Date : 06/05/2021

High Level System Design :

(AlgoExpert i.e. SystemExpert)

content : 22 Fundamental Lectures (Algoexpert)

Aman Barnwal

Resources: ① Grokking the system Design (Educative.io) pdf

② System Design Videos (clement)

③ Gaurav Sen

④ Tech Dummies

⑤ codeKode (Yogita's channel) → System Design Primer Course

⑥ GitHub link → donnemartin (System Design Primer)
checkcheck33 (System Design Interviews)
shashank88

Other Top Youtube channels for HLD:

① Codekarle
② Think software
③ Udit Agarwal
④ Success in Tech
⑤ Jackson Gabbard
⑥ Coding simplified
⑦ System Design Interview

Blogs and websites:

highscalability.com
Interviewbit → system Design
Gaurav Sen Answer (Quora)

Syllabus

Main Topics of System Design:

Cracking System Design (Educative.io)

- System Design Basics
- Key characteristics of Distributed systems
- Load Balancing
- Caching
- Data Partitioning
- Indexes
- Proxies
- Redundancy and Replication
- SQL vs NoSQL
- CAP Theorem
- Consistent Hashing
- Long Polling vs Websockets vs Server-Sent Events

SystemExpert (Clement)

- Client-server model
- Network Protocols
- Storage
- Latency and Throughput
- Availability
- Hashing
- Relational Databases
- Key-value stores
- Specialized storage Paradigms
- Replication & sharding
- Leader Election
- Peer to peer networks
- Polling & streaming
- Configuration
- Rate Limiting
- Logging and Monitoring
- Publish/Subscribe Pattern
- MapReduce
- Security and Https
- API Design

content of this copy

System Design Problems

educative.io

1. Step by step guide
2. Designing a URL shortening service like TinyURL
3. Designing Pastebin
4. " Instagram
5. " Dropbox
6. " Facebook Messenger
7. " Twitter
8. " Youtube or Netflix
9. " Typeahead suggestion
10. " an API Rate Limiter
11. " Twitter search
12. " a web crawler
13. " Facebook's Newsfeed
14. " Yelp or Nearby Friends
15. " Uber Backend
16. " Ticketmaster
17. Design Algoexpert

systemexpert (clement)

18. Design a Stockbroker
19. " Google Drive
20. " Reddit API
21. " Netflix
22. " Uber API
23. " Slack
24. " Airbnb
25. " a code deployment system

Latency and Throughput:

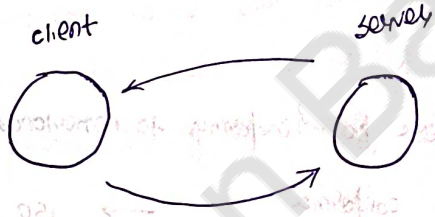
Latency and Throughput are 2 most important measures of performance of the system.

Latency:

Latency is basically how long it takes for data to get from one point in the system to another point in the system.

Eg: Latency of a network request
(How long it takes for one request to go from a client to a server and then back from the server to a client)

Let's say, you are reading the data from either disk or memory then the time it takes to read data is also referred to as latency.



There is a tradeoffs in the ways systems were built; bcz certain things will have higher latency but other things in the system will have lower latency.

Note:

① If you are reading 1 MB sequentially from memory, time it takes is 250 us.
(i.e. It is a very fast operation).

② Reading 1 MB sequentially from SSD (solid state drive) → 1000 us

i.e. Reading data from memory is a lot more faster than reading data from solid state drive (SSD).

③ Sending 1 MB data over 1 Gbps network ⟶ 10,000 us

(1 Gig Gigabit per sec)

(We are assuming that we are sending 1 MB to the machine that is right next to us ie. we are not accounting any distance here)

Eg: Making a network request
" " API call

④ Reading 1 MB from HDD ⟶ 20,000 us
(Hard drive)

Depending on the hardware that you have, (ie. HDD vs SDD) sending something over the network might actually be faster.

⑤ Sending a packet over the network from California to Netherlands and then back to California ⟶ 150,000 us

(Let's say, packet here is much smaller than 1 MB)

Eg: 1000 to 1500 bytes typically

↓

The time taken a lot more time than the remaining.

→ when you are designing the system, you are typically gonna optimize those systems by lowering the overall latency of the system.

Some systems will not necessarily be worried optimising for latency as much as others.

Eg: certain types of system might really really care about low latency

(Eg: A High Trading Firm
Video Game)

i.e. If you find some lag in a vidrogame, that just means you as a player in that video game is having high latency. That might be bcz the server in which you are playing on might be located half way across the world from you and it just takes a while for a computer or client to make a network request to a video game server.

Even though 150,000 us is fast, but if you are sending a lot more data over the network, than this 150,000 us quickly add up and it will have a really bad user experience bcz of high latency.

→ on the other hand, certain websites might care less about latency. But what they care about more is accuracy or uptime of the website.

→

Throughput:

Throughput is how much work a machine could perform in a given period of time.

(we are usually talking about how much data can be transferred to one point in the system from another point in the system in a given amount of time).

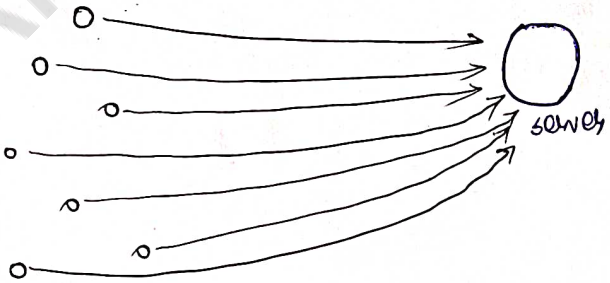
→ Typically, we measure throughput in Gigabits per sec ie. Gbps
or Kilobits per sec ie. Kbps
or Megabits per sec ie. Mbps.

Eg: sending 1 MB data over 1 Gbps network takes → 10,000 us

↓

This just means this network can support 1 Gigabit per sec
or
a billion bit per sec

Eg: Let's say, we have a bunch of clients and they all are issuing request from the same server.



clients

Here, throughput is basically how many these requests can this server handle in a given amount of time.

or

how many bits can this server handle per sec?

How will I increase the throughput ?

or

How will I optimize the system for throughput ?

→ You just pay to ↑ the throughput.

Eg: The things that determine how many bits can go in and out of Algoexpert server at any given time is determined by a cloud provider.
(Eg: Google cloud platform, AWS, etc)

ie. we will just pay these cloud providers to increase the throughput

Note: Above is just a naive soln.

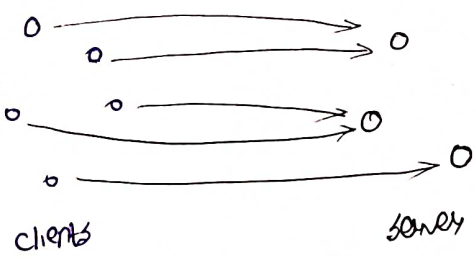
Let's say, a single server is handling multiple clients having a ton of requests. It might be Google search or Facebook messanger that is expected to serve thousands of requests or perhaps millions of request per sec.

so, just blindly increase the throughput on this network probably won't make sense.

↓

so, a better way to fix this is to have multiple servers, handling these requests.

Eg:



Note:

Even though latency and Throughput are very much related, they are not necessarily co-related.

Eg: You might have small parts of a system that has very low latency ie. they support really fast data transfers.

while you might have other part of the system which can have really low throughput.

ie. ~~low-lat~~

You can't make assumptions on the latency and throughputs based on one another.

Availability:

You can think of availability of a system as

→ How resistant the system is to failure?

How fault tolerance is the system?

(Referred as System's Fault Tolerance)

→ % of time in a given period of time like a month or year are at least operational enough such that all of its primary functions are satisfied.

→ Availability is a very important aspect to think about when you are evaluating a system.

→ Most systems almost have employed guarantee of availability.

Eg: When you ~~are~~ purchase Algoexpert, the main thing you purchase is access to our content and courses.

There is a employed guarantee of availability when you purchase Algoexpert bcz you expect the website of AlgoExpert.io to be up and fully operational.

↓

So, availability matters a lot for any company.

→ There are varying degrees of availability that you can expect from different systems.

Eg: Let's say, we are dealing with a software that supports Airplane software.

↓

The software that allows Airplane to function properly when its in flight

If that software goes down, that would be fully unacceptable bcz it is directly attached to lives of people flying in that aeroplane.

So, in this system, you would expect extremely high amount of availability.

Eg: 2nd example of this could be Youtube.

If Youtube ever goes down, then that's really bad. Bcz 100's of millions of users use Youtube everyday

Eg: Or take cloud providers for instance.

Let's say, if parts of Gcp or AWS, then that will affect all the business and the customers that rely on their services for their own platforms.

How do you measure availability ?

→ We measure typically availability as the percentage of system's uptime in a given year.

Eg: If the system is up and operational for half of an entire year, then you would say that system has 50% availability.

↓

which is really really bad for most services

You could imagine if Facebook, uber, etc is down for half of the time.

VVI

→ Most systems or services often aim for really really high availability. So, we often end up measuring availability not exactly in percentages but rather in Nines.

Nines are effectively percentages but they are specifically percentages with the number 9.

Eg: If you have systems having 99% of availability, meaning that it is up 99% of the time during a year then we say that your system has 2 nines of availability (bcz the number 9 comes twice in this percentage).

Eg: (ii) If your system has 99.9 % of availability then we say that your system has 3 nines of availability.

(iii) " " has 99.99 % " - - - - - - - - - - - - - -

- - 4 nines of availability

Note:

% of Availability	Nines of availability	Downtime per year
Similarly, 90 %	→ 1 nine	→ 36.53 days
99 %	→ 2 nines	→ 3.65 days
99.9 %	→ 3 nines	→ 8.77 hours
99.99 %	→ 4 nines	→ 52.6 minutes
99.999 %	→ 5 nines	→ 5.26 min
99.9999 %	→ 6 nines	→ 31.56 sec
99.99999 %	→ 7 nines	→ 3.16 sec
99.999999 %	→ 8 nines	→ 315.58 ms
99.9999999 %	→ 9 nines	→ 31.56 ms

→ calling a system highly available is an actual recognized term in system Design, ~~else~~ and it's often abbreviated as "HA"

↓
means for Highly Available

So, a lot of system aims to be highly available system bc3 they really care for availability.

→ Availability matters a lot – both for system designers and end user of the system. Infact, it matters so much that for certain systems, you don't have an employed guarantee of availability, you have an explicit guarantee of availability.

SLA :

→ Many service providers out there have an SLA .

↓

Service Level Agreement

SLA is an agreement b/w service providers and and user of this service on that system's availability on that other things.

So, many service providers have explicit written SLA that basically tells customers that "Hey, we guarantee this amount of availability or this % of uptime of system

SLO : It stands for "service Level Objective."

You can think of SLO as the components of SLA.

SLA is made up of SLO.

Eg:

→ If I provide service for you and if I guarantee some percentage of uptime for that service, that % of uptime guarantee would be SLO.

Or

If I guarantee you that you'll get only x no. of errors when you use my service, then that is another SLO.

Note :

All cloud providers like GCP or AWS have the very clear cut SLA's.

Infact, you could see them on their product pages

↓

These are very important when you are considering buying their services

SLA is very serious and availability is taken very seriously by any firm.

→ Availability is not always super super important ie. it's not always the case that you must have 5 nines of availability for all systems.

Bcz having high availability comes with trade offs, such as higher latency and low throughput.
It's difficult to achieve high availability

So, you as a system designer have to think long and hard about whether or not your system should be highly available.

what part of system require higher availability and what part of the system would be okay to have lower availability.

How do you actually improve the availability of the system?

→

You want to make sure first and foremost that your system does not have a single point of failure.

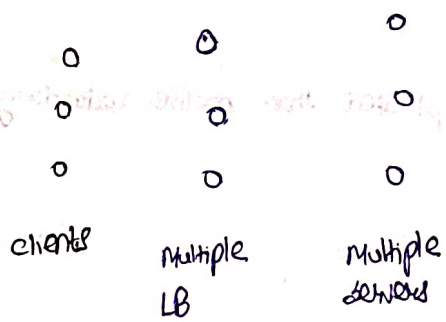
& How do you eliminate single point of failure?

↓
with Redundancy

Redundancy:

It is basically the act of duplicating or triplicating or multiplying even more certain parts of the system.

Eg: Adding servers using multiple load balancers to avoid SPOF.



SL

→ M → You can introduce redundancy in a lot of different parts of your system just by literally adding machines to those parts of the system.

(i) Passive Redundancy, when you have multiple components at a given level layer in your system. (using multiple servers and multiple load balancers, etc).

→ If at any point, one of these components will be down, nothing really will happen bcz other components can handle the requests smoothly.

Eg of a Passive redundancy →

An airplane engine

Eg →

If one engine fails, the airplane can very smoothly function using other engine.

(ii) Active Redundancy:

It is when you have multiple machines that work together in such a way that only one or few of the machines are typically handling the traffic.

→ If the machine which is handling traffic fails, the other machine are gonna somehow know that the other one failed. and they are gonna take over.

→ Active redundancy is a little more complicated than passive redundancy.

Caching :

We use caching in algorithms to avoid redoing the same operation especially computationally complex operations, which might take a lot of time.

↓

So, we use caching to ↑ the time complexity of to speed up our algorithms.

→ In system design, caching is used to speedup the system. It is to reduce or improve the latency of the system.

You can use caching in a bunch of different places in a system.

Eg: You can cache at the client level.

ie. caching can be somewhere at the client level such that it no longer has to go to the server to retrieve it.

similarly, you could cache at the server level. May be you always need server to interact with the client but may be the server do not always need ~~datab~~ to go the database to retrieve data.

→ You can also have cache in between two components of the system.

Maybe, you could have a cache bwn a server and a database.

→ infact, you can ~~bee~~ even have caching at the hardware level. in modern day computers.

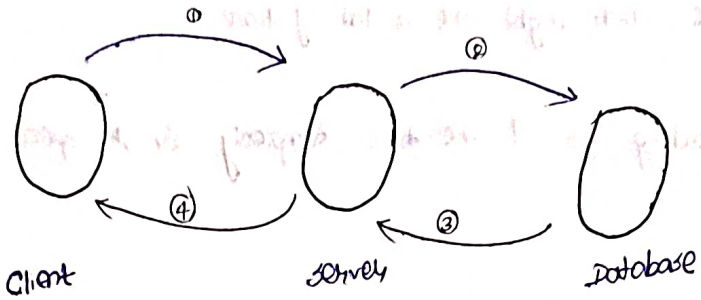
Eg: There are CPU caches that basically make it faster to retrieve data from memory.

SL

→ M

→ Caching is really helpful if you are doing a lot of network requests.

Eg:



You can imagine here, the client might have make request to the server. The server might then make request to the database. The database is gonna return data to the server. The server is gonna return data to the client.

Perhaps here, you might want to speed up this operation by caching the results of this network request.

Eg

→ → Another e.g in which caching is needed is when you are doing some very computationally long operation.

Eg: Imagine you have multiple servers instead of just 1 server. All of these servers are hitting the database with the same network request.

may be there are bunch of clients, they also are making individual requests to the different servers to get a certain instagram profile, for instance.

(Eg: A popular celebrity)

↓ so much

Here, you can use caching, not to use increase the speed of each network request when it's trying to get the profile. That's not something that you need to here optimize on. Individual network requests would be very quick.

But you don't want to read from dB a million times tho so you can use caching so that you do not need data from dB each and every time

↓

Read operation from dB ↓

Note:

→ Caching is used to

~~① either increase the speed of network request~~

either to speed up an operation that involves network requests by avoiding having to do network request altogether or atleast avoiding to do it multiple times.

May be you want to speed an operation bcz it's computationally long.

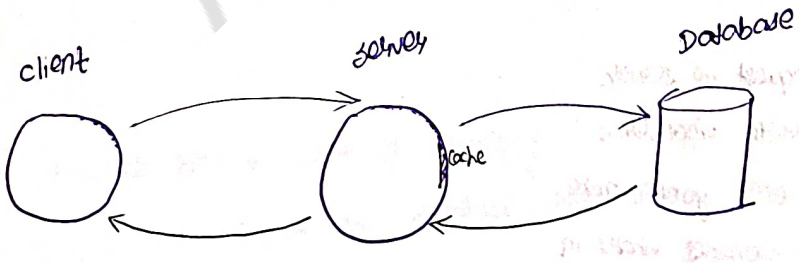
Or may be, you have an operations that is being done tons of time, you don't necessarily need to speed them up individually. You just use caching here.

Reddis → It is a very popular in-memory database.
(key-value store)

Eg:

→ Let's imagine, we are designing a web application where users can read or write posts. And they can also edit their post.

Eg: writing facebook or Linkedin post



Let's say, you are caching the posts at the server level in memory. Now, you have 2 sources of truth. Your post is stored both in database and also in cache stored in your server.

Let's say, now you're editing an post that you write.

How do you deal with these 2 sources of truth?
and how do you know that when to write to the cache and when to write to the database?

Caching

write-through cache

write back cache

→ It is a type of caching where when you make an edit to a piece of data or when you write a piece of data, your system will write that piece of data to both in the cache and in main source of truth at the same time or rather at the same operation.

Eg:
Let's say, your post is stored both in dB and in cache in the server.

And as a user, you want to edit your post.

You're gonna make a network request to server. And the server is gonna overwride whatever is in the cache. And then also, it is gonna make a request to the database and overwrides what's in the database.

On this way, the cache and dB is always in sync.

But the downside here is you still end up going to the dB.

Note: With write through caching, every single time you are gonna override something in the cache and in the database, you are doing 2 things and you still end up going to the dB.

Write-back cache:

Eg: Let's say, you are a user and you are editing your post.
You are gonna make a network request to the server and the server is gonna update the cache, ie. only the cache. And then it will immediately go back to the client.

↓

So here, your cache is not in sync with database.

Your system will asynchronously update the database with the value that are stored in the cache. It can be done either at every intervals
ie. every 5 sec or every 5 min, etc.

One of the downside here is that if something ever happens to your cache, and you lose the data in the cache before the database has been updated asynchronously, then you're gonna lose data.

↓

That's really bad.

Eg: Let's say, we have to design Youtube comment section system.

You have a bunch of servers. Every server caches in memory the comments on a single video.

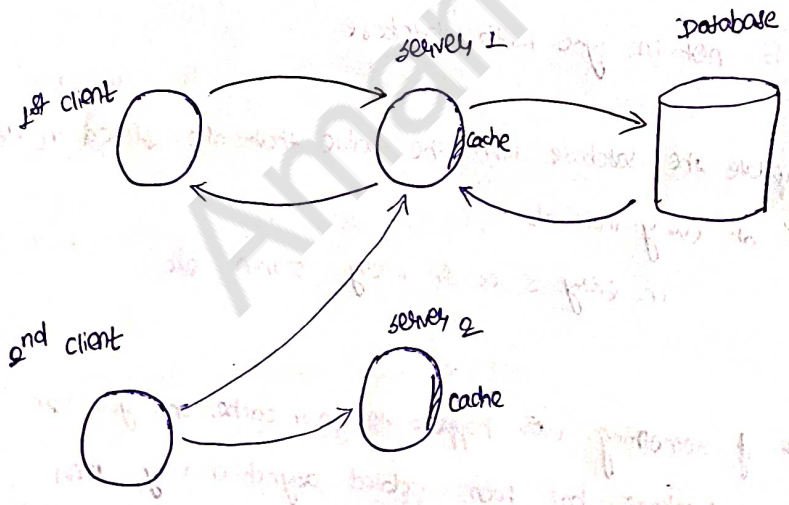
We have clients which are communicating to these respective servers. When they are reading comments, they just read from caches in their respective servers.

Now, let's say, 1st client posted a comment on the video. And our 2nd client goes to the video. The server goes to the dB to fetch all the comments and stores them in the cache.

Then 1st client edits that comment. And the 2nd ~~comm~~ client if goes to that comment sees the old version of 1st client's comment, not a newly edited one.

(Bcz it doesn't go to dB. It fetches data from cache in the server).

And 2nd client respond to the old comment, not the newly edited one



That's really bad. You don't want people responding to older version of comments.

↓

results in staleness of caches

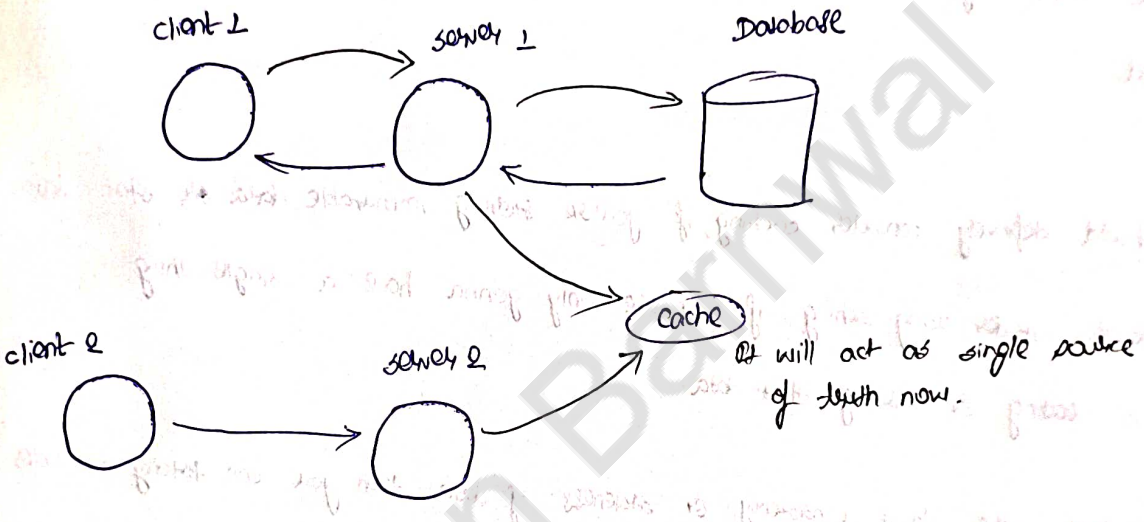
Here, our client will often deal with stale caches, ~~see~~ i.e. the caches that hold stale data.

Note:

Caches can become stale if they have not been updated properly.

On the above e.g, one of the soln is to move our cache out of the servers to and to put a single cache. And all servers hit this cache and then we'll have a (Let's say, Redis)
single source of truth for the caching mechanism.

ie.



→ For some part of the system, we might not care much about staleness or non-staleness of data in our caches.

Eg: Let's take viewcount on your Youtube video.

viewcount is not necessarily the most important info on the Youtube video.

And if one user sees a slightly stale version of a viewcount on a video, that's not trouble much or that's not a big concern.

So, these things can be discussed with the interviewer in the system design interview that do we care about the accuracy of data that much?

Summary:

Caching has a lot of pitfalls.

→ In general, if the data that you're dealing with is static data or immutable data, then caching is just beautiful.

→ But if you're dealing with data that is mutable, then you have 2 or more locations where the data exists. You have to make sure that those locations are in sync. otherwise, data might become stale and depending on your use-case, that might not be good.

(i) So, you should definitely consider caching, if you're storing immutable data or static data

(ii) You should consider using caching, if you are only gonna have a single thing that is reading or writing that data.

(iii) If you don't care about consistency or stateness of data, then you can totally consider caching.

(iv) If you are able to design a system such that you can properly validate or get rid of stale data in your caches, then you can consider caching.