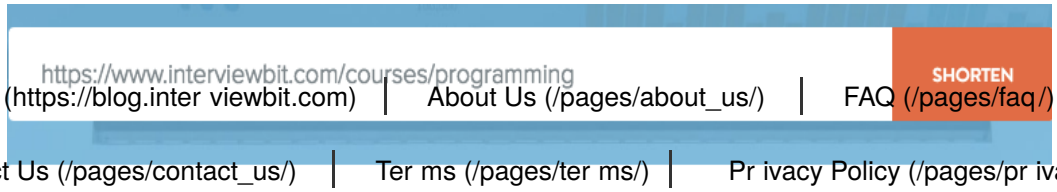
Design a URL shor tening ser vice, like bit.ly



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]}
```

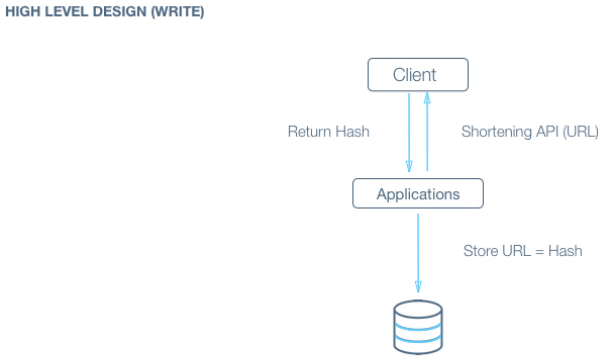
Both APIs ar e ver y lightweight, their computation will not the bottleneck plus we don't have to stor e any session infor mation about user s.
Basically, we ar e tr ying to build a ser vice which ser ves as a huge HashMap

()

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

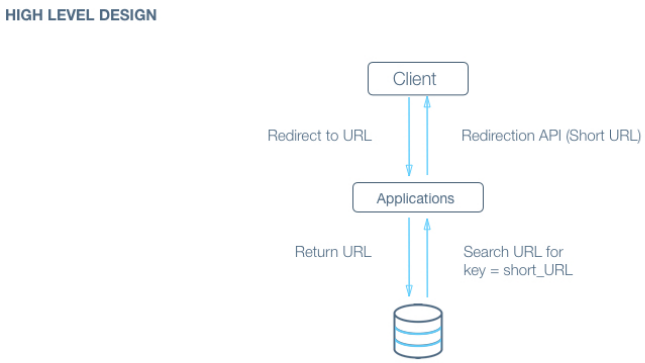
Database ser ver which stor es the hash => ur l mapping



()

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 estimonial. ()

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

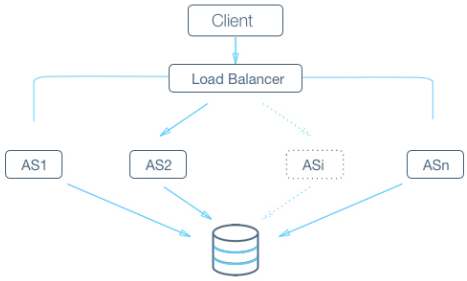
Q: How would you take car e of application layer fault toler ance?

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