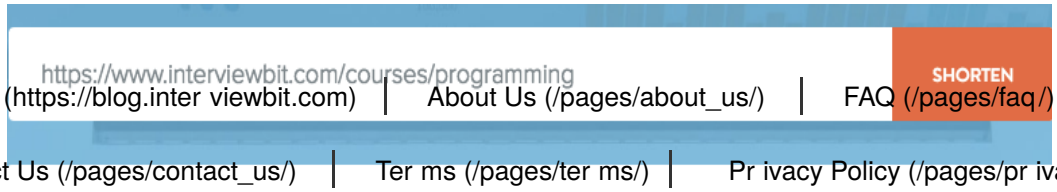
Design a URL shor tening ser vice, like bit.ly



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.*

Shortening: Take a ur l and r etur n a much shor ter ur l.
Ex: http://www.inter viewbit.com/cour ses/pr ogr amming/topics/time-complexity/ =>
http://goo.gl/GUKA8w/

Redirection: Take a shor t ur l and r edir ect to the or iginal ur l.
Ex: http://goo.gl/GUKA8w => http://www.inter viewbit.com/cour ses/pr ogr amming /topics/time-complexity/

Custom url: Allow the user s to pick custom shor tened ur l.
Ex: http://www.inter viewbit.com/cour ses/pr ogr amming/topics/time-complexity/ => http://goo.gl/ib-time

Analytics: Usage statistics for site owner .
Ex: How many people clicked the shor tened ur l in the last day?
Gotcha: What if two people tr y to shor ten the same URL?

● 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. ”*

Q: How many quer ies per second should the system handle?(Assuming 100 Million new URLs added each month)

Hint: Assuming aver age lifetime of a shor tened URL is 2 weeks and 20% of websites cr eates 80% of the tr affic, we see that we'll r eceive ar ound 1 Billion quer ies in a month.

A: 400 quer ies per second in total. 360 r ead quer ies and 40 wr ite quer ies per second.

Q: How much data will we need to stor e so that we don't have to r estr uctur e our ar chitectur e for the next 5 year s consider ing constant gr owth r ate?

Q: How many URLs will we need to handle in the next 5 year s?

()

Hint: Ear lier we saw, we would see 100 Million new URLs each month. Assuming same gr owth r ate for next 5 year s, total URLs we will need to shor ten will be 100 Million * 12 * 5 = 6 Billion.
A: 6 Billion.

Q: What is the minimum length of shor tened ur l to r epr esent 6 Billion URLs?

()

Hint: We will use (a-z, A-Z, 0-9) to encode our URLs. If we call x as minimum number of char acter s to r epr esent 6 Billion total URLs, then will be the smallest integer such that x^62 > 6*10^9.
A: Log ($6*10^9$) to the base 62 = 6

A: 3 Ter aBytes for URLs and 36 GigaBytes for shor tened URLs
Note that for the shor tened URL, we will only stor e the slug(6 char s) and compute the actual URL on the fly.

()

Q: Data r ead/wr itten each second?

Hint: Data flow for each r equest would contain a shor tened URL and the or iginal URL. Shor tened URL as we saw ear lier , will take 6 bytes wher eas the or iginal URL can be assumed to take atmost 500 bytes.

A: Wr itten : 40 * (500 + 6) bytes, Read : 360 * (500 + 6) bytes

()

- **Design Goals:**

> “
>
> **Latency** - Is this pr oblem ver y latency sensitive (Or in other wor ds, Ar e r equests with high latency and a failing r equest, equally bad?). For example, sear ch typeahead suggestions ar e useless if they take mor e than a second.
> **Consistency** - Does this pr oblem r equir e tight consistency? Or is it okay if things ar e eventually consistent?
> **Availability** - Does this pr oblem r equir 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. Our system is similar to DNS r esolution, higher latency on URL shor tener is as good as a failur e to r esolve.

()

Q: Should we choose Consistency or Availability for our ser vice?

A: This is a tr icky one. Both ar e extr emenly impor tant. However , CAP theor em dictates that we choose one. Do we want a system that always answer s cor r ectly but is not available sometimes? Or else, do we want a system which is always available but can sometime say that a URL does not exists even if it does?
This tr adeoff is a pr oduct decision ar ound what we ar e tr ying to optimize. Let's say, we go with consistency her e.

()

Q: Tr y to list down other design goals?

A: URL shor tener by definition needs to be as shor t as possible. Shor ter the shor tened URL, better it compar es to competition.

()

- **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. ”*

Q: How should we define our APIs?

A:

```
ShorteningAPI(url) {store the url_mapping and return hash(url)}

RedirectionAPI(hash) {redirect_to url_mapping[hash]}
```

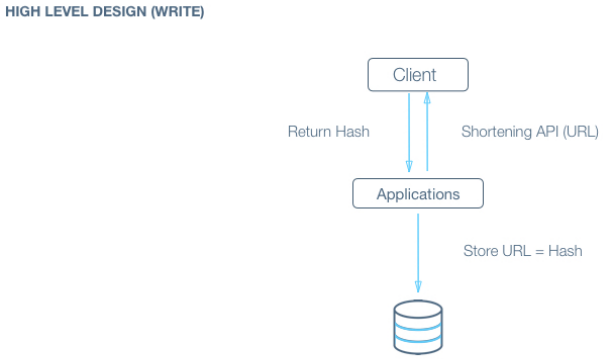
Both APIs ar e ver y lightweight, their computation will not the bottleneck plus we don't have to stor e any session infor mation about user s.
Basically, we ar e tr ying to build a ser vice which ser ves as a huge HashMap

()

Q: How would a typical wr ite quer y look like?

A: Components:
Client (Mobile app / Br owser , etc) which calls ShorteningAPI(url)
Application ser ver which inter pr ets the API call and gener ates the shor tened hash for the ur l

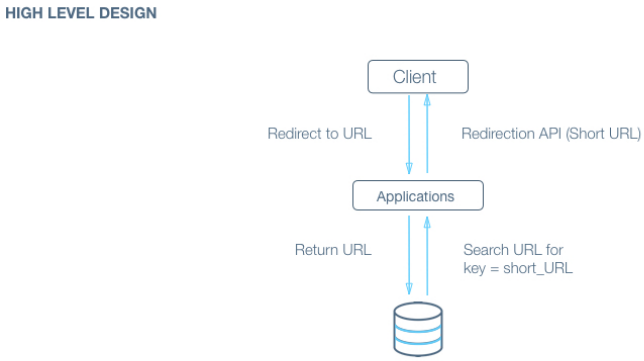
Database ser ver which stor es the hash => ur l mapping



()

Q: How would a typical r ead q uer y look like?

A: Components:
Client (Mobile app / Br owser , etc) which calls Redir ectionAPI(shor t_ur l)
Application ser ver which inter pr ets the API call, extr acts the hash fr om shor t_ur l, asks database ser ver for ur l cor r esponding to hash and r etur ns the ur l.
Database ser ver which stor es the hash => ur l mapping



- **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). ”*

Q: How should we compute the hash of a URL?

Gotcha: How should we handle the case wher e two separ ate URL gets shor tened to the same URL?

()

A: We can use a list of salts in case of collision.
For each r ead r equest, we can compute all possible shor tened URLs using our list of salts and quer y for them in par allel to save time.

A: convert_to_base_62(md5(original_url + salt))[:6](first six characters)

Links: MD5-Wiki (https://en.wikipedia.or g/wiki/MD5) Base 62 Conver sion-Stackover flow (http://stackover flow.com/questions/1119722/base-62-conver sion-in-python)

Gotchas:

Dir ectly encoding URL to base_62 will allow a user to check if a URL has been shor tened alr eady or not, r ever se enginner ing can lead the user to the exact hash function used, this should not be allowed. Ther efor e r andomization has to be intr oduced. Also, if two user s shor tens the same URL, it should r esult in two separ ate shor tened URL(for analytics)

Database ID encoded to base_62 also won't be suitable for a pr oduction envir onment because it will leak infor mation about the database. For example, patter ns can be lear nt to compute the gr owth r ate(new URLs added per day) and in the wor st case, copy the whole database.

Got suggestions ? We would love to hear your feedback.

Loved InterviewBit ? Write u testimonial. ()

(http://www.quora.com/What-is-your-review-of-InterviewBit)

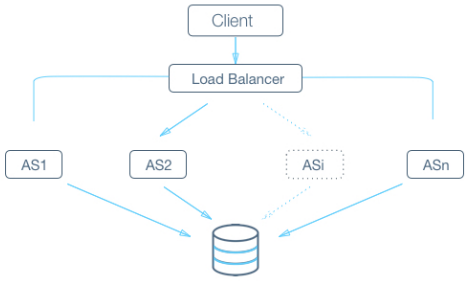
Q: How would you take car e of application layer fault toler ance?

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.

HIGH LEVEL DESIGN



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 eq uest to any of the load balancer s who then for war d the r eq uest 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.

()

Q: What all data should we stor e?

A: Data stor age layer : Hash => URL mapping.
Billions of small sized(1kb) object. Ther e is no r elationship between objects.
We would also need to stor e data for analytics, for example, how many times was the ur l opened in the last hour ?

()

Q: Should we choose RDBMS or NOSQL?

Hint : Things to consider :
Ar e joins r eq uir ed :
NoSQL databases ar e ineffcient for joins.
In this case, assuming we don't need analytics, we only need to answer the quer y "Given a hash, give me the cor r esponding URL" - a standar d key to value lookup. As such, we don't need any joins her e.
Size of the DB :
If the size of the data is so small that it fits on a single machine's main memor y, SQL is a clear winner . SQL on a single machine has next to zer o maintenance over head and has gr eat per for mance with r ight index built. If your index can fit into RAM, its best to go with a SQL solution. Lets analyze our cur r ent case :

Assumptions:
# of wr ites per month: 100 Million (Refer to estimations section)
Avg size of URL : 500 bytes
Pr ovisioning for : 5 year s
Space r eq uir ed : 500 bytes * 100M * 12 * 5 = 3TB

3TB of data can fit on a single machine's har d disk. However , the index might not. If we stor e all data on a single machine, our wr ite and r ead oper ations would be ver y slow (Page swaps for indices). Given r ead latency is cr itical for us, we can't stor e the data on a single machine.
So, a SQL solution will have a shar ding over head. Most NoSQL solutions however ar e built with the assumption that the data does not fit on a single machine and

hence have shar ding builtin.

A: As discussed in hints, NoSQL would be a better fit for our case.
Since we ar e optimizing for consistency over availability, we will choose a NoSQL DB which is highly consistent like HBase.

()

Q: What would the database schema look like?

A: We want to stor e the mapping fr om hash -> URL which is ideal for a key value stor e. In NoSQL domain, we need to be car eful when designing the key as entr ies ar e indexed by the key.
In our case, since we will only quer y by hash which will never update, our key will be hash with the value being the URL.

()

Q: How would we do shar ding?

A: HBase inher ently use consistent hashing for shar ding. We explain it in detail at https://www.inter viewbit.com/pr oblems/shar ding-a-database/ (https://www.inter viewbit.com/pr oblems/shar ding-a-database/). Since our total data size is a few TBs, we don't need to think about shar ding acr oss datacentr es either .

()

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

A: NoSQL again has single machine failur e handling inbuilt. A NoSQL DB like HBase would have an availability issue for a few seconds for the affected keys, as it tr ies to maintain tight consistency.

()

Bonus Exercise :

Q: How can we optimize r ead q uer ies?

A: Caching can be used to r educe aver age r ead time. Since most URL's ar e accessed for only a small amount of time after its cr eation, LRU cache fits our use case. Also, if a shor tened URL goes vir al, ser ving it thr ough a cache will r educe the chances over loading the database. Redis and Memcached ar e examples of well known and widely used caches.
Redis-Tutor ial (http://tr y.r edis.io/) is an inter active tutor ial for Redis.
Redis vs Memcached (http://stackover flow.com/q uestions/10558465 /memcached-vs-r edis) lists down compar ison between them.

()

Q: How would you shar d the data if you wer e wor king with SQL DB?

Hint: Consistent Hashing - Stackover flow (http://stackover flow.com/q uestions /144360/simple-basic-explanation-of-a-distr ibuted-hash-table-dht)

A: We can use integer encoding of the shor tened URL to distr ibute data among our DB shar ds.
Assuming we assign values fr om 0 to 61 to char acter s a to z, A to Z, and 0 to 9, we can compute the integer encoding of the shor tened URL.
We can see that the maximum value of integer encoding will be less than 10^13, which we can divide among our DB shar ds.
We will use consistent hashing to ensur e we don't have to r ehash all the data again if we add new DB shar ds later .

()

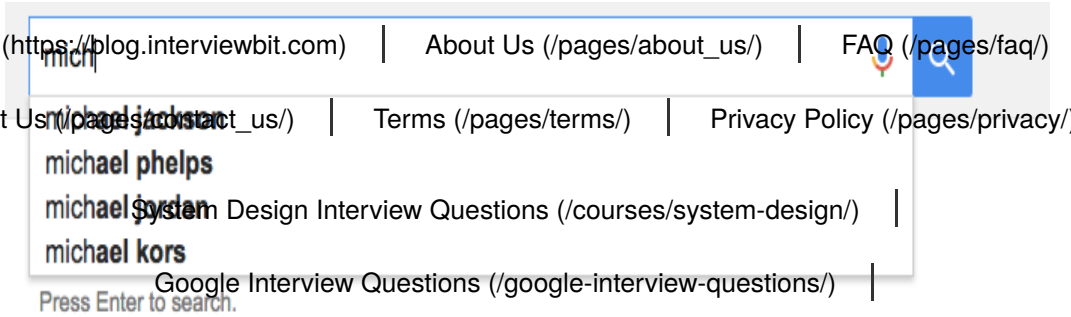
Q: How would you handle machine dying in the case of SQL DB?

A: This is a bit tr icky. Obviously, for ever y shar d, we need to have mor e than one machine
We can have a scheme better known as master slave scheme, wher ein ther e is one machine(master) which pr ocesses all wr ites and ther e is a slave machine which just subscr ibes to all of the wr ites (http://www.quora.com/What-is-your-that the master goes down, the slave can take over and star t r esponding to the

Design Search Typeahead

Bookmark

Design a search typeahead (Search autocomplete) system at Google's scale.



Blog (https://blog.interviewbit.com) | About Us (/pages/about_us/) | FAQ (/pages/faq/) |
Contact Us (/pages/contact_us/) | Terms (/pages/terms/) | Privacy Policy (/pages/privacy/)
System Design Interview Questions (/courses/system-design/) |
Google Interview Questions (/google-interview-questions/) |
Facebook Interview Questions (/facebook-interview-questions/) |
Amazon Interview Questions (/amazon-interview-questions/) |
Microsoft Interview Questions (/microsoft-interview-questions/) | Puzzles Questions (/puzzles/)
Like Us (https://www.facebook.com/interviewbit) Follow Us (https://twitter.com/interview_bit)
Email (mailto:hello@interviewbit.com)

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: How many typeahead suggestions are to be provided?

A: Let's assume 5 for this case.

Q: Do we need to account for spelling mistakes ?
A: Example : Should typing *mik* give michael as a suggestion because michael is really popular as a query?

Q: What is the criteria for choosing the 5 suggestions ?
A: As the question suggests, all suggestions should have the typed phrase/query as the strict prefix. Now amongst those, the most relevant would be the most popular 5. Here, popularity of a query can be determined by the frequency of the query being searched in the past.

Q: Does the system need to be realtime (For example, recent popular events like “Germany wins the FIFA worldcup” starts showing up in results within minutes).
A: Let's assume that it needs to be realtime.

Q: Do we need to support personalization with the suggestions? (My interests / queries affect the search suggestions shown to me).
A: Let's assume that we don’t need to support personalization

- **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. ”*

Clients can query my system for top 5 suggestions given a query prefix.

Every search query done should feed into the system for an update.

Lets estimate the volume of each.

Q: How many search queries are done per day?

A: Assuming the scale of Google, we can expect around 2-4 Billion queries per day.

()

Q: How many queries per second should the system handle?

A: We can use the estimation from the last question here.
Total Number of queries : 4 Billion
Average length of query : 5 words = 25 letters (Since avg length of english word is 5 letters).
Assuming, every single keystroke results in a typeahead query, we are looking at an upper bound of 4 x 25 = 100 Billion queries per day.

()

Q: How much data would we need to store?

A: Lets first look at the amount of new data we generate every day. 15% of the search queries are new for Google (~500 Million new queries). Assuming 25 letters on average per query, we will 12.5G new data per day.
Assuming, we have accumulated queries over the last 10 years, the size would be 12.5 * 365 * 10 G which is approximately 50TB.

()

Design Goals:

> “
>
> **Latency** - Is this problem very latency sensitive (Or in other words, Are requests with high latency and a failing request, equally bad?). For example, search typeahead suggestions are useless if they take more than a second.
> **Consistency** - Does this problem require tight consistency? Or is it okay if things are eventually consistent?
> **Availability** - Does this problem require 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 very important metric for us?

A: A big Yes. Search typeahead almost competes with typing speed and hence needs to have a really low latency.

()

Q: How important is Consistency for us?

A: Not really important. If 2 people see different top 5 suggestions which are on the same scale of popularity, its not the end of the world. I, as a product owner, am happy as long as the results become eventually consistent.

()

Q: How important is Availability for us?

A: Very important. If search typeahead is not available, the site would still keep working. However, it will lead to a much degraded experience.

()

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 before, there are essentially 2 parts to this system :

Given a query, give me 5 most frequent search terms with the query as strict prefix
Given a search term, update the frequencies.

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

A:

```
Read: List(string) getTopSuggestions(string currentQuery)
Write: void updateSuggestions(string searchTerm)
```

()

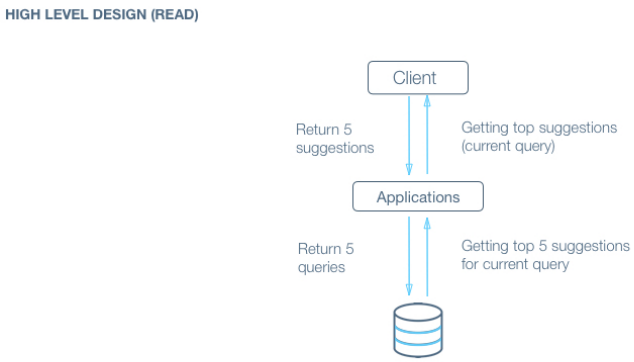
Q: What is a good data structure to store my search queries so that I can quickly retrieve the top 5 most popular queries?

A: For this question, we need to figure out top queries with another string as strict prefix. If you have dealt with enough string questions, you would realize a prefix tree (or trie) would be a perfect fit here.
Devil however lies in the details. We will dig deeper into the nitty gritty of this in the next section.

()

Q: How would a typical read query look like?

A: Components:
Client (Mobile app / Browser, etc) which calls getTopSuggestions(currentQuery)
Application server which interprets the API call and queries the database for the corresponding top 5 queries.
Database server which looks up the top queries in the trie.

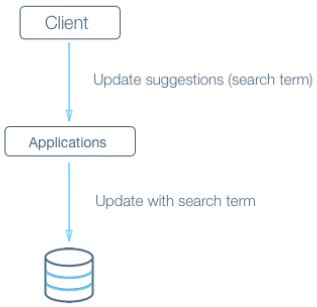


()

Q: How would a typical write query look like?

A: Components:
Client (Mobile app / Browser, etc) which calls updateSuggestions(searchTerm)
Application server which interprets the API call and forwards the searchTerm to database for update.
Database server which updates its trie using the searchTerm

HIGH LEVEL DESIGN



()

- **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 every component one by one.

Application layer:

Think about all details/gotchas yourself before beginning.

Q: How would you take care of application layer fault tolerance?

Q: How do we handle the case where our application server dies?

()

A: The simplest thing that could be done here is to have multiple application server. They do not store 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 servers who would keep the site running.

Q: How does our client know which application servers to talk to. How does it know which application servers have gone down and which ones are still working?

()

A: We introduce load balancers. Load balancers are a set of machines (an order of magnitude lower in number) which track the set of application servers which are active (not gone down). Client can send request to any of the load balancers who then forward the request to one of the working application servers randomly.

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

()

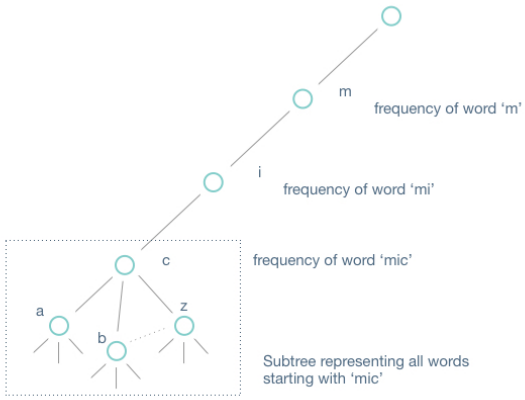
Database layer:

Let's first dig deeper into the trie we talked about earlier.

Q: How would a read query on the trie work?

A: The read query would require us to fetch the top 5 results per query. A traditional trie would store the frequency of the search term ending on the node n1 at n1. In such a trie, how do we get the 5 most frequent queries which have the search term as strict prefix. Obviously, all such frequent queries would be the terms in the subtree under n1 (as shown in diagram).
So, a bruteforce way is to scan all the nodes in the subtree and find the 5 most frequent. Lets estimate the number of nodes we will have to scan this way. Let's say, the user just typed in letter 'a'. In such a case, we would end up scanning all queries which begin with 'a'. As we discussed earlier this would mean scanning terabytes of data which is clearly very time-consuming and inefficient.

Also, the high latency does not align with our design goals.

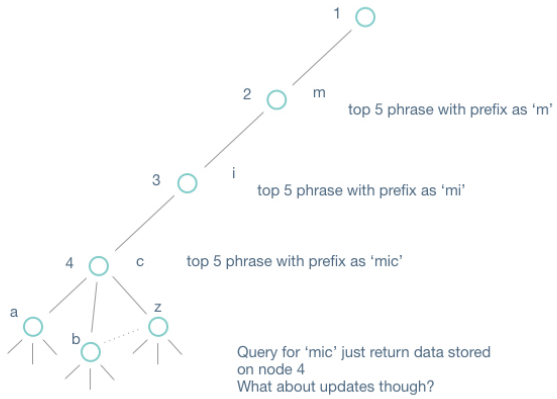


()

Q: How can we modify the trie so that reads become super efficient?

Hint : Store more data on every node of the trie.

A: Storage is cheap. Lets say we were allowed to store more stuff on each node. How would we use the extra storage to reduce the latency of answering the query.
A good choice would be storing the top 5 queries for the prefix ending on node n1 at n1 itself. So, every node has the top 5 search terms from the subtree below it. The read operation becomes fairly simple now. Given a search prefix, we traverse down to the corresponding node and return the top 5 queries stored in that node.



()

Q: How would a typical write work in this trie?

A: So, now whenever we get an actual search term, we will traverse down to the node corresponding to it and increase its frequency. But wait, we are not done yet. We store the top 5 queries in each node. Its possible that this particular search query jumped into the top 5 queries of a few other nodes. We need to update the top 5 queries of those nodes then. How do we do it then? Truthfully, we need to know the frequencies of the top 5 queries (of every node in the path from root to the node) to decide if this query becomes a part of the top 5.
There are 2 ways we could achieve this.

Along with the top 5 on every node, we also store their frequency. Anytime, a node's frequency gets updated, we traverse back from the node to its parent till we reach the root. For every parent, we check if the current query is part of the top 5. If so, we replace the corresponding frequency with the updated frequency. If not, we check if the current query's frequency is high enough to be a part of the top 5. If so, we update the top 5 with frequency.
On every node, we store the top pointer to the end node of the 5 most frequent queries (pointers instead of the text). The update process would involve comparing the current query's frequency with the 5th lowest node's frequency and update the node pointer with the current query pointer if the new frequency is greater.

Q: Can frequent writes affect read efficiency?

A: Yes, potentially. If we are updating the top 5 queries or the frequencies very frequently, we will need to take a lock on the node to make sure the reader thread does not get an inconsistent value. As such, writes start to compete with reads.

()

Q: What optimizations can we do to improve read efficiency?

Q: Can we use sampling?

()

A: Yes. If we assume Google's scale, most frequent queries would appear 100s of times in an hour. As such instead of using every query to update, we can sample 1 in 100 or 1 in 1000 query and update the trie using that.

Q: Offline update?

()

A: Again if we assume that most queries appearing in the search typeahead would appear 100s of times in an hour, we can have an offline hashmap which keeps maintaining a map from query to frequency. Its only when the frequency becomes a multiple of a threshold that we go and update the query in the trie with the new frequency. The hashmap being a separate datastore would not collide with the actual trie for reads.

A: As mentioned earlier, writes compete with read. Sampling writes and Offline updates can be used to improve read efficieny.

()

Q: What if I use a separate trie for updates and copy it over to the active one

periodically?

A: Not really, there are 2 major problems with this approach.
You are not realtime anymore. Lets say you copy over the trie every hour. Its possible a search term became very popular and it wasn't reflected for an hour because it was present in the offline trie and did not appear till it was copied to the original trie
The trie is humungous. Copying over the trie can't be an atomic operation. As such, how would you make sure that reads are still consistent while still processing incoming writes?

()

Q: Would all data fit on a single machine?

A: Refer to estimations section. We would need to store more than 50TB of data.
Ideally, we would want most of it in memory to help with the latency. Thats a lot to ask from a single machine. We will go with a "No" here.

()

Q: Alright, how do we shard the data then?

Q: Would we only shard on the first level?

()

A: The number of shards could very well be more than the number of branches on first level(26). We will need to be more intelligent than just sharding on first level.

Q: What is the downside of assigning one branch to a different shard?

()

A: Load imbalance. Storage imbalance. Some letters are more frequent than the others. For example, letters starting with 'a' are more likely than

letters starting with 'x'. As such, we can run into cases of certain shards running hot on load. Also, certain shards will have to store more data because there are more queries starting with a certain letter. Another fact in favor of sharding a little more intelligently.

A: Lets say we were sharding till the second or third level and we optimize for load here. Lets also say that we have the data around the expected load for every prefix.
We keep traversing the 2 letter prefixes in order ('a', 'aa', 'ab', 'ac',...) and break when the total load exceeds an threshold load and assign that range to a shard. We will need to have a master which has this mapping with it, so that it can route a prefix query to the correct shard.

 ()

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

A: As we discussed earlier, availability is more important to us than consistency. If thats the case, we can maintain multiple replica of each shard and an update goes to all replicas. The read can go to multiple replicas (not necessarily all) and uses the first response it gets. If a replica goes down, reads and writes continue to work fine as there are other replicas to serve the queries.
The issue occurs when this replica comes back up. There are 2 options here :
If the frequency of the replica going down is lower or we have much higher number of replicas, the replica which comes back up can read the whole data from one of the older working replica while keeping the new incoming writes in a queue.
There is a queue with every server which contains the changelog or the exact write query being sent to them. The replica can request any of the other replicas in its shard for all changelog since a particular timestamp and use that to update its trie.

 ()

You have now mastered this problem!

Discussion

Post a comment (https://discuss.interviewbit.com/session/sso?return_path=https://discus

T

thecoder2016 about 1 year ago

What would be the schema for storing the trie ? Would we use SQL or no-SQL databa 2

Reply (https://discuss.interviewbit.com/session/sso?return_path=https://discuss.interviewbit.com/t/what-would-be-the-schema-for-storing-the-trie-would-we-use-sql-or-no-sql-databa)

H

himanshu_setia over 1 year ago

But How would we store Trie in DB? 2

Reply (https://discuss.interviewbit.com/session/sso?return_path=https://discuss.interviewbit.com/t/but-how-would-we-store-trie-in-db)

D

Got suggestions ? We would love to hear your feedback.

Loved InterviewBit ? Write us a testimonial. (http://www.quora.com/What-is-your-review-of-InterviewBit)

Highly Consistent Database

Bookmark

Design a distr ibuted key value stor e which is highly consistent and is networ k par tition toler ant.

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 is the amount of data that we need to stor e?
Answer: Let's assume a few 100 TB.

Q: Do we need to suppor t updates?
A: Yes.

Q: Can the size of the value for a key incr ease with updates?
A: Yes. In other wor ds, its possible a sequence of keys could co-exist on one ser ver pr eviously, but with time, they gr ew to a size wher e all of them don't fit on a single machine.

Q: Can a value be so big that it does not fit on a single machine?
A: No. Let's assume that there is an upper cap of 1GB to the size of the value.

Q: What would the estimated QPS be for this DB?
A: Let's assume around 100k

- **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."*

Total storage size : 100 TB as estimated earlier
Total estimated QPS : Around 10M

Q: What is the minimum number of machines required to store the data?

A: Assuming a machine has 10TB of hard disk, we would need minimum of 100TB / 10 TB = 10 machines to store the said data. Do note that this is barebe minimum. The actual number might be higher if we decide to have replication or more machines incase we need more shards to lower the QPS load on every shard.

()

- **Design Goals:**

> “
>
> **Latency** - Is this problem very latency sensitive (Or in other words, Are requests with high latency and a failing request, equally bad?). For example, search typeahead suggestions are useless if they take more than a second.
> **Consistency** - Does this problem require tight consistency? Or is it okay if things are eventually consistent?
> **Availability** - Does this problem require 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 very important metric for us?

A: No, but it would be good to have a lower latency.

()

Q: Consistency vs Availability?

A: As the question states, we need tight consistency and partitioning. Going by the CAP theorem (Nicely explained at http://rohergr einer.com/2014/08/cap-theorem-revisited/ (http://rohergr einer.com/2014/08/cap-theorem-revisited/)), we would need to compromise with availability if we have tight consistency and partitioning. As is the case with any storage system, data loss is not acceptable.

()

- **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).* ”

> “ Note : In questions like these, the interviewer is looking at how you approach designing a solution. So, saying that I’ll use a NoSQL DB like HBase is not an ideal answer. It is okay to discuss the architecture of HBase for example with rationale around why some components were designed the way they were.”

Q: Is shar ding r equir ed?

A: Lets look at our ear lier estimate about the data to be stor ed. 100TB of data can’t be stor ed on a single machine.
Let's say that we somehow have a r eally beefy machine which can stor e that amount of data, that machine would have to handle all of the quer ies (All of the load) which could lead to a significant per for mance hit.

> “ Tip: You could argue that there can be multiple copies of the same machine, but this would not scale in the future. As my data grows, its possible that I might not find a big beefy enough machine to fit my data. ”

So, the best cour se of action would be to shar d the data and distr ibute the load amongst multiple machines.

()

Q: Should the data stor ed be nor malized?
http://www.studytonight.com/dbms/database-nor malization.php
(http://www.studytonight.com/dbms/database-nor malization.php)

Q: Can I shar d the data so that all the data r equir ed for answer ing my most fr equent quer ies live on a single machine?

()

A: Most applications ar e built to stor e data for a user (consider messaging for example. Ever y user has his / her own mailbox). As such, if you shar d based on ever y user as a r ow, its okay to stor e data in a denor malized fashion so that you won't have to quer y infor mation acr oss user s. In this case, lets say we go with stor ing data in denor malized fashion.

A: If the data is nor malized, then we need to join acr oss tables and acr oss r ows to fetch data. If the data is alr eady shar ded acr oss machine, any join acr oss machines is highly undesir able (High latency, Less indexing suppor t).
With stor ing denor malized infor mation however , we would be stor ing the same fields at mor e than one place. However , all infor mation r elated to a r ow (or a key) would be on the same machine. This would lead to lower latency.
However , if the shar ding cr iter ia is not chosen pr oper ly, it could lead to consistency concer ns (After all, we ar e stor ing the same data at multiple places).

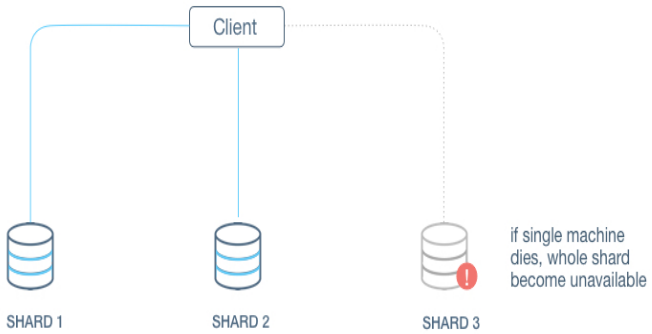
()

Q: How many machines per shar d ? How does a r ead / wr ite look in ever y shar d?

Q: Can we keep just one copy of data?

()

A: Since ther e is only one copy of the data, r eading it should be consistent. As long as ther e ar e enough shar d to ensur e a r easonable load on each shar d, latency should be acceptable as well. Reads and wr ites would wor k exactly how they wor k with a single DB just that ther e would be a r ow -> shar d -> machine IP (Given a r ow, tell me the shar d it belongs to and then given the shar d, give me the machine I should be quer ying / wr iting to) r esolution layer in between.
Ther e is just one tiny pr oblem with this model. What if the machine in the shar d goes down? Our shar d will be unavailable (which is fine as gover ned by the CAP theor em). However , what if the machine dies and its har d disk becomes cor r upt. We suddenly r un into the r isk of losing the data which is not acceptable. Imagine losing all your messages because your shar d went down and the har d disk got cor r upted. That means we definitely need mor e than one copy of data being wr itten with us.



Q: What pr oblem may ar ise if we keep multiple copies of data?

()

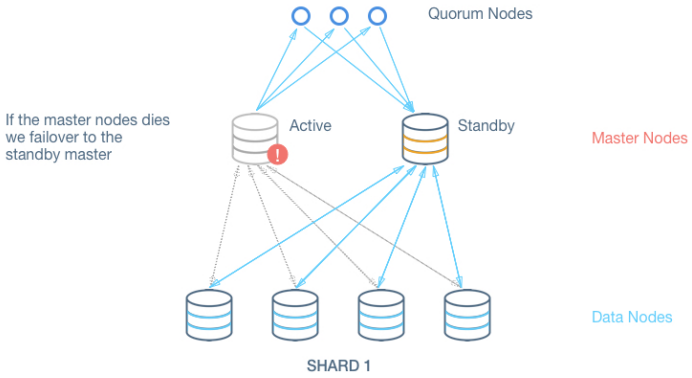
A: Let's say we keep 3 copies of data (The pr obability of all 3 machines dying and having a cor r upted disk is negligible). Now, the issue is how do we maintain all of the copies in absolute sync (Consistency, r emember ?). One naive way would be that a wr ite would not succeed unless its wr itten to all 3 copies / machines. That would make our wr ite latency go up significantly apar t fr om making wr ites ver y unr eliable (My wr ite fails if it fails on any of the machines). Let's see if we can make this a bit better .
If we have to allow wr ites succeeding even if the wr ite has been wr itten on a major ity of machines (2 out of 3, let's say), to maintain consistency, its impor tant that ther e is a master machine which keeps tr ack of this infor mation. This master machine can tr ack which machines have a par ticular block in each shar d. This means that ever y r ead will go thr ough this master machine, figur e out the machines with the block and quer y fr om the r equir ed block. The machines which do not have the block can check with this master machine to see which block ar e not pr esent on it, and catch up to r eplicate the block on it.

However , now if this master machine dies, our whole shar d is unavailable till this machine comes back up. If this machine has a cor r upted har d disk, then the unavailability becomes indefinite (Note that we do not loose data in this case, as total data is the union of data pr esent on 3 nodes). This is not an ideal design, but we will talk about improvements to it later in this question.

Q: What if the master keeping tr ack of wher e the blocks ar e stor ed dies?

()

Anwer: To over come this pr oblem we keep a standby master which in the failover pr ocess becomes the acting master and keeps unavilability to minimum. Now, to keep the standby master upto date we can have a shar ed networ k file system. When any namespace modification is per for med by the Active master , it dur ably logs a r ecor d of the modification to an edit log file stor ed in the shar ed dir ector y. The Standby node constantly watches this dir ector y for edits, and when edits occur , the Standby node applies them to its own namespace. In the event of a failover , the Standby will ensur e that it has r ead all of the edits fr om the shar ed stor age befor e pr omoting itself to the Active state. This ensur es that the namespace state is fully synchr onized befor e a failover occur s.



A: Going back to our design goals, latency and consistency ar e our design goals.

A simple way to r esolve this is to make sur e we only have one machine per shar d. Reads and wr ites would wor k exactly how they wor k with a single DB. However , if the machine holding the only copy dies and its har d disk becomes cor r upt, we suddenly r un into the r isk of losing the data which is not acceptable. That means we definitely need mor e than one copy of data being wr itten with us. Lets say that number is 3. Now, the issue is how do we maintain all of the copies in absolute sync (Consistency, r emember ?).

One naive way would be that a wr ite would not succeed unless its wr itten to all 3 copies / machines. That would make our wr ite latency go up significantly apar t

fr om making wr ites ver y unr eliable (My wr ite fails if it fails on any of the machines).
If we have to allow wr ites succeeding when the wr ite has been wr itten on a major ity of machines (2 out of 3, lets say), to maintain consistency, its impor tant that ther e is a master machine which keeps tr ack of this infor mation. This master machine can tr ack which machines have a par ticular block in each shar d. However , now if this master machine dies, our whole shar d is unavailable till this machine comes back up. If this machine has a cor r upted har d disk, then the unavailability becomes indefinite.
Ther e ar e couple of ways to keep unavailability to minimum using a standby master keeping tr ack of master node data thr ough a shar ed file system(Explained in detail in the last hint).

()

You have now mastered this problem!

Discussion

Post a comment (https://discuss.interviewbit.com/session/sso?return_path=https://discus

P

piscado over 1 year ago

What are the cons if we use last modified time(solution given in highly available 3

Reply (https://discuss.inter viewbit.com/session/sso?r etur n_path=https://discuss.inter viewbit.com/t/what-ar e-the-cons-if-we-use-last-modified-time-solution-given-in-highly-available)

K

Got suggestions ? We would love to hear your feedback.

Loved InterviewBit ? Write us a testimonial. (http://www.quora.com/What-is-your-review-of-InterviewBit)

Highly Available Database

Bookmark

Design a distr ibuted key value stor e which is highly available and is networ k par tition toler ant

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 is the amount of data that we need to stor e?
A: Let's assume a few 100 TB.

Q: Do we need to suppor t updates?
A: Yes.

Q: Can the size of the value for a key incr ease with updates?
A: Yes. In other wor ds, its possible a sequence of keys could co-exist on one ser ver pr eviously, but with time, they gr ew to a size wher e all of them don't fit on a single machine.

Q: Can a value be so big that it does not fit on a single machine?
A: No. Let's assume that there is an upper cap of 1GB to the size of the value.

Q: What would the estimated QPS be for this DB?
A: Let's assume around 100k.

- **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."*

Total storage size : 100 TB as estimated earlier
Total estimated QPS : Around 100k

Q: What is the minimum number of machines required to store the data?

A: Assuming a machine has 10TB of hard disk, we would need minimum of 100TB / 10 TB = 10 machines to store the said data. Do note that this is barebe minimum. The actual number might be higher if we decide to have replication or more machines incase we need more shards to lower the QPS load on every shard.

()

- **Design Goals:**

> “
>
> **Latency** - Is this pr oblem ver y latency sensitive (Or in other wor ds, Ar e r equests with high latency and a failing r equest, equally bad?). For example, sear ch typeahead suggestions ar e useless if they take mor e than a second.
> **Consistency** - Does this pr oblem r equir e tight consistency? Or is it okay if things ar e eventually consistent?
> **Availability** - Does this pr oblem r equir 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: Since we want to be available all the time, we should tr y to have lower latency.

()

Q: Consistency vs Availability?

A: As the question states, we need good availability and par tition toler ance. Going by the CAP theor em (Nicely explained at http://r ober tgr einer .com /2014/08/cap-theor em-r evisited/ (http://r ober tgr einer .com/2014/08/cap-theor em-r evisited/)), we would need to compr omise with consistency if we have availability and par tition toler ance.
We can however aim at having eventual consistency. As is the case with any stor age system, data loss is not acceptable.

()

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).*

> Note : In questions like these, the interviewer is looking at how you approach designing a solution. So, saying that I'll use a NoSQL DB like Cassandra is not an ideal answer. It is okay to discuss the architecture of Cassandra for example with rationale around why some components were designed the way they were..

Q: Is sharding required?

A: Lets look at our earlier estimate about the data to be stored. 100TB of data can't be stored on a single machine.
Lets say that we somehow have a really beefy machine which can store that amount of data, that machine would have to handle all of the queries (All of the load) which could lead to a significant performance hit.

> Tip: You could argue that there can be multiple copies of the same machine, but this would not scale in the future. As my data grows, its possible that I might not find a big beefy enough machine to fit my data.

So, the best course of action would be to shard the data and distribute the load amongst multiple machines.

()

Q: Should the data stored be normalized?
(http://www.studytonight.com/dbms/database-normalization.php (http://www.studytonight.com/dbms/database-normalization.php))

A: If the data is normalized, then we need to join across tables and across rows to fetch data. If the data is already sharded across machine, any join across machines is highly undesirable (High latency, Less indexing support).
With storing denormalized information however, we would be storing the same fields at more than one place. However , all information related to a row (or a key) would be on the same machine. This would lead to lower latency.

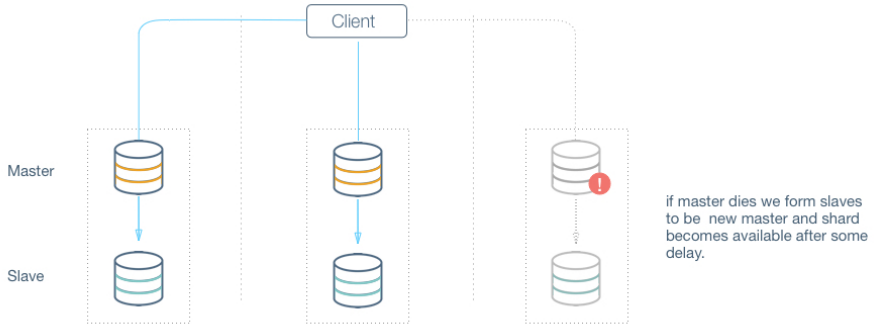
However , if the shar ding cr iter ia is not chosen pr oper ly, it could lead to consistency concer ns (After all, we ar e stor ing the same data at multiple places). However , for this case, we ar e mor e concer ned with availability and r eady to compr omise on consistency as long as things become eventually consistent.
In this case, it seems like having denor malized r ows makes shar ding easier for us and suits our use case better .

()

Q: How many machines per shar d ? How does a r ead / wr ite look in ever y shar d?

A: Going back to our design goals, low latency and high availability ar e our design goals.
Lets assume we have somehow shar ded the r ows into shar ds. Hence, lets fir st look at how the ar chitectur e might look at within a shar d.

Master Slave



One simple solution might be to have a master node in each shar d which has a slave node which r eads all new updates fr om master and keeps updating itself (The slave in this case might not have the same view as master and would lag a little bit). Clients can r ead fr om either the master or the slave depending on which r esponds ear lier (or being slightly mor e intelligent with the r eads to give mor e pr efer ence to the master , and fallback to slave after the r ead call to master). That could lead to inconsistent views on newer entr ies acr oss master and client, but would ensur e high r ead availability.
However , what happens to wr ites when the master goes down. The wr ites star t failing since only master was taking up the wr ites.

We can ar gue that we can have the slave become the new master in such a case. However , even that implies unavailability for the per iod of failover fr om master to the slave as new master .
Also, if the slave is lagging a bit, and then the master has a har dwar e failur e, we r un the r isk of losing data.

This means that we definitely need mor e than one machine taking the wr ite tr affic if we ar e to be available all the time.

Multi Master

Lets say we modify the pr evious design wher e both machines accept wr ite AND r ead tr affic. Lets name the machine m1 and m2.
If m1 accepts wr ite without depending on m2, then it is possible m1 is doing wr ite on a r ow state which is not the same as m2. That could lead to huge consistency concer ns and the DB might become for ever inconsistent. DBs might get oper ations out of or der and it can cause eventual inconsistency if the or der of oper ation matter s (double the column value and then incr ement it vs incr ement the column value and then double it).

Fr om above examples we see that any system with a*master* node will not be highly available, ther efor e we move to peer to peer systems.

()

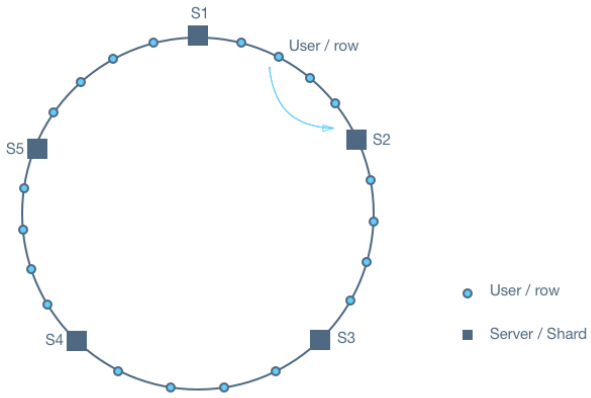
Q: Can a peer to peer system be highly available in case of a DB machine dying?

Hint: Single point of failur e!

A: We defne a peer to peer system wher e ever y node is eq ually pr ivileged and any two nodes can communicate. Yes, since we don't have a single point of failur e anymor e, ther efor e our system can theor etically be available even in pr esence of dying DB machines. Dynamo and Cassandr a ar e examples of examples of such systems, both of them lack the *master* node and ther efor e have no single point of failur e. Our highly available datastor e will be highly based on Dynamo and Cassandr a, as a r eader you don't need to know about them.

Q: How will we exactly shar d data for a peer to peer system?

A: Refer to https://www.inter viewbit.com/pr oblems/shar ding-a-database/ (https://www.inter viewbit.com/pr oblems/shar ding-a-database/) for a detailed answer .



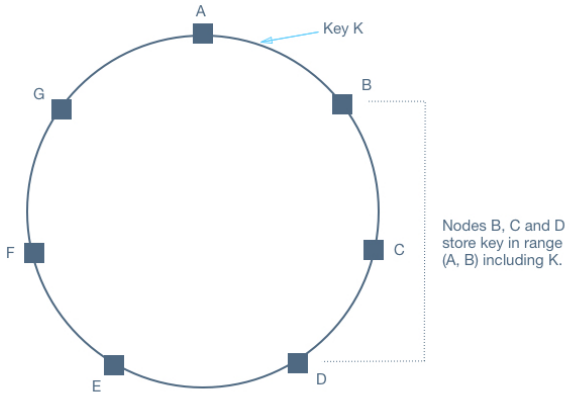
()

Q: How do we stor e r edundant data?

A: We will need to stor e duplicate data to pr event data loss in case of some of our DB machines getting cor r upted. To stor e the data we can use consistent hashing which assigns ever y data to a par ticular node on the r ing. Let's call as our r eplication factor (we will stor *P* copies of data). Now for a data D, we have to choose *P* nodes wher e we will stor e copies of D.
Now, how do we choose these P nodes? We will choose the *P* clockwise consecutive nodes star ting fr om the node wher e D is mapped by our hashing function.
An impor tant point to dicuss her e is that even though any data might be mapped to a par ticular vir tual node, the vir tual node is not the master node for this data for either r ead or r ight. A client can r eq uest to r ead or wr ite a data fr om any node they want. This is essential in cr eating a highly available system.

Got suggestions ? We would love to hear your feedback.

Loved InterviewBit ? Write us a testimonial. (http://www.quora.com/What-is-your-review-of-InterviewBit)



()

Q: How does a wr ite/r ead happen in our system?

A:

Write request:

A client can contact any node in the r ing with a put() r eq uest to wr ite data, this node acts as a coor dinating node for this r eq uest. Coor dinating node then for war ds the r eq uest to the mapping nodes for the data(hashing) and waits for acknowledgement fr om them. When it r eceives **W**(explained later) acknowledgements, it r etur ns a wr ite-accepted message to the client.

Read request:

To per for m a get() r eq uest, client can connect to any node in the r ing which becomes the coor dinating node for this r eq uest. The coor dinating node then asks all r eplica nodes for this data and r etur ns consolidated data to the client when **R** of them r eplies back.

Read and Write consistency:

W and **R** ar e called wr ite and r ead consistency number r espectively. To r ecap, **W** is the minimum number of nodes fr om which the coor dinating node should get an ack befor e making a wr ite successful and **R** is the minimum number of nodes fr om which the coor dinating node should get back r ead values to r etur n them back to the client.

R, W together for ms q uor um of the system. For a r ead to be consistent(r etur n the latest wr ite), we need to keep **W + R > P**.

Depending on the featur e r equir ement W and R can be adjusted, for example to have ver y fast wr ites we can keep **W = 1** and **R = P**. If our system is r ead heavy we can keep **R = 1** and **W = P**. If r ead and wr ite ar e equally distr ibuted, we can keep both R and W as (P+1)/2.

()

Q: What if a machine goes down?

A: Since no node is only r esponsible for a piece of data, it's going down won't r eally affect our wr ites/r eads. As long as **W** out of P nodes ar e available for some key, it can be updated(similar ily for r ead).
Note that in case of less than **W** nodes available to wr ite for some data, we can r elax our wr ite consistency(sacr ificing data consistency for availability).

()

Q: What kind of consistency can we pr ovide?

A: If we keep **W = P**, we can pr ovide str ong consistency but we won't be available for some wr ites even if one of our DB machine dies.
Ear lier we saw in master -master configur ation that in networ k par tition cases, our master s may diver ge to pr ovide availability. In our cur r ent system, essentially all of our nodes ar e master and the point that they will diver ge should be taken for gr anted and we should build our system consider ing it.
Ther efor e we should build for the case wher e **W** is less than P, hence our wr ites will be pr opagated i.e. some machines will have an updated view of data and some will not, ther efor e they ar e not consistent. The best we can guar entee her e is eventual consistency, that is in due time, all the changes will be applied to ever y ser ver and in the end they will all have a consistent view of the data.

To achieve eventual consistency, we need to be able to r esolve differ ences between data on two ser ver s. Ther e ar e a couple of detect and r esolve data conflicts that may ar ise.
Fir st, if data(key, value) we stor e is such that value is just a single column, we can use a simple cr iter ia of LWW(last wr ite wins) to r esolve conflicts. So if two ser ver s have differ ent view of a key, in the r esolve step we can update the ser ver with the stale with the new data and ther efor e become consistent.

The other way is to stor e augmented data for each r ow indicating all the coor dinating nodes for the r ow till now. Now, to detect and under stand conflict we can compar e the augmented data. If one is a subset of the other (all the wr ites seen by one of the r ow has been seen by the other r ow) we can safely ignor e the one with smaller augmented data. Other wise, we have a conflit for our r ow and need application level logic to r esolve it. This way is usually r eq uir ed when our value if composed of mor e than one independent column.

()

You have now mastered this problem!

Discussion

Post a comment (https://discuss.interviewbit.com/session/sso?return_path=https://discus

H

henry_henry almost 2 year s ago

How exactly do peer-to-peer systems work? Doesn't there still have to be a high-l 2

Reply (https://discuss.inter viewbit.com/session/sso?r etur n_path=https://discuss.inter viewbit.com/t/how-exactly-do-peer -to-peer -systems-wor k-doesnt-ther e-still-have-to-be-a-high-l)

S

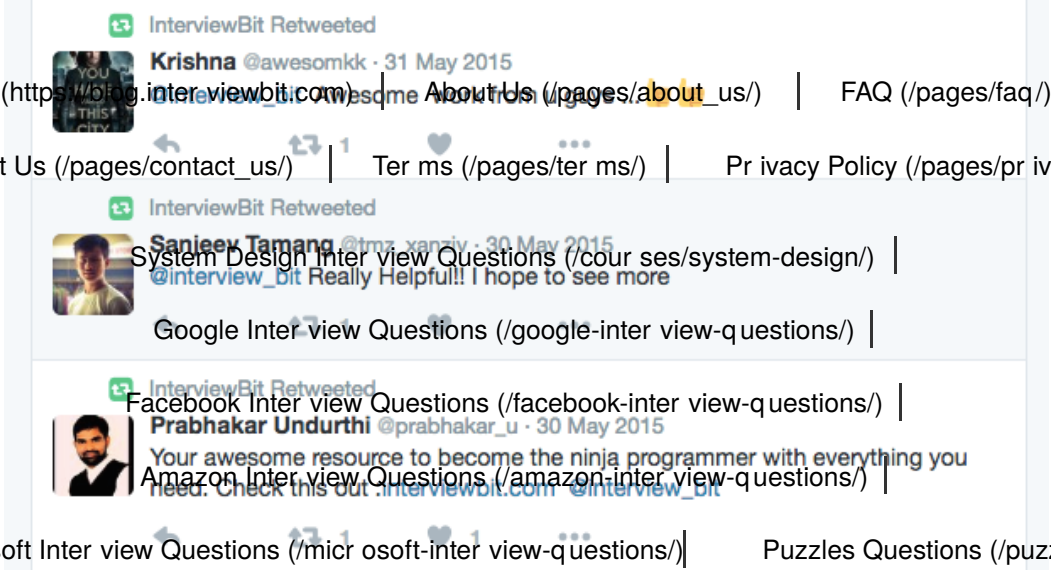
Got suggestions ? We would love to hear your feedback.

Loved InterviewBit ? Write us a testimonial. (http://www.quora.com/What-is-your-review-of-InterviewBit)

Design Twitter

Bookmark

Let's design a Twitter like system.



Blog (https://blog.inter viewbit.com) | About Us (/pages/about_us/) | FAQ (/pages/faq/) |
Contact Us (/pages/contact_us/) | Ter ms (/pages/ter ms/) | Pr ivacy Policy (/pages/pr ivacy/)
System Design Inter view Questions (/cour ses/system-design/) |
Google Inter view Questions (/google-inter view-q uestions/) |
Facebook Inter view Questions (/facebook-inter view-q uestions/) |
Amazon Inter view Questions (/amazon-inter view-q uestions/) |
Micr osoft Inter view Questions (/micr osoft-inter view-q uestions/) | Puzzles Questions (/puzzles/)

Like Us (https://www.facebook.com/inter viewbit) Follow Us (https://twitter .com/inter view_bit)
Email (mailto:hello@inter viewbit.com)

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 are some of the Twitter features we should support?
A: Let's assume that we are looking at posting tweets, following people and favoriting tweets. A user should also be able to see a feed of tweets of his/her followers.

Q: Do we need to support replies to tweets / grouping tweets by conversations?
A: Let's assume we don't need to for this case.

Q: How about privacy controls around each tweet?
A: Not required. Let's assume for this case that all tweets are public.

Q: Do we need to support trending tweets? If so, do we need to support localization and personalization?
A: For this case, lets just assume we are not focussing on building the trending tweets feature.

Q: How about Direct 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 support mentions/tagging.

Q: Do we need to support a notification system?
A: For the purpose 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 equests with high latency and a failing r equest, equally bad?). For example, sear ch typeahead suggestions ar e useless if they take mor e than a second.
> **Consistency** - Does this pr oblem r equir e tight consistency? Or is it okay if things ar e eventually consistent?
> **Availability** - Does this pr oblem r equir 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 ofiles 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 quer 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 quer 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 quer 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 q uestion. 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 eq uir 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 stor e?

()

A: If the size of the data is so small that it fits on a single machine's main memor y, SQL is a clear winner . SQL on a single machine has next to zer o maintenance over head and has gr eat per for mance with r ight index built. If your index can fit into RAM, its best to go with a SQL solution. Lets analyze our cur r ent case :
Size of tweets :
Number of tweets per day : 500 million
Maximum size of a tweet : 140 char s + 1 byte for timestamp + 1 byte for user Id = 142 bytes
Pr ovisioning for : 5 year s = 365 * 5 days
Space r eq uir ed : 142bytes * 500M * 365 * 5 = 129.5TB

Size of user - follower r elation :
Assuming total of 1 Billion user s and ever y user has 200 follower s on aver age, we end up with 200B total connections. To stor e it, we would need 200B * 2 bytes (size of 2 user IDs) = 400G.

Size of tweet to favor ites r elation :
Aver age number of favor ites per tweet : 2 (Ref. Estimations section)
Total number of tweets daily : 500M
Pr ovisioning for : 5 year s = 365 * 5 days
Space r eq uir ed : (2 bytes + 1 byte for tweetId) * 500M * 365* 5 = 2.7TB
So, total space r eq uir ed is close to 130TB. That'd definitely not fit on a single machine's har d disk.

Q: How impor tant is technology matur ity?

()

A: SQL DBs like MySQL have been ar ound for a long time and have hence been iter ated enough to be ver y stable. However , most NoSQL databases ar e not matur e enough yet. Quoting an ar ticle fr om PInter est Engineer ing 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 :
Ar e joins r equir ed?
Size of the DB
Technology Matur ity

In pr actice, the scor e is equal for both RDBMS or NoSQL for this one. In theor y, NoSQL would be a better fit.
We can choose either to pr oceed fur ther . Lets go with a r elational DB like MySQL for this one.

Q: What would the database schema look like?

A: Always be pr epar ed for this question in cases wher e the schema might be a bit mor e elabor ate.
We have two main entities: user s and tweets. Ther e could be two tables for them. Table user s would have per sonal infor mation 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 q uer ies, we can look at the additional index we would need :

Get the feed of a user - This would r equir e us to q uickly 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 would like to have an 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) -

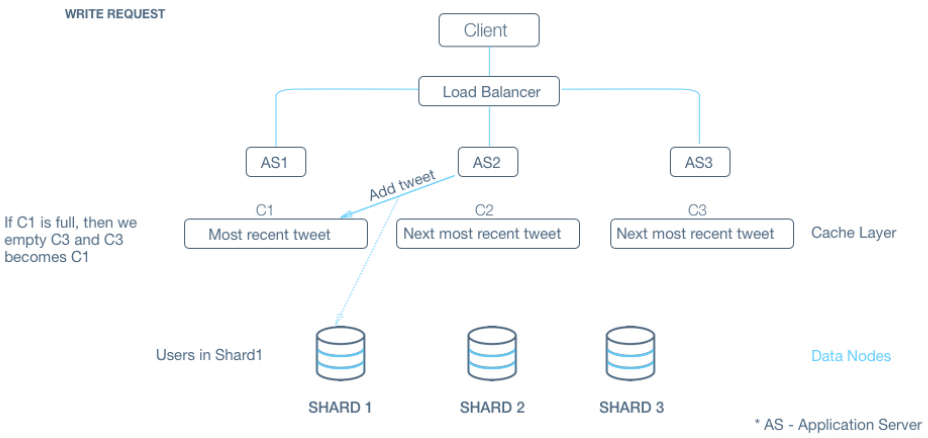
Most r ecent tweets in the most r ecent shar d. The idea is that most of the time we ar e only wor king with most r ecent tweets. Its r ar e to dig up tweets which ar e mor e than a few weeks old. New tweets ar e r equested most fr equently.
user s : All of user s table r esides in one shar d separ ately.
tweets : This table is shar ded acr oss shar ds by r ecency. When the most r ecent shar d star ts getting full, we cr eate a new shar d and assign the new incoming tweets to the newly cr eated shar d.
connections : All of the table r esides in the shar d with user s.
favor ites: Stor ed with the tweets in their shar d

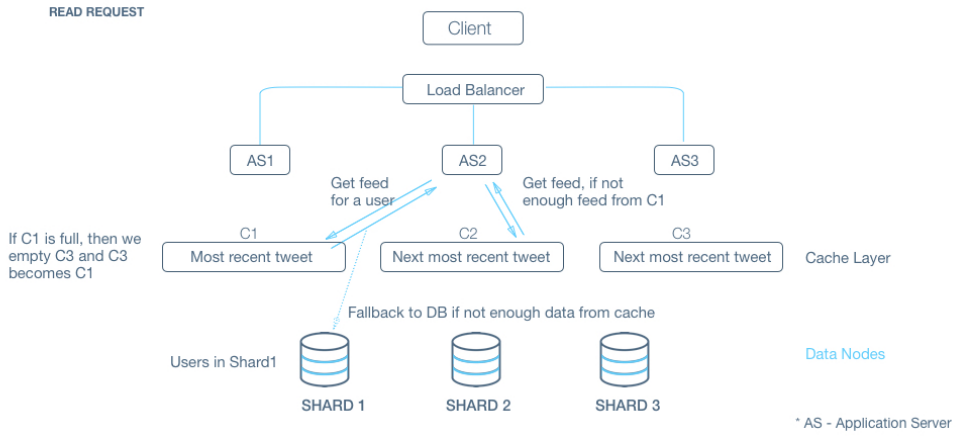
Pros:
Fetching the user feeds r equir es just quer ying 2 shar ds (user s and tweets). Mor e r eliable and has low latency. Most of the quer ies would only inter act with 2 shar ds.

Cons:
Load imbalance : The most r ecent tweet shar d will be handling almost all of the tr affic while the other shar ds will r emain idle.
Huge maintenance over head : Ever y time, you need to cr eate a new shar d, you'll need to allocate new machines, setup r eplication and make things switch almost instantly so that no downtime is induced. All in all, a nightmar e for DBAs at that scale.

A: We have alr eady established ear lier that we would need to shar d as data would not fit into a single machine. The r ead quer y we ar e optimizing :
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.
We can model our data with two basic appr oaches, shar ding based on r ecency of tweet or based on user s, we will call these appr oaches Appr oach1 and Appr oach2 r espectively. Our answer consists of a hybr id appr oach of Appr oach1 and Appr oach2, so we highly r ecommend you to go thr ough 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.





()

Got suggestions ? We would love to hear your feedback.

Loved InterviewBit ? Write us a testimonial. (http://www.quora.com/What-is-your-review-of-InterviewBit)

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 .

()

Design Messenger

Bookmark

Design a messaging service, like Facebook Messenger.

Blog (https://blog.interviewbit.com) | About Us (/pages/about_us/) | FAQ (/pages/faq/) |
Contact Us (/pages/contact_us/) | Terms (/pages/terms/) | Privacy Policy (/pages/privacy/)

System Design Interview Questions (/courses/system-design/) |
Google Interview Questions (/google-interview-questions/) |
Facebook Interview Questions (/facebook-interview-questions/) |
Amazon Interview Questions (/amazon-interview-questions/) |
Microsoft Interview Questions (/microsoft-interview-questions/) | Puzzles Questions (/puzzles/)

Like Us (https://www.facebook.com/interviewbit) Follow Us (https://twitter.com/interview_bit)
Email (mailto:hello@interviewbit.com)

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 is the scale that we are looking at?
A: Let's assume the scale of Facebook Messages. Let's say we need to hand around 10B message sends a day and around 300M users.

Q: Do we only need to support 1:1 conversations or group conversations as well?
A: Let's assume we are building things just for 1:1 conversations. We will extend it to group conversations if need be.

Q: Do we need to support attachments?
A: For now, let's assume that we don't. We will only look at building plain-text messaging system.

Q: What is a reasonable limit to the size of a message?
A: Let's assume that we are building a chat messaging system. As such, we would expect every message to be shorter in length. We can impose a limit here on the maximum size of such a message. Let's say we will only handle messages less than 64Kb in size and reject the others.

Q: What about the notification system for new messages received?
A: Considering the size of the discussion here (45 mins in the interview), we will not delve into the notification system for messages.

- **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. ”*

Let's estimate the volume of each. Assume that our system would be the one of the most popular messaging service.

Q: Given the number of messages being sent, what is the amount of message sent data size we are generating everyday?

A: Number of message sends : 10B
Assuming each message on average has 160 characters , that results in 10B * 160 = 1.6TB assuming no message metadata.

()

Q: What is the expected storage size?

A: From the previous section, we know that we generate 1.6TB data everyday if we only store one copy of the message. If we were to provision for 10 years, we are looking at 1.6 * 365 * 10 TB which is approximately 6 Petabytes.

()

Design Goals:

> “
>
> **Latency** - Is this problem very latency sensitive (Or in other words, Are requests with high latency and a failing request, equally bad?). For example, search typeahead suggestions are useless if they take more than a second.
> **Consistency** - Does this problem require tight consistency? Or is it okay if things are eventually consistent?
> **Availability** - Does this problem require 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 very important metric for us?

A: Yes. Chat is supposed to be realtime, and hence the end to end time actually matters.

()

Q: How important is Consistency for us?

A: Definitely, yes. Its not okay if someone sends me a sequence of message and I don't see some of them. That could lead to huge confusion. Think of cases when you miss an emergency message or missed messages cause misunderstanding between individuals.

()

Q: How important is Availability for us?

A: Availability is good to have. If we had to choose between consistency and availability, consistency wins.

()

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. ”*

Q: What are the operations that we need to support?

A:
Send a message to another person
For a user, fetch the most recent conversations
For every conversation, fetch the most recent messages

()

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

Q: How would the *sendMessage* API look like?

()

A: Send Message : Things to take care of in this API
sendMessage should be idempotent. If a client retries the message, the message should not be added twice. We can resolve this by generating a random timestamp based ID on the client which can be used to de-duplicate the same message being sent repeatedly.
Ordering of messages should be maintained. If I send message A, and then send message B, then A should always appear before B. However, it is possible that due to delays, if two messages are sent quickly one after another, then the requests reach the DB out of order. How do we solve such a case? Obviously, we need to resolve based on the timestamp when they

were sent at.
Timestamp on the client is always unreliable. So, we would need to record the timestamp the first time the request hits the servers (Need not be a part of the API to the client)

```
sendMessage(senderId, recepientId, messageContent,
clientMessageId)
```

Q: How would the API for fetching user's latest conversation look like?

A: This API would be called if I need to show a page of conversations/threads (Think of the view you see when you open the Whatsapp / Messenger app).
At a time, only a certain number of conversations will be in the viewport (let's say 20). Let's call it a page of conversations. For a user, we would only want to fetch a page of conversations at a time.
Gotcha: Would the page size remain constant across different situations? Probably not. The page size would be different across clients based on screen size and resolution. For example, a mobile's page size might be lower than that of a web browser's.
Delta fetch: In most cases, our API calls will be made by users who are active on the site. As such, they already have a view of conversations till a certain timestamp and are only looking for updates after the timestamp (which would typically be 0-2 more conversations). For clients which are data sensitive (like mobile), fetching the whole page every time even when I have all of the conversations can be draining. So, we need to support a way of fetching only the updates when the lastFetchedTimestamp is closer to currentTimestamp.

Keeping the above 2 facts in mind, following is how a hybrid API might look like :

```
ConversationResult fetchConversation(userId, pageNumber,
pageSize, lastUpdatedTimestamp)
```

where ConversationResult is the following :

```
ConversationResult {
```

```
    List(Conversation) conversations,
    boolean isDeltaUpdate
}
Conversation {
    conversationId,
    participants,
    snippet,
    lastUpdatedTimestamp
}
```

Q: How would the API for fetching most recent messages in a conversation look like?

A: Fetch most recent message in a conversation :
This API is almost identical to the fetchConversation API.

```
MessageResult fetchMessages(userId, pageNumber, pageSize,
lastUpdatedTimestamp)
```

where MessageResult has the following fields :

```
MessageResult {
    List(Message) messages,
    boolean isDeltaUpdate
}
Message {
    messageId,
    senderId,
    participants,
    messageContent,
    sentTimestamp
}
```

A: The first and last operation ends up doing a write to the database. The other operations are purely read operations. Following is how API's may look like:
Send Message:

```
sendMessage(senderId, recepientId, messageContent,
clientMessageId)
```

Conversations of a user :

```
ConversationResult fetchConversation(userId, pageNumber,
pageSize, lastUpdatedTimestamp)
```

where ConversationResult has the following fields :

```
ConversationResult {
    List(Conversation) conversations,
    boolean isDeltaUpdate
}
Conversation {
    conversationId,
    participants,
    snippet,
    lastUpdatedTimestamp
}
```

Fetch most recent message in a conversation : This API is almost identical to the fetchConversation API.

```
MessageResult fetchMessages(userId, pageNumber, pageSize,
lastUpdatedTimestamp)
```

where MessageResult has the following fields :

```
MessageResult {
    List(Message) messages,
    boolean isDeltaUpdate
}
Message {
    messageId,
    senderId,
```

```
    participants,
    messageContent,
    sentTimestamp
}
```

Q: How would a typical write query look like?

A: Components:
Client (Mobile app / Browser, etc) which calls sendMessage(senderId, recepientId, messageContent, clientMessageId)
Application server which interprets the API call and calls DB to do the following:
Puts in the serverTimestamp
Figures out the conversation to which the message should be appended based on the other participant
Figures out if a recent message exists with the clientMessageId
Store the message
Database server which stores the message.

Q: How would a typical read query look like?

A: Components:
Client (Mobile app/Browser, etc) which calls fetchConversation
Application server which interprets the API call and queries the database for the top conversation.
Database server which looks up the user's conversations.

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). ”*

Let's dig deeper into every component one by one.

Application layer:

Think about all details/gotchas yourself before beginning.

Q: How would you take care of application layer fault tolerance?

Q: How do we handle the case where our application server dies?

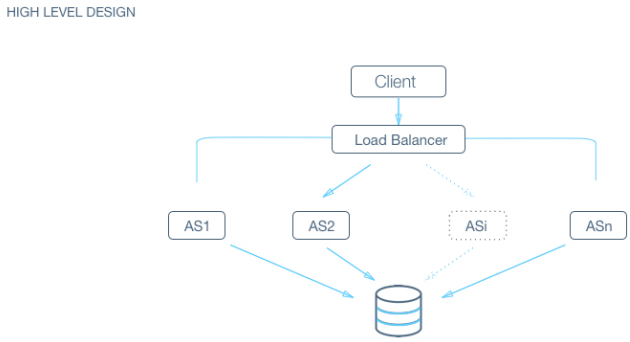
()

A: The simplest thing that could be done here is to have multiple application server. They do not store 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 servers who would keep the site running.

Q: How does our client know which application servers to talk to. How does it know which application servers have gone down and which ones are still working?

()

A: We introduce load balancers. Load balancers are a set of machines (an order of magnitude lower in number) which track the set of application servers which are active (not gone down). Client can send request to any of the load balancers who then forward the request to one of the working application servers randomly.



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

()

Database layer

This is the heart of the question. In the skeleton design, we assumed that the database is a black box which can magically store or retrieve anything efficiently. Let's dig into how we will build that magic black box.

Q: RDBMS or NoSQL?

Q: Are joins required?

()

A: NoSQL databases are inefficient for joins or handling relations. As such, NoSQL databases store everything in a denormalized fashion. In this case, we do have relations like
user -> messages
user -> conversations
conversation -> messages

SQL seems to win on this parameter 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. In our earlier estimations, we had already established that we will need to provision petabytes of data which most definitely does not fit on a single machine. So, a SQL solution will have a sharding overhead. Most NoSQL solutions however are built with the assumption that the data does not fit on a single machine and hence have sharding builtin. NoSQL wins on this parameter.

Q: What is the read-write pattern?

 ()

A: Messaging is going to be very write heavy. Unlike photos or tweets or posts which are written once and then consumed a lot of times by a lot of people, messaging is written once and consumed by the other participant once.
For a write heavy system with a lot of data, RDBMS usually don't perform well. Every write is not just an append to a table but also an update to multiple index which might require locking and hence might interfere with reads and other writes.
However, there are NoSQL DBs like HBase where writes are cheaper. NoSQL seems to win here.

A: Things to consider :
Are joins required?
Size of the DB
Technology Maturity
From the looks of it, NoSQL seems like a better fit. Let's proceed with NoSQL for

now.

Q: How would we store the data? Discuss schema

A: As discussed before, with NoSQL, we need to store the data in denormalized form.
First thing first, this means that every user would have his/her own copy of the mailbox. That means that we will store 2 copies of the message, one for each participant for every message send.
Let's delve into how the schema would look. We'll assume that we are using HBase for this problem.
If this is your first time designing schema, we strongly recommend you go through a primer here (https://www.mapr.com/blog/guidelines-hbase-schema-design).
For schema design, its good to recognize our access patterns. To achieve that, let's look at our major operations :
For a user, append a message to a conversation
Fetch timestamp ordered conversations for a user (Most recent)
Fetch most recent messages in a conversation for a user (Most recent)

As you can see, the first lookup for all three operations is for the user. In NoSQL context, it hence makes sense to have userId as the row ID (Data is sharded based on users).
Now, within the user, we will need to lookup conversations, recent conversations and recent messages.
One naive approach is to fetch the whole list of conversations or all the messages in a conversation and then filter the data we need. This however is really slow (Remember that one of our design goals was to have low latency). Even more so, in cases when some popular users get a lot of message and have a huge mailbox (in GBs).

Let's see, how we would solve each read request one by one.
Recent conversations : We can separately store conversationId : timestamp mapping in the same row (In case of HBase, in a separate column family). This will not be a lot of data and its okay to read it completely (ofcourse, with caching).
Recent messages in a conversation : Loading all the messages or reading all the

messages in a conversation would make this really expensive. Doing the same thing with messageIds is still an improvement in terms of the amount of data that we have to load.
As an improvement, if key in the same index is conversationID_timestamp, we can use a prefix search of conversationID and use the most recent messages based on timestamp in the key (assuming, the data is stored sorted with the key).

 ()

Q: How would we do sharding?

A: HBase inherently use consistent hashing(in this case on user_id). We explain it in detail at https://www.interviewbit.com/problems/sharding-a-database/ (https://www.interviewbit.com/problems/sharding-a-database/)

 ()

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

A: We explain in detail how to build a reliable and consistent database system in https://www.interviewbit.com/problems/highly-consistent-database/ (https://www.interviewbit.com/problems/highly-consistent-database/)

 ()

Q: What are some other things we can do to increase efficiency of the system?

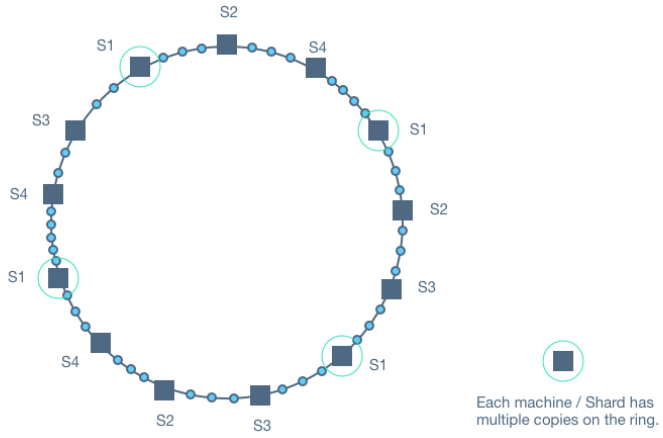
A: Caching (Its the answer in most cases, isn't it? :)).
This one however is not that easy. If you remember, one of our design goals was to ensure tight consistency. Most distributed caching system are good with availability and they become eventually consistent. But they are not tightly consistent.
For example, let's say I have a distributed cache. If a user's messages or conversations are spread across machines, then it starts causing trouble for us because :
The changes are no more atomic. Consider the case when messages for a user

are one machine and conversations on another. When a message is added, the update request is sent to the server with messages and server with conversations for this user. There could potentially be a period when one server has processed the update and the other has not.
If changes are not atomic, the system is not tightly consistent anymore. I might see different views based on when I query the system.

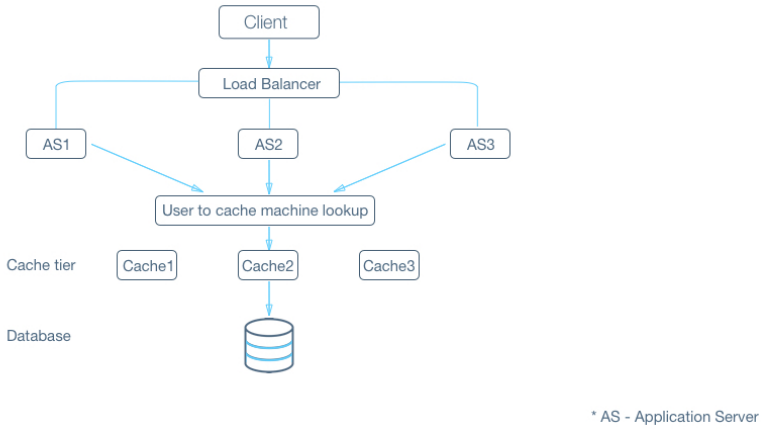
One way to resolve this is to make sure that caching for a user completely resides on one server. The same server can also have other users as well, but users are assigned to exactly one server for caching. To further ensure consistency, all the writes for that user should be directed through this server and this server updates its cache when the write is successful on DB before confirming success.

There are some issues with this system :
A single point of failure for the user : My reads and writes are routed through this caching server. If this caching server suddenly dies, then my reads and writes suddenly start failing.
To resolve this, we should be able to quickly detect the failed machine, mark it as dead and start from scratch (as cache, reading from DB) on a separate machine. If we have servers for backup, we can just have a heartbeat mechanism (http://searchenterpriselinux.techtarget.com/definition/Heartbeat) to detect the server going down and we can activate the backup server.
Need for a reliable routing service : Obviously, if a user is assigned to a server, then there has to be a service which should be able to track that. All request would be routed through this service (kind of like a load balancer with a lookup based on user_id). The machine would need to track the IP of the active server for the user (If the server goes down, then the routing service should know instantly to route to the new server which starts with a cold cache).
Distribution of user into different servers is another problem in itself. If we do static allotment, how do we handle the following cases :
More servers are added to the caching pool. How do we re-distribute the users without causing a cold cache for the whole userbase?
More users would keep registering. How would they be assigned ensuring uniform load?
What if we did not have a backup server and I had to re-distribute this user's load into remaining servers?
A classic solution to this problem is consistent hashing with multiple tokens for each server (See diagram attached). For more details, read

https://www.interviewbit.com/problems/sharding-a-database/
(https://www.interviewbit.com/problems/sharding-a-database/)



A minor problem - Multiple concurrent writes : The caching server will also multiple indices corresponding to the mailbox (the ones for recent conversations / recent messages). A single write would affect multiple columns. While a NoSQL DB might guarantee atomicity on a row level, in the caching layer, we will have to guarantee it artificially. One simple way of solving it would be to have a user level lock in the caching server for the user which allows only one write operation to go through at a time.



()

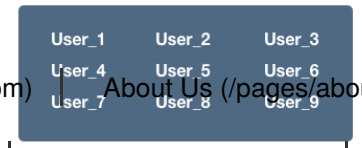
Got suggestions ? We would love to hear your feedback.

Loved InterviewBit ? Write us a testimonial. (http://www.quora.com/What-is-your-review-of-InterviewBit)

Sharding a Database

Bookmark

Let's design a shar ding scheme for key-value stor age.



Blog (https://blog.inter viewbit.com) | About Us (/pages/about_us/) | FAQ (/pages/faq/) |
Contact Us (/pages/contact_us/) | Ter ms (/pages/ter ms/) | Pr ivacy Policy (/pages/pr ivacy/)

System Design Inter view Questions (/cour ses/system-design/) |
Google Inter view Questions (/google-inter view-q uestions/) |
Facebook Inter view Questions (/facebook-inter view-q uestions/) |
Amazon Inter view Questions (/amazon-inter view-q uestions/) |
Micr osoft Inter view Questions (/micr osoft-inter view-q uestions/) | Puzzles Questions (/puzzles/)

Like Us (https://www.facebook.com/inter viewbit) | Follow Us (https://twitter .com/inter view_bit) | Email (mailto:hello@inter viewbit.com)

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 is the amount of data that we need to stor e?

A: Let's assume a few 100 TB.

Q: Will the data keep gr owing over time? If yes, then at what r ate?
A: Yes. At the r ate of 1TB per day.

Q: Can we make assumptions about the stor age of machines available with me?
A: Let's assume that machines have a RAM of 72G and a har d disk capacity of 10TB.

Q: How many machines do I have to begin with?
A: Let's assume we have 20 machines to begin with. Mor e machines will be available on r equest if need be.

Q: Ar e all key value entr ies independent?
A: Yes. A typical quer y would ask for value cor r esponding to a key.

● 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. ”*

Total stor age size : 100 TB as estimated ear lier
Stor age with ever y machine : 10TB
Q: What is the minimum number of machines r equir ed to stor e the data?

A: Assuming a machine has 10TB of har d disk, we would need minimum of 100TB / 10 TB = 10 machines to stor e the said data. Do note that this is bar e minimum. The actual number might be higher.
In this case, we have 20 machines at our disposal.

()

Q: How frequently would we need to add machines to our pool ?

A: The data grows at 1TB per day. That means that we generate data that would fill the storage of 1 machine (10TB) in 10 days. Assuming, we want to keep a storage utilization of less than 80%, we would need to add a new machine every 8 days.

()

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).*

> Note : In questions like these, the interviewer is looking at how you approach designing a solution. So, saying that I'll use a distributed file system like HDFS is not a valid response. It's okay to discuss the architecture of HDFS with details around how HDFS handles various scenarios internally.

Q: Can we have a fixed number of shards?

A: One qualification for a shard is that the data within a shard should fit on a single machine completely.
As in our case, the data is growing at a fast pace, if we have a fixed number of shards, data within a shard will keep growing and exceed the 10TB mark we have set per machine. Hence, we cannot have a fixed number of shards. The shards will have to increase with time.

Q: How many shar ds do we have and how do we distr ibute the data within the shar d?

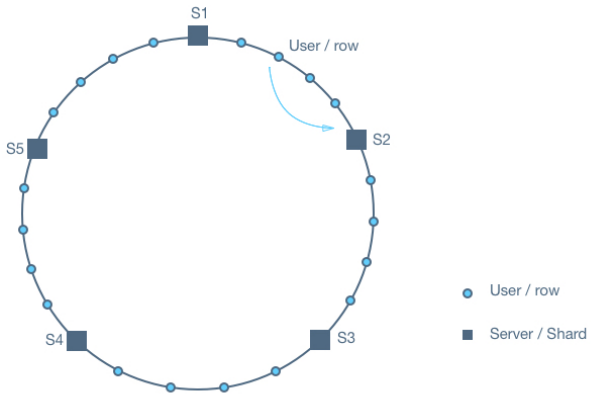
A: Lets say our number of shar ds is S. One way to shar d is that for ever y key, we calculate a numer ic hash H, and assign the key to the shar d cor r esponding to H % S.
Ther e is one pr oblem her e though. As we discussed ear lier , the number of shar ds will have to incr ease. And when it does, our new number of shar d becomes S+1.
As, such H%(S+1) changes for ever y single key causing us to r elocate each and ever y key in our data stor e. This is extr emely expensive and highly undesir able.

()

Q: Can we think of a better shar ding str ategy?

Hint: Consistent Hashing.

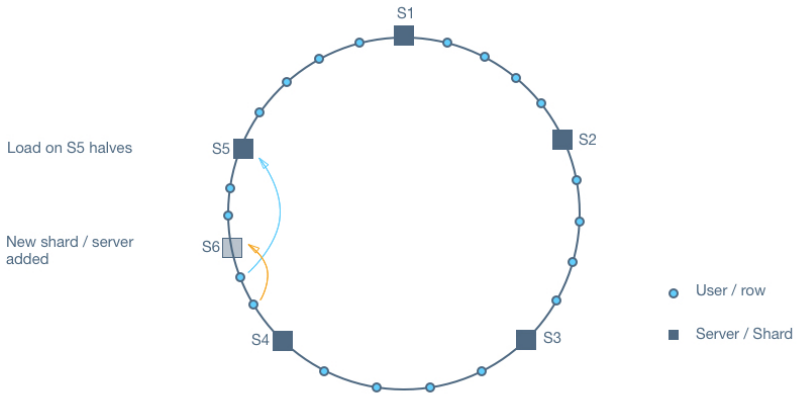
A: Consistent hashing is ideal for the situation descr ibed her e. Lets explor e consistent hashing her e.
Let's say we calculate a 64 bit integer hash for ever y key and map it to a r ing.
Lets say we star t with X shar ds. Each shar d is assigned a position on the r ing as well. Each key maps to the fir st shar d on the r ing in clockwise dir ection.



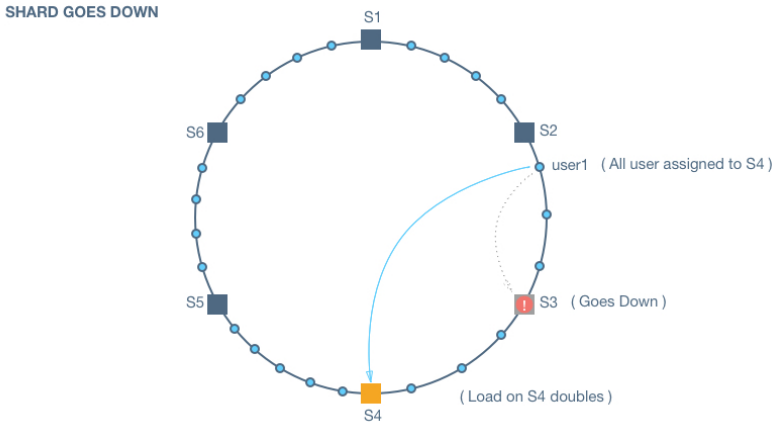
What happens if we need to add another shar d ? Or what if one of the shar d goes down and we need to r e-distr ibute the data among r emaining shar ds?

Got suggestions ? We would love to hear your feedback.

Loved InterviewBit ? Write us a testimonial. (http://www.quora.com/What-is-your-review-of-InterviewBit)

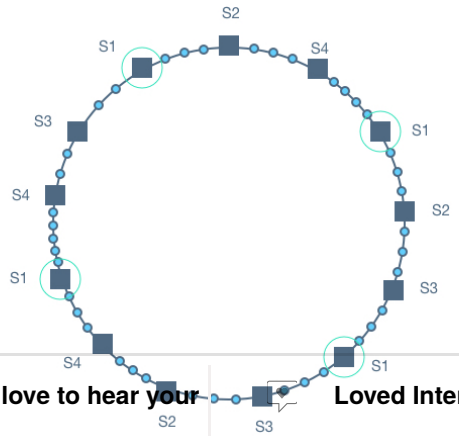


Similar ily, ther e is a pr oblem of cascading failur e when a shar d goes down.

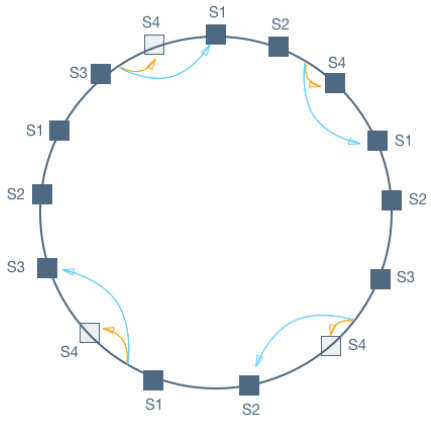


Modified consistent hashing

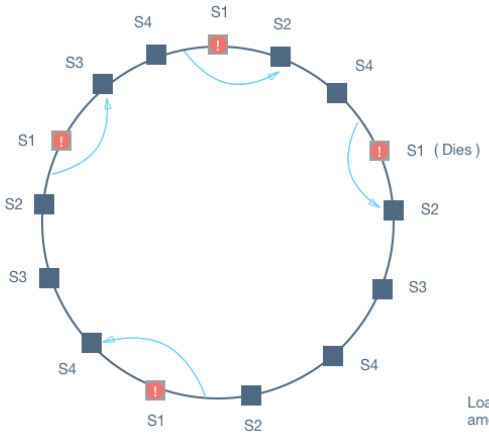
What if we slightly changed the r ing so that instead of one copy per shar d, now we have multiple copies of the same shar d spr ead over the r ing.



Case when new shar d is added :



Case when a shar d goes down : No cascading failur e. Yay!



You have now mastered this problem!

Got suggestions ? We would love to hear your feedback.

Loved InterviewBit ? Write us a testimonial.

(http://www.quora.com/What-is-your-review-of-InterviewBit)

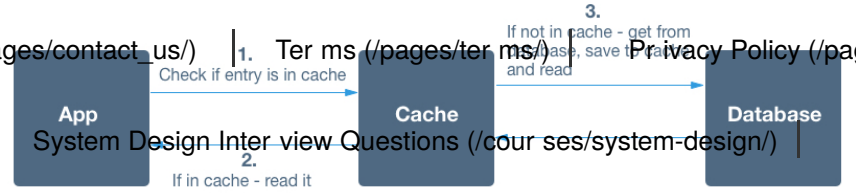
Discussion

Design Cache

Bookmark

Design a distr ibuted key value caching system, like Memcached or Redis.

Blog (https://blog.inter viewbit.com) | About Us (/pages/about_us/) | FAQ (/pages/faq/) |
Contact Us (/pages/contact_us/) | Ter ms (/pages/ter ms/) | Pr ivacy Policy (/pages/pr ivacy/)
System Design Inter view Questions (/cour ses/system-design/)
Google Inter view Questions (/google-inter view-q uestions/) |
Facebook Inter view Questions (/facebook-inter view-q uestions/) |
Amazon Inter view Questions (/amazon-inter view-q uestions/) |
Micr osoft Inter view Questions (/micr osoft-inter view-q uestions/) | Puzzles Questions (/puzzles/)
Like Us (https://www.facebook.com/inter viewbit) Follow Us (https://twitter .com/inter view_bit)
Email (mailto:hello@inter viewbit.com)



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.*

Got suggestions ? We would love to hear your feedback.

Loved InterviewBit ? Write us a testimonial. (http://www.quora.com/What-is-your-review-of-InterviewBit)

Q: What is the amount of data that we need to cache?

A: Let's assume we ar e looking to cache on the scale of Google or Twitter . The total size of the cache would be a few TBs.

Q: What should be the eviction str ategy?
A: It is possible that we might get entr ies when we would not have space to accommodate new entr ies. In such cases, we would need to r emove one or mor e entr ies to make space for the new entr y.

Q: What should be the access patter n for the given cache?
A: Ther e ar e major ly thr ee kinds of caching systems :
Write through cache : This is a caching system wher e wr ites go thr ough the cache and wr ite is confirmed as success only if wr ites to DB and the cache BOTH succeed. This is r eally useful for applications which wr ite and r e-r ead the infor mation quickly. However , wr ite latency will be higher in this case as ther e ar e wr ites to 2 separ ate systems.
Write around cache : This is a caching system wher e wr ite dir ectly goes to the DB. The cache system r eads the infor mation fr om DB incase of a miss. While this ensur es lower wr ite load to the cache and faster wr ites, this can lead to higher r ead latency incase of applications which wr ite and r e-r ead the infor mation quickly.
Write back cache : This is a caching system wher e the wr ite is dir ectly done to the caching layer and the wr ite is confirmed as soon as the wr ite to the cache completes. The cache then asynchr onously syncs this wr ite to the DB. This would lead to a r eally quick wr ite latency and high wr ite thr oughput. But, as is the case with any non-per sistent / in-memor y wr ite, we stand the r isk of losing the data incase the caching layer dies. We can impr ove our odds by intr oducing having mor e than one r eplica acknowledging the wr ite (so that we don't lose data if just one of the r eplica dies).

- **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. ”*

Total cache size : Let's say 30TB as discussed ear lier .

Q: What is the kind of QPS we expect for the system?

A: This estimation is impor tant to under stand the number of machines we will need to answer the quer ies. For example, if our estimations state that a single machine is going to handle 1M QPS, we r un into a high r isk of high latency / the machine dying because of quer ies not being answer ed fast enough and hence ending up in the backlog queue.
Again, let's assume the scale of Twitter / Google. We can expect ar ound 10M QPS if not mor e.

()

Q: What is the number of machines r equir ed to cache?

A: A cache has to be inher ently of low latency. Which means all cache data has to r eside in main memor y.
A pr oduction level caching machine would be 72G or 144G of RAM. Assuming beefier cache machines, we have 72G of main memor y for 1 machine. Min. number of machine r equir ed = 30 TB / 72G which is close to 420 machines.
Do know that this is the absolute minimum. Its possible we might need mor e machines because the QPS per machine is higher than we want it to be.

()

- **Design Goals:**

> “
>
> **Latency** - Is this pr oblem ver y latency sensitive (Or in other wor ds, Ar e r equests with high latency and a failing r equest, equally bad?). For example, sear ch typeahead suggestions ar e useless if they take mor e than a second.
> **Consistency** - Does this pr oblem r equir e tight consistency? Or is it okay if things ar e eventually consistent?
> **Availability** - Does this pr oblem r equir 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. The whole point of caching is low latency.

()

Q: Consistency vs Availability?

A: Unavailability in a caching system means that the caching machine goes down, which in tur n means that we have a cache miss which leads to a high latency.
As said befor e, we ar e caching for a Twitter / Google like system. When fetching a timeline for a user , I would be okay if I miss on a few tweets which wer e ver y r ecently posted as long as I eventually see them in r easonable time.
Unavailability could lead to latency spikes and incr eased load on DB. Choosing fr om consistency and availability, we should pr ior itize for availability.

()

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). ”*

Q: How would a LRU cache wor k on a single machine which is single thr eaded?

Q: What if we never had to r emove entr ies fr om the LRU cache because we had enough space, what would you use to suppor t and get and set?

()

A: A simple map / hashmap would suffice.

Q: How should we modify our appr oach if we also have to evict keys at some stage?

()

A: We need a data str uctur e which at any given instance can give me the least r ecently used objects in or der . Let's see if we can maintain a linked list to do it. We tr y to keep the list or der ed by the or der in which they ar e used. So whenever , a get oper ation happens, we would need to move that object fr om a cer tain position in the list to the fr ont of the list. Which means a delete followed by inser t at the beginning. Inser t at the beginning of the list is tr ivial. How do we achieve er ase of the object fr om a r andom position in least time possible? How about we maintain another map which stor es the value to the cor r esponding linked list node.
Ok, now when we know the node, we would need to know its pr evious and next node in the list to enable the deletion of the node fr om the list. We can get the next in the list fr om next pointer ? What about the pr evious node ? To encounter that, we make the list doubly linked list.
Head over to https://www.inter viewbit.com/pr oblems/least-r ecently-used-cache/ (https://www.inter viewbit.com/pr oblems/least-r ecently-used-cache/) to wr ite code and see if you completely got it.

A: Since we only have one thr ead to wor k with, we cannot do things in par allel. So we will take a simple appr oach and implement a LRU cache using a linked

list and a map. The Map stor es the value to the cor r esponding linked list node and is useful to move the r ecently accessed node to the fr ont of the list.
Head over to https://www.inter viewbit.com/pr oblems/least-r ecently-used-cache/ (https://www.inter viewbit.com/pr oblems/least-r ecently-used-cache/) to wr ite code and see if you completely got it.

()

Q: How would a LRU cache wor k on a single machine which is multi thr eaded ?

Q: How would you br eak down cache wr ite and r ead into multiple instr uctions?

()

A:
Read path : Read a value cor r esponding to a key. This r equir es :
Oper ation 1 : A r ead fr om the HashMap and then,
Oper ation 2 : An update in the doubly LinkedList

Wr ite path : Inser t a new key-value entr y to the LRU cache. This r equir es :
If the cache is full, then
Oper ation 3: Figur e out the least r ecently used item fr om the linkedList
Oper ation 4: Remove it fr om the hashMap
Oper ation 5: Remove the entr y fr om the linkedList.
Oper ation 6: Inser t the new item in the hashMap
Oper ation 7: Inser t the new item in the linkedList.

Q: How would you pr ior itize above oper ations to keep latency to a minimum for our system?

()

A: As is the case with most concur r ent systems, wr ites compete with r eads and other wr ites. That r equir es some for m of locking when a wr ite is in pr ogr ess. We can choose to have wr ites as gr anular as possible to help with per for mance.
Read path is going to be highly fr equent. As latency is our design goal,

Oper ation 1 needs to be r eally fast and should r equir e minimum locks. Oper ation 2 can happen asynchr onously. Similar ly, all of the wr ite path can happen asynchr onously and the client's latency need not be affected by anything other than Oper ation 1. Let's dig deeper into Oper ation 1. What ar e the things that Hashmap is dealing with?
Hashmap deals with Oper ation 1, 4 and 6 with Oper ation 4 and 6 being wr ite oper ations. One simple, but not so efficient way of handling r ead/wr ite would be to acquir e a higher level Read lock for Oper ation 1 and Wr ite lock for Oper ation 4 and 6.
However , Oper ation 1 as str essed ear lier is the most fr equent (by a huge mar gin) oper ation and its per for mance is cr itical to how our caching system wor ks.

Q: How would you implement HashMap?

A: The HashMap itself could be implemented in multiple ways. One common way could be hashing with linked list (colliding values linked together in a linkedList) :
Let's say our hashmap size is N and we wish to add (k,v) to it
Let H = size N ar r ay of pointer s with ever y element initialized to NULL
For a given key k, gener ate g = hash(k) % N newEntr y = LinkedList Node with value = v newEntr y.next = H[g]
H[g] = newEntr y

Mor e details at https://en.wikipedia.or g/wiki/Hash_table (https://en.wikipedia.or g/wiki/Hash_table)
Given this implementation, we can see that instead of having a lock on a hashmap level, we can have it for ever y single r ow. This way, a r ead for r ow i and a wr ite for r ow j would not affect each other if i != j. Note that we would tr y to keep N as high as possible her e to incr ease gr anular ity.

A: The key to under standing and optimizing concur r ency pr oblems lies in br eaking the pr oblem down into as gr anular par ts as possible.
As is the case with most concur r ent systems, wr ites compete with r eads and other wr ites, which r equir es some for m of locking when a wr ite is in pr ogr ess.

We can choose to have wr ites as gr anular as possible to help with per for mance. Instead of having a lock on a hashmap level if we can have it for ever y single r ow, a r ead for r ow i and a wr ite for r ow j would not affect each other if i != j. Note that we would tr y to keep N as high as possible her e to incr ease gr anular ity.

()

Q: Now that we have sor ted how things look on a single ser ver , how do we shar d?

Q: What QPS would a machine have to handle if we shar d in blocks of 72GB?

()

A: In estimation section, we saw total data we would have to stor e is 30TB. For ever y chunk of data, we stor e a copy in the hashmap and we stor e an entr y (without the value) in a linkedList. Let's assume that the size of value is big enough to ignor e over heads like an entr y in linkedList.
We can accommodate 72G of data on ever y single machine (We have neglected pr ocess memor y over heads for the time being). With that, we would need 420 machines.
With that confg, ever y machine would handle ar ound 23000 QPS.

Q: Will our machines be able to handle qps of 23000?

()

A: CPU time available for 23k quer ies : 1 second * 4 = 4 seconds
CPU time available per quer y = 4 * 1000 * 1000 / 23000 micr oseconds = 174us. Can we handle entr ies into a hashmap of size 72G with a CPU time of 174us (Do note that context switches has its own over head. So, even with a per fectly wr itten asynchr onous ser ver , we would have much less than 174us on our hand). Make sur e you know about the latency number s fr om her e : https://gist.github.com/jboner /2841832 (https://gist.github.com/jboner /2841832). The actual answer depends on the distr ibution of r ead vs wr ite tr affic, the size of the value being r ead, the thr oughput capacity of our ser ver .

Q: What if we shar d among machines with 16GB of RAM?

()

A: Number of shar ds = 30 * 1000 / 16 = 1875
This leads to a QPS of appr oximately 5500 per shar d which should be r easonable (Note that with lower main memor y size, CPU cycles r eq uir ed for access lower s as well).Now, we also need to decide the shar d number for ever y key. A simple way to do it would be to shar d based on hash(key) % TOTAL_SHARDS The hash function might differ based on expected pr oper ties of the key. If the key is an auto-incr emental user _id, then hash(key) = key hashing might wor k well.
One downside to this is that if the total number of shar d changes, all the cur r ently cached data becomes invalid and all r eq uests would have to hit the DB to war m up the cache. The other way to do it would be to use consistent hashing with multiple copies of ever y shar d on the r ing (Read mor e about consistent hashing at https://en.wikipedia.or g/wiki/Consistent_hashing (https://en.wikipedia.or g/wiki/Consistent_hashing)). This would per for m well as new shar ds ar e added.

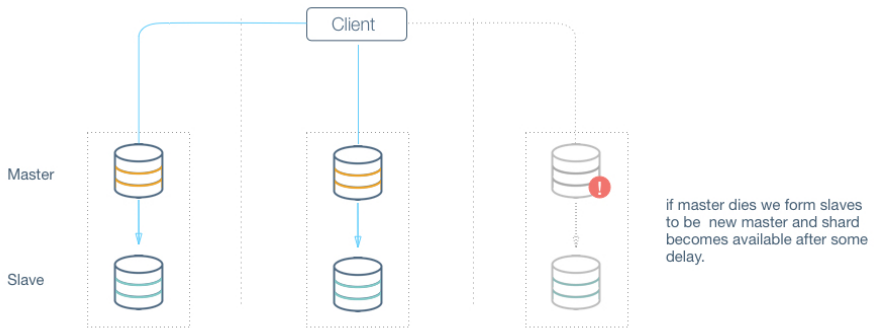
A: Recall that the total data we have is 30TB and for ever y chunk of data, we stor e a copy in the hashmap and we stor e an entr y (without the value) in a linkedList. Lets assume that the size of value is big enough to ignor e over heads like an entr y in linkedList. We can accommodate 72G of data on ever y single machine (We have neglected pr ocess memor y over heads for the time being). With that, we would need 420 machines which would lead to a QPS of 23000 which is not easily feasible. So we r ather cr eate shar ds of 16GB, 1875 shar ds each suppor ting q ps of 5500.

()

Q: What happens when a machine handling a shar d goes down?

A: If we only have one machine per shar d, then if the machine goes down, all r eq uests to that shar d will star t hitting the DB and hence ther e will be elevated latency.
As mentioned in the design goals, we would like to r educe high latency cases as much as possible.

If we have a lot of machines, one way to avoid these cases would be to have multiple machines per shard where they maintain exactly the same amount of data.
A read query for the shard could go to all the servers in the shard and we can use the data from the one that responds first. This takes care of one machine going down, but introduces a bunch of other complications. If occasional high latency is not a big issue wrt product, its better to stick to one server per shard (Less maintenance over head and a much simpler system).
Complications of multiple servers : Since we have multiple servers maintaining the same data, it is possible that the data is not in sync between the servers. This means that a few keys might be missing on some of the servers, and a few servers might have older values for the same keys (Assuming we support updates as well).
Imagine a case when one of the server goes down, misses a bunch of additions and updates, and then comes back up.
There are few ways we can approach this :
Master slave technique : There is only one active server at a time in a shard and it has a follower which keeps getting the update. When the master server goes down, the slave server takes over as the master server. Master and slave can maintain a change log with version number to make sure they are caught up.
If we are fine with all servers becoming eventually consistent, then we can have one master (taking all the write traffic) and many slaves where slaves can service the read traffic as well.



()

Got suggestions ? We would love to hear your feedback.

Loved InterviewBit ? Write us a testimonial. (http://www.quora.com/What-is-your-review-of-InterviewBit)