**STAY FOCUSED**

**SHOW FIGHTING SPIRIT**

**Also Refer to**

[**https://www.interviewbit.com/courses/system-design/**](https://www.interviewbit.com/courses/system-design/)

[**http://blog.gainlo.co/index.php/2015/12/22/how-to-crack-the-coding-interview-with-practicing-only-30-questions/**](http://blog.gainlo.co/index.php/2015/12/22/how-to-crack-the-coding-interview-with-practicing-only-30-questions/)

[**https://www.youtube.com/watch?v=UzLMhqg3\_Wc**](https://www.youtube.com/watch?v=UzLMhqg3_Wc) **( gives the idea what to study for design questions)**

[**System Design Cheatsheet**](https://gist.github.com/vasanthk/485d1c25737e8e72759f) **( System design cheat sheet)**

[**https://github.com/donnemartin/system-design-primer**](https://github.com/donnemartin/system-design-primer) **(System design gist)**

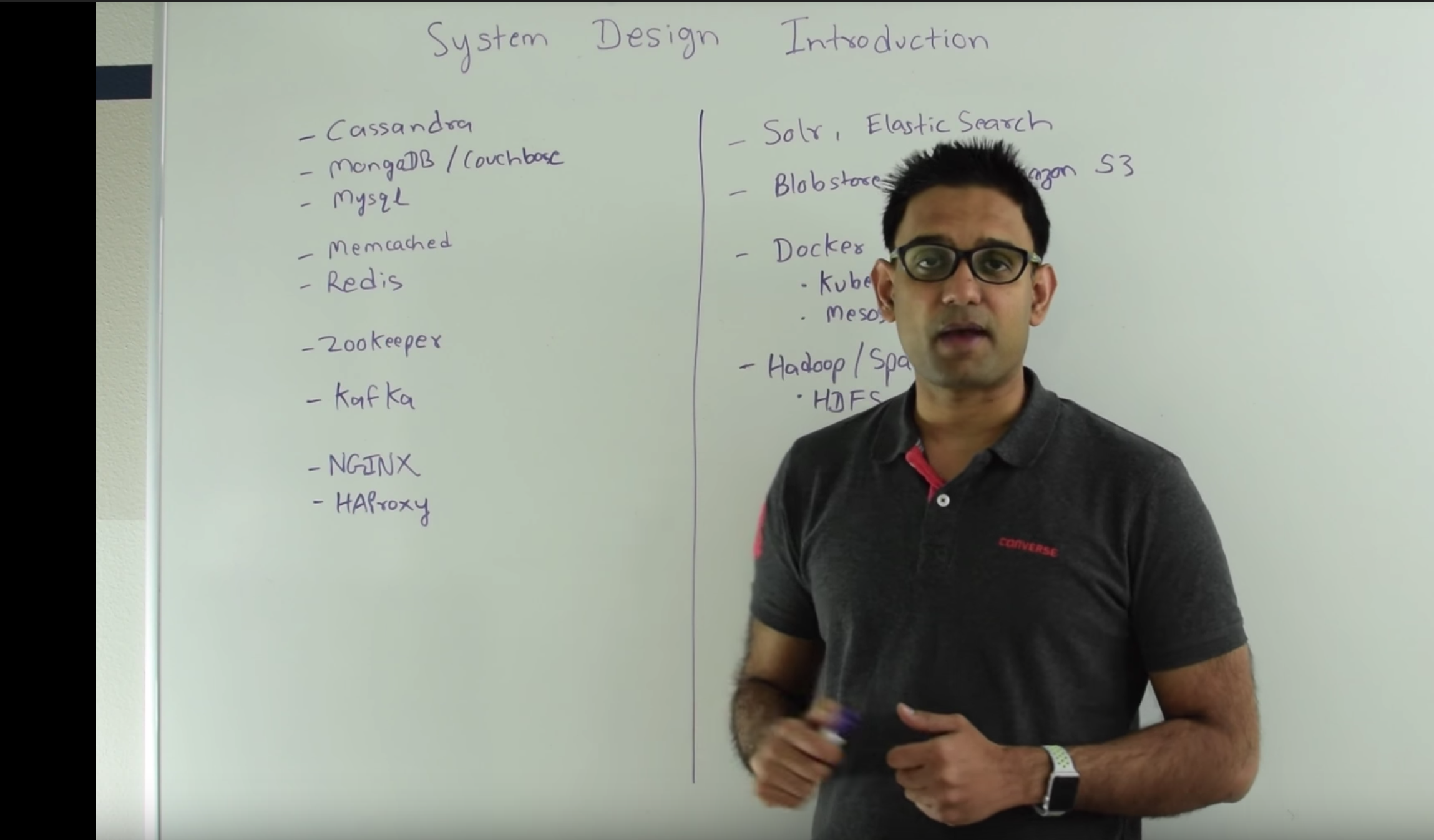
**To be checked YT channels : codekarle, thinksoftware**

During this process, discussion is the core. The candidate is more likely to l**ead the conversation and discussion high-level components, details, pros and cons, and everything with the interviewer.**During the interview session, your communication and problem-solving ability are mainly evaluated. Given an open-ended problem, how do you analyze the issue, how do you solve it step by step, how do you explain your idea and discuss with others, how do you evaluate your system and optimize it are what interviewers mostly care about.

**Thinking in terms of design problems :**

1. **Find out the use case scenarios, ask questions to gather as many requirements as possible.**
2. **Do the estimates, find out the numbers doing math to get an idea of the actual data,requests, network calls, storage etc.**
3. **Clarify doubts before attempting the solution**
4. **Attempt a high level design keeping scalability in mind.**
   1. **Think about load balancers**
   2. **Caching layer**
   3. **DB partitioning**
   4. **Normalized vs denormalized data for performance**

<https://www.youtube.com/watch?v=UzLMhqg3_Wc>

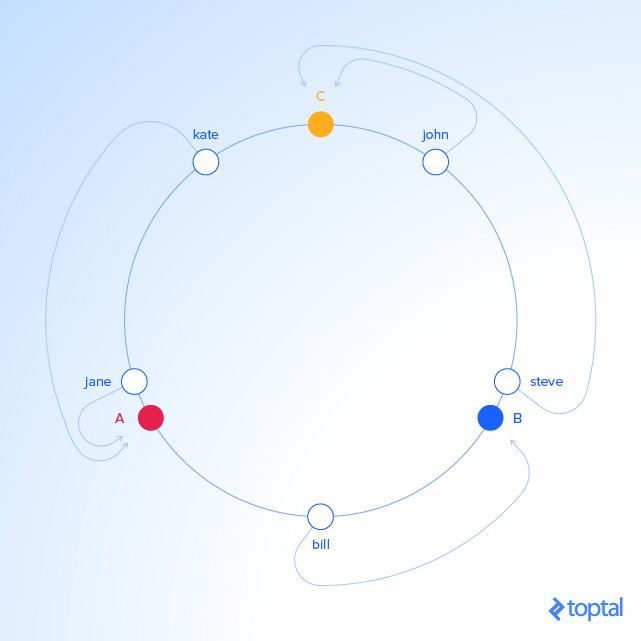
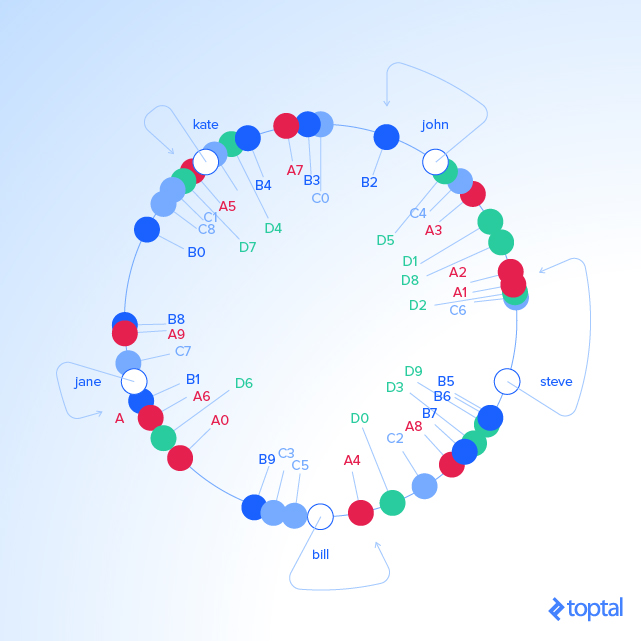
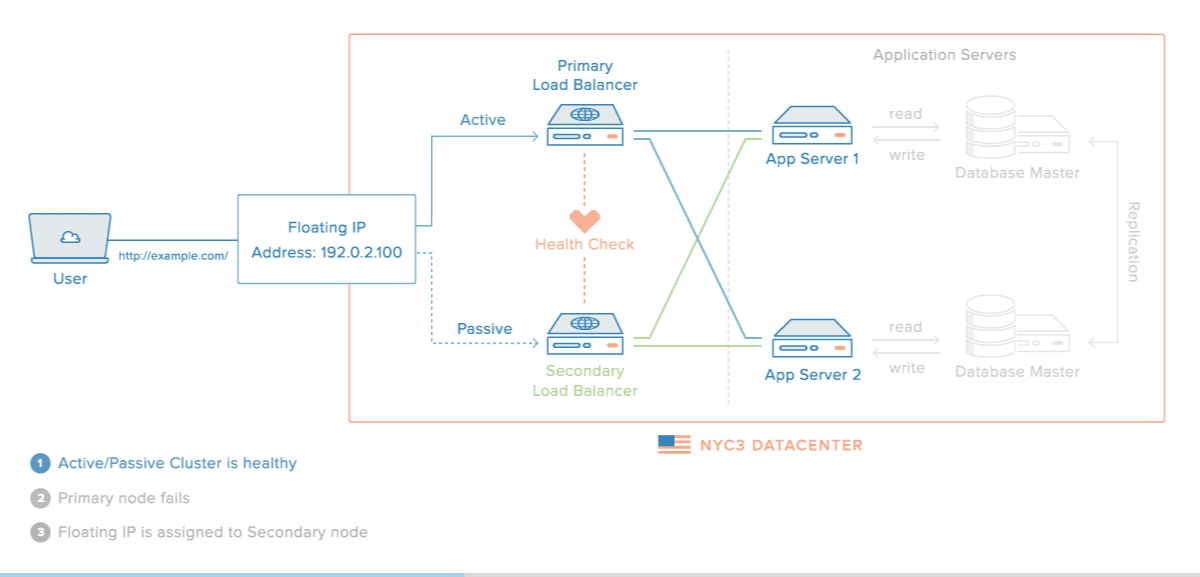


**IF I DON’T KNOW I’LL LEARN IT…!!!**

**(Gaps to fill : Data modeling, estimations, Database techs)**

**Pre-requisites :**

Systems are complex, and when you’re designing a system you’re grappling with its full complexity. Given this, there are many topics you should be familiar with, such as:

* **Concurrency.** Do you understand threads, deadlock, and starvation? Do you know how to parallelize algorithms? Do you understand consistency and coherence?
* **Networking.** Do you roughly understand IPC and TCP/IP? Do you know the difference between throughput and latency, and when each is the relevant factor?
  + Two way SSL : <http://www.robinhowlett.com/blog/2016/01/05/everything-you-ever-wanted-to-know-about-ssl-but-were-afraid-to-ask/>
  + Public key is shared to public(sender) and private key is kept with a person, public key is used to encrypt the message and then private key is used to decrypt it.
* **Abstraction.** You should understand the systems you’re building upon. Do you know roughly how an OS, file system, and database work? Do you know about the various levels of caching in a modern OS?
* **Real-World Performance.** You should be familiar with the speed of everything your computer can do, including the relative performance of RAM, disk, SSD and your network ( <http://highscalability.com/numbers-everyone-should-know> - Just the ball park numbers ).
  + Try to find the estimates for the defined solutions. Can be ballpark estimates initially but you should get the actual idea.
* **Estimation.** Estimation, especially in the form of a back-of-the-envelope calculation, is important because it helps you narrow down the list of possible solutions to only the ones that are feasible. Then you have only a few prototypes or micro-benchmarks to write.
* **Availability and Reliability.** Are you thinking about how things can fail, especially in a distributed environment? Do know how to design a system to cope with network failures? Do you understand durability ?
* Understand **CAP** theorem. The theorem states that of these three properties only 2 can be satisfied at a time:
  + <http://ksat.me/a-plain-english-introduction-to-cap-theorem>
  + Consistency: all nodes see the same data at the same time.
  + Availability: node failures do not prevent survivors from continuing to operate.
  + Partition tolerance: the system continues to operate despite message loss due to network and/or node failure
* **BASE** : **B**asically **A**vailable, **S**oft State, **E**ventual consistent (Used in NoSQL Databases)
* Know about **consistent hashing**.
  + For n number of servers a key o can be hashed : hash(o) modulo n.
  + But in case of addition or deletion of a server, n changes hence the keys need to be rehashed. To remedy this problem consistent hashing is used which distributes all the servers virtually on a circle and a range is decided for each server and then the keys are hashed falling in a particular range.
  + In case of deletion of a server, existing keys can be assigned to nearest server lying in the range
  + In case of addition of a server, keys can be rehashed based on the range.
  + 
  + 
  + The servers can be broken up in multiple instances and assigned the range, to avoid load on a particular server in case of addition/deletion or the keys falling in a particular range.
  + <https://www.toptal.com/big-data/consistent-hashing>
  + Consistent hashing real world example : <https://ably.com/blog/implementing-efficient-consistent-hashing>
* Know about **distributed** **caching**.
  + Distributed caching uses consistent hashing to allot the range of the keys for particular servers.
* Know about load balancing (HaProxy, nginx)
  + HAProxy and distributed load balancing notes :
  + ACL’s : access control lists are defined to route traffic based on rules/conditions
  + acl url\_blog path\_beg /blog (<http://yourdomain.com/blog/abcd>) will match this acl
  + Backend is defined to forward the request to specific server(s)
  + Backend web-backend  
     balance roundrobin  
     server web1 web1.yourdomain.com:80 check  
     server web2 web2.yourdomain.com:80 check  
      
    backend blog-backend  
     balance roundrobin  
     mode http //( routing layer)  
     server blog1 blog1.yourdomain.com:80 check  
     server blog1 blog1.yourdomain.com:80 check
  + 
  + Nginx architecture:
    - <https://www.nginx.com/blog/inside-nginx-how-we-designed-for-performance-scale/>
    - <http://www.aosabook.org/en/nginx.html>
* Know about Map-Reduce

**Cassandra notes:**

* Gossip protocol to identify status of a node within a data center, node publishes its own status & other nodes status to max 3 nearby nodes.
* Version is kept to know the most recent update and a log is updated in case node goes down and this log is used during bootstrap.
* <http://docs.datastax.com/en/cassandra/3.0/cassandra/architecture/archIntro.html>
* <https://blog.pythian.com/cassandra-use-cases/>
  + The ideal Cassandra application has the following characteristics:
  + Writes exceed reads by a large margin.
  + Data is rarely updated and when updates are made they are idempotent.
  + Read Access is by a known primary key.
  + Data can be partitioned via a key that allows the database to be spread evenly across multiple nodes.
  + There is no need for joins or aggregates.

**Redis**

* [**Overview Of Redis Architecture**](http://qnimate.com/overview-of-redis-architecture/)

Memcache

**MongoDB**

* [**https://www.mongodb.com/mongodb-architecture**](https://www.mongodb.com/mongodb-architecture)
* [**http://s3.amazonaws.com/info-mongodb-com/MongoDB\_Architecture\_Guide.pdf**](http://s3.amazonaws.com/info-mongodb-com/MongoDB_Architecture_Guide.pdf)

**DynamoDB**

* <https://www.youtube.com/watch?v=HaEPXoXVf2k>

**Kafka**

**RabbitMQ**

**SQS**

**Zookeeper**

Detailed organized notes

<https://tianpan.co/hacking-the-software-engineer-interview/?isAdmin=true#how-to-crack-the-coding-interview>

<https://astikanand.github.io/techblogs/high-level-system-design/>

CodeKarle video for system design :

<https://www.youtube.com/playlist?list=PLhgw50vUymycJPN6ZbGTpVKAJ0cL4OEH3>

**First Step :**

Cover everything here : <http://www.hiredintech.com/system-design/>

Watch the video at : <https://www.youtube.com/watch?v=-W9F__D3oY4> ( The video has few audience questions which is not properly audible, and it moves at a fairly slow pace. Maybe set the play speed to 1.25 / 1.5x ).

If you have time left over, go through : <http://book.mixu.net/distsys/ebook.html>

**Things to consider :**

* For read heavy systems try to cache or precomute, for example timeline of twitter can be precomputed and stored in Redis for fast loading.

**Basic checklist**

With every question you attempt, make sure you ask yourself the following questions where applicable :

* Have I understood the requirement correctly ? Do I completely understand the expectations in terms of performance, features and reliability? What are critical requirements for the system :
  + Does it need to be high throughput ?
  + Does it need to be highly available ?
  + Does it have high write traffic and hence concurrency issues, or is it read heavy.

Check

<https://player.vimeo.com/video/86413525> and

<https://player.vimeo.com/video/86413528>

<https://player.vimeo.com/video/86413531>

for example.

* Is my design fault tolerant ? What happens when a machine goes down ? ( application / database / load balancer machine ).
* What can we compromise on from CAP ( Consistency / Availability / Partitioning ) ?
* For write heavy questions, how do you handle concurrent reads / writes ? Can writes be batched / sampled ?
* Have you taken care of data not fitting on a single machine ? How do you shard data across machines ?
* Once data is sharded, how do you correctly route the requests ?
* Is the current system latency sensitive ? Search typeahead for example is extremely latency sensitive. If yes, then how do you guarantee extremely low latencies ?
* Relational DB / NoSQL ? Why ?

We will cover around **20 case studies** here.

**Make sure you attempt these cases yourself before looking into the final solution.**

**CASE STUDY 1 :**

*Design a URL shortening service.*

Already covered at the hiredintech link shared.

Check

<https://player.vimeo.com/video/86413525> ,

<https://player.vimeo.com/video/86413528> and

<https://player.vimeo.com/video/86413593> incase you missed it.

Application layer design : then scaling - Using hash modulo function

Database layer design : then scaling

Horizontal(adding more machines, partitioning) vs Vertical scaling (Adding more RAM,CPU)

More detailed answer(multiple approaches):

[How to design a tiny URL or URL shortener?](http://massivetechinterview.blogspot.in/2015/06/how-to-design-tiny-url-or-url-shortener.html)

**CASE STUDY 2:**

Design an elevator system

[Modelling an elevator using Object-Oriented Analysis and Design - Stack Overflow](https://stackoverflow.com/questions/493276/modelling-an-elevator-using-object-oriented-analysis-and-design)

<https://www.quora.com/What-is-the-best-answer-to-Design-an-elevator-system>

*How would you optimize an elevator system for a building with 50 floors and 4 elevators ? Optimize in terms of lowest wait times for the users.*

[*http://www.quora.com/What-are-ways-to-optimize-the-service-algorithm-for-an-elevator*](http://www.quora.com/What-are-ways-to-optimize-the-service-algorithm-for-an-elevator)

[*http://dan-nolan.com/how-i-would-optimize-the-elevators-in-our-office-building/*](http://dan-nolan.com/how-i-would-optimize-the-elevators-in-our-office-building/)

*Multiple approaches :*

[*http://massivetechinterview.blogspot.in/2015/07/thought-works-object-oriented-design.html*](http://massivetechinterview.blogspot.in/2015/07/thought-works-object-oriented-design.html)

**CASE STUDY 3 : Can be done again**

*A random ID generator which generates unique IDs. Think about such a generator for a company like Google.*

[*http://www.slideshare.net/davegardnerisme/unique-id-generation-in-distributed-systems*](http://www.slideshare.net/davegardnerisme/unique-id-generation-in-distributed-systems)

**CASE STUDY 4 :**

*Design a simplified version of Twitter where people can post tweets, follow other people and favorite\* tweets*.

<http://www.hiredintech.com/data/uploads/hiredintech_system_design_the_twitter_problem_beta.pdf>

<https://www.interviewbit.com/problems/design-twitter/>

<https://www.youtube.com/watch?v=KmAyPUv9gOY>

1. How db indexes data ? B+ tree structure, indexes are stored as pointers and takes space, a pointer to actual block is stored in B+ tree node.
   1. It helps in binary search, since indexes are done on unique-values(advised) and hence sorted. Makes searching efficient
   2. Index column in order of uniqueness and the column which comes in where condition first should be indexed first.
   3. <https://stackoverflow.com/questions/1108/how-does-database-indexing-work>
   4. <http://opensourceforgeeks.blogspot.in/2017/05/how-does-database-indexing-work.html>
2. Design restful service for this solution - done

**CASE STUDY 5 :**

*Design Facebook Timeline*

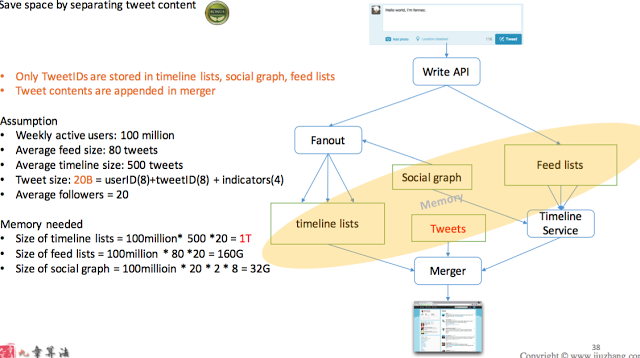
<https://www.facebook.com/note.php?note_id=10150468255628920>

[**https://www.quora.com/What-are-the-best-practices-for-building-something-like-a-news-feed**](https://www.quora.com/What-are-the-best-practices-for-building-something-like-a-news-feed)

[**http://blog.gainlo.co/index.php/2016/03/29/design-news-feed-system-part-1-system-design-interview-questions/**](http://blog.gainlo.co/index.php/2016/03/29/design-news-feed-system-part-1-system-design-interview-questions/)

[**http://massivetechinterview.blogspot.in/2014/12/design-facebook-timeline-function.html**](http://massivetechinterview.blogspot.in/2014/12/design-facebook-timeline-function.html)

* Push vs pull model, push activity stream in case less number of friends/followers
* Pull in case large number of followers, there should be a backup algorithm in case pull fails to read data. ( celebrity scenario: large number of followers)



**CASE STUDY 6 :**

*Design a trending topic system*

<http://www.michael-noll.com/blog/2013/01/18/implementing-real-time-trending-topics-in-storm/>

A small gist at <https://www.quora.com/How-does-Twitter-select-trending-topics>

**CASE STUDY 7 : WIP**

*Design notification system*

<http://stackoverflow.com/questions/9735578/building-a-notification-system>

**CASE STUDY 7 :**

*Design a search typeahead.*

Clarifying questions :

* How many typeahead suggestions are to be provided ?
* Do we need to account for spelling mistakes ? Example : Should typing “mik” give michael as a suggestion because michael is really popular as a query ?
* What can be the length of a search query ?

Lets assume for this question, we focus on only providing 5 suggestions at max. We need not account for spelling mistakes, and assume that the suggestions will have the typed phrase as the strict prefix.

So, in effect we have a system which does 2 major things :

* Given a query, it gives back upto 5 typeahead suggestions.
* Given a search query ( which is actually search for by the user ), it updates the suggestions if needed.

One approach would be to store the data as a trie. But, do you construct a complete trie ? Is there a limit to the number of characters in a query ? Users can type any random string as a query and that can cause the trie size to blow up.

Alright, so we only construct the nodes that are needed.

How do we calculate the top 5 suggestions then ? Top 5 frequency query terms with the user typed query as prefix seems to be a good approach. How do we find top 5 frequency query results for a query ?

Do we go and traverse the whole subtree in the trie to find the top frequency terms ? If so, what can be the size of the subtree ? Do you think its going to grow too inefficient ( especially because typeahead is latency sensitive ) ?

Can we store some additional information on the node itself ? How about we store the top 5 terms along with the frequency on the top 5 terms ? Query becomes really fast then. Update in the trie would mean percolating up the new term with its frequency, and see if its eligible to be in top 5 at every node.

What about the updates ? Do you update the trie with every update ? Would that cause things to be really slow ? Would sampling work here ?

Now, what do you think about the trie size ? Do you think it fits on a single machine ?

There are 2 options here :

1. You shard the trie. How do you shard the trie ? Do you only shard it on the first level ?
2. Maintain the trie as a refined set of queries which are more frequent than a certain threshold. All query terms along with the actual frequencies are stored in another hashmap. How do you do the update ? Batch update ? Would you compromise on real time updates for recent trending search terms ? What if you trigger the entry / update on search terms when it crosses certain threshold post sampling ?

What about fault tolerance ? Replication ?

How about optimizations on the client side ? Do you trigger off a request to the backend on every keystroke ? Or do you wait for 100ms and trigger off request if there have been no other keystroke ?

**CASE STUDY 8 :**

*Design an online multiplayer game*

* [How to Create an Asynchronous Multiplayer Game](http://www.indieflashblog.com/how-to-create-an-asynchronous-multiplayer-game.html)
* [How to Create an Asynchronous Multiplayer Game Part 2: Saving the Game State to Online Database](http://www.indieflashblog.com/how-to-create-async-part2.html)
* [How to Create an Asynchronous Multiplayer Game Part 3: Loading Games from the Database](http://www.indieflashblog.com/how-to-create-async-part3.html)
* [How to Create an Asynchronous Multiplayer Game Part 4: Matchmaking](http://www.indieflashblog.com/how-to-create-async-part4-html.html#comment-4447)
* [Real Time Multiplayer in HTML5](http://buildnewgames.com/real-time-multiplayer/)

**CASE STUDY 8 :**

*Design a Facebook like status system*

<http://stackoverflow.com/questions/7072924/what-is-the-design-architecture-behind-facebooks-status-update-mechanism>

**CASE STUDY 9 :**

*Design Facebook messages*

<https://www.facebook.com/notes/facebook-engineering/inside-facebook-messages-application-server/10150162742108920>

**CASE STUDY 10 :**

*Design Facebook places with check-in. Focus on implementing places suggestions as well*

*Hint : K-D tree*

**CASE STUDY 11 :**

*How would you design the feature in LinkedIn where it computes how many hops there are between you and another person? ( Or degree of connection at Facebook )*

*Hint: Meet in the middle*

**CASE STUDY 12 :**

*Design a Google document system*

[*https://neil.fraser.name/writing/sync/*](https://neil.fraser.name/writing/sync/)

**CASE STUDY 13 :**

*Design a key value store ( distributed )*

<http://www.slideshare.net/dvirsky/introduction-to-redis>

**CASE STUDY 14 :**

*Design gmail backend ( think about how requirements should be different from Facebook messages )*

[*http://blog.sajithmr.me/gmail-architecture/*](http://blog.sajithmr.me/gmail-architecture/)

**CASE STUDY 17 :**

*Design a caching system ( Think about Write back vs write through and cache invalidation)*

[*https://msdn.microsoft.com/en-us/library/dd129907.aspx*](https://msdn.microsoft.com/en-us/library/dd129907.aspx)

**CASE STUDY 18 :**

*Design a scalable web crawling system*

[*http://cis.poly.edu/tr/tr-cis-2001-03.pdf*](http://cis.poly.edu/tr/tr-cis-2001-03.pdf)

**CASE STUDY 19 :**

*Design a recommendation system ( this is fairly wide scope. Points for narrowing down the requirements )*

[*http://ijcai13.org/files/tutorial\_slides/td3.pdf*](http://ijcai13.org/files/tutorial_slides/td3.pdf)

[*http://tech.hulu.com/blog/2011/09/19/recommendation-system/*](http://tech.hulu.com/blog/2011/09/19/recommendation-system/)

**CASE STUDY 20 :**

*Design image sharing website ( Flickr / Instagram )*

[*http://highscalability.com/blog/2011/12/6/instagram-architecture-14-million-users-terabytes-of-photos.html*](http://highscalability.com/blog/2011/12/6/instagram-architecture-14-million-users-terabytes-of-photos.html)

**Interesting and Useful reads :**

* <http://highscalability.com/blog/2013/4/15/scaling-pinterest-from-0-to-10s-of-billions-of-page-views-a.html>
* <http://highscalability.com/youtube-architecture>

**CASE STUDY 21 :**

Rate limiting design problem:

<https://www.youtube.com/watch?v=FU4WlwfS3G0>

More questions for system design :

<https://hackernoon.com/top-10-system-design-interview-questions-for-software-engineers-8561290f0444>

<https://github.com/checkcheckzz/system-design-interview>

<https://www.educative.io/collection/5668639101419520/5649050225344512>