

A: As stated in design goals, we need to make sur e our system is mor e available at all times.
We had shar ded the database based on user s. We can have a r eplica for each of them which follows the updates happening on the master database shar d. When the master goes down, the slave can take over . Now ther e is a pr oblem her e. What if ther e wer e some updates which the slave had not caught up to yet. Do we lose that infor mation? We can take a call either way. If we ar e par ticular about getting the data back, we know that we can get that infor mation fr om the cache layer and r esolve stuff on the DB layer .

()