**Introduction**:

Where design fundamentals come into play into whole system design interview ?

If we were to give someone with no software engineering background a canonical system design question, that person would not be able to answer it.

System design interview require a lot of knowledge about how to build robust, functional and scalable systems.

Design principles are absolutely necessary. There are lot of design fundamentals to know.

SQL

Server

Cache

Polling

Load Balancer

Leader Election

Peer to Peer

Availability

Proxies

Nginx

MapReduce

Client

HTTP

Database

Hashing

Replication

**Lecture 2: What are Design Fundamentals?**

System design interview questions are intentionally vague.

Design Uber, etc.

Turn 1 line into 45 minute discussion.

Asking questions.

Investigation

Requires lot of fundamental knowledge.

A proposed solution to a system design question may not be objectively correct or objectively incorrect.

Confidently justify my solution. Why I have designed some part of the system in one way over the other ?

Eliminate doubts that the interviewer may have. Defend the choices if the interviewer challenges it.

This is impossible to do without proper fundamental knowledge.

Design fundamentals can be put into 4 categories which build upon each other.

1. Fundamentals:

Client server

Network protocols

1. Key characteristics:

Availability, throughput, redundancy, latency, consistency.

1. Actual components: load balancers, proxies, caches, rate limiting, leader election
2. Actual tech: real life tools like Redis, Google cloud storage to build the system. Important category.

Robust, Functional and Scalable systems.

**Lecture 3**: **Client Server Model**

A client requests data from a server. And a server returns some form of data to the client.

Browser is a client.

The client does not know about the server. All it knows is that it can communicate with it. It can request stuff from it. It does not actually know what the server represents. Based on the response from the server, the client will be able to do stuff.

When we type algoexpert.io in the url bar, at first our browser does not even know how to talk to the algoexpert or the server.

What it does behind the scene is it makes a DNS query to find the IP address of the Algo Expert.

Only then it can really speak to Algo Expert.

DNS query is a special request that goes to

pre-determined set of servers and says, What is the IP address of AlgoExpert.io.

Browser makes a DNS query and gets the IP address of AlgoExpert.io.

IP address is a unique identifier for a machine.

All computers connected to Internet have ways to find public IP addresses or have ways to discover routes to those addresses.

They can send data to IP addresses. They can send packets of information in the form of bytes to IP addresses.

We can always think of an IP address as a mailbox that some entity has granted to a machine.

The entity that has granted AlgoExpert an IP address is Algo Expert’s cloud provider which happens to be Google Cloud Platform.

Google cloud has an IP address for algo expert.

Algo Expert servers are owned by Google.

On Unix machine

“**dig algoexpert.io**”

After getting the IP address, the browser knows where the algoexpert.io is. The browser sends an http request to this IP address.

HTTP is a way to send information that the server is able to understand.

When we say the browser or client sends an HTTP request to the AlgoExpert servers, what we mean is that it sends a bunch of bytes or characters that are gonna get packed into “packets” in some special format and that is gonna be sent over to Algo Expert servers. This request also contains the IP address of the browser or of the computer and this is called the source address of the request.

When the server gets this request, it is gonna know what IP address it should send a response to.

Algo Expert server will use the source IP address which is contained in the http request and use it to sends its response.

Servers are machines which wait to receive requests from other machines called clients. Once they get requests, they can send responses to those client. A server usually listens for requests on specific ports.

Any machine that has a distinct IP address has 16000 ports that programs on the machine can listen to. So when we are communicating with other machine, we actually have to specify what port we want to communicate on.

As a client, when we are communicating with a server, we actually have to specify the port that we want to communicate on.

Mailbox analogy

We can think of the IP address as the mailbox to an apartment complex. And then we can think of the ports as the actual apartment numbers that the mail arriving to that mail box has to route to.

Most clients know about the ports that they should use depending on the protocol that they are using for speaking to the server.

For example, if a client is trying to speak to the server using http protocol, it is always going to use port 80.

http: port 443

“**nc**” netcat command

Used to read from or write to network connections using certain network protocols.

It is going to allow us to visualize this whole thing of communicating to an IP address using specific ports in action.

nc -l 8081

-l is for listen. Listening on port 8081.

nc 127.0.0.1 8081

(IP address 127.0.0.1 points to the local machine, that is local machines/computer’s IP address.)

This terminal here is basically entering a communication channel with the machine at IP address 127.0.0.1 at the port 8081.

If we start writing anything on second terminal, we will see that on the first terminal.

We are writing the data in 2nd terminal and reading it in terminal 1.

Cool way to visualize what sending data or information to an IP address on a specific port and listening through that specific port looks like.

The algo expert server is able to read the request because it understands the http format.

When we try to go to algoexpert.io we are able to see the html of algoexpert.io.

The server returns the html of algoexpert.io to the client. The browser receives that response and renders the html on the page for us.

**Lecture 4**:

IP

TCP

HTTP

A protocol is an agreed upon set of rules for an interaction between 2 parties.

For example, communication protocol when 2 humans interact with one another.

Network protocols are for communication between 2 machines.

Clients and servers are machines which interact with one another following network protocols.

Network protocols consist of kinds of messages that are gonna be sent and received by machines, by clients and servers

We can hear some terms like: Messages sent over the wire, over the network.

These expressions are synonymous. They really mean “sent over the internet” or from one machine to another machine.

Network protocols consist of types of messages that are gonna be sent over the network.

The format of these messages, how they are structured, the order of those messages if they have an order and whether or not there should be some sort of response to a message.

If there is a response, what that response should look like. Whether or not there should be rules around when messages can be sent to one another.

There are a lot of network protocols out there. The majority of them, we don’t need to know about atleast in the context of system design interviews.

Internet protocol address

The modern internet runs on IP. Following the internet protocol.

When a machine tries to interact with another machine and it sends data to that other machine, that data is sent in the form of, what is known as an IP packet.

We can think of the IP packet as the fundamental unit of data sent from one machine to another. IP packets are the building blocks of communication between machines over the Internet.

There are other units as well, beyond IP packets because IP packets are made up of bytes.

IP packets have 2 main sections. The 2 sections are known as the IP header and the data.

IP header is the section of an IP packet that is gonna be at the beginning of the packet.

The header contains useful information about the packet. All the information in the packet is stored in bytes.

The header contains information about the source IP address of the packet (IP address of the machine from which this packet is coming from), destination IP address (IP address of the destination to which this packet is going to).

If we have a single IP packet, we know where that IP packet is coming from and where it is going to.

This is how information flows over the network on the Internet. This is how information knows to go from one machine to another because all the IP packets have the source and destination IP addresses of the machines they are coming from and going to.

The header also has other information like total size of the packet.

From the header we can know what the total size of the packet is and header has the version of Internet protocol that this IP packet is operating by.

There are multiple versions of the internet protocol. Today, there are 2 versions in practice.

Version 4, IPv4 and version 6, IPv6.

In the header, we can see the IP version of the packet.

Based on the IP version, the packet might look a little different.

It might be structured a little differently and a machine will know how it should interpret it based on that version.

Header is between 20 and 60 bytes.

Rest of the part is the “data” part of the IP packet. Here the information that one machine is trying to send to another machine will be stored.

IP packets are limited in size.

2^16 bytes.

If we are sending information, then that may not fit into 1 IP packet. It will fit into multiple IP packets.

When multiple packets are sent from one machine to another, if all we are using is the internet protocol, we don’t actually have a way of guaranteeing that these packets are actually going to be received.

Some packets may get lost over the network and this may prevent sending all of the data that we were trying to send.

We are also not guaranteeing the order in which the packets will be read or interpreted.

The internet protocol alone, kind of falls apart here.

This is where TCP comes into play.

TCP is Transmission Control Protocol. It is built on top of the Internet Protocol and is meant to solve the issues that we just discussed.

It is meant to send IP packets in an ordered way, that is we are guaranteeing the order in which those IP packets will be read by the destination machine in a reliable way, meaning we are guaranteed to get those packets actually received by the destination machine or we will know if some packet get failed or keep failing from getting received and in an error free way.

If the packets get corrupted for whatever reason when they are sent over the network, we will know and we will resend those packets to make sure that they are received in an uncorrupted way.

This is what TCP aims to solve in a nutshell.

TCP is used in almost all web applications. It allows us to send arbitrarily long pieces of data to other machines.

If we have an IP packet, we have got the IP header, data portion.

In the data portion, we will have a TCP header. TCP header contains important information about the TCP part of the IP packet.

In this, we have information about ordering of packets.

The core idea behind TCP is that when a machine wants to communicate with another machine over TCP, for instance, when a browser wants to communicate with website servers (for example with AlgoExpert’s servers), it first creates a TCP connection with the destination computer/server.

The way this happens is through a handshake.

A handshake is a special TCP interaction where one computer contacts the other by sending 1 or more packets, saying hey I want to connect with you? The other computer responds and says, okay we can connect and we can chat.

Then the machine that was trying to establish the connection re-responds again and says, we are now connected and we have got an open connection.

This is known as a handshake. Once this connection is established, both machines can freely send data to one another.

If one of the machines does not send data in a given amount of period, the connection can be timed out.

If one machine wants to end the connection for whatever reason, it can do so by sending some sort of special message to let the other one know that, hey I am about to end the connection and then the TCP connection is done.

It is a more powerful or a functional wrapper around IP but still what it lacks is a really robust framework that developers, software engineers can use to define meaningful and easy to use communication channels for clients and servers in a system.

With TCP, all we are really sending is arbitrary data that fits into the underlying IP packets and this is where HTTP comes into play.

Hyper Text Transfer Protocol is a protocol built on top of TCP. It introduces higher level abstraction above TCP and thus above IP.

This abstraction is the request, response paradigm.

HTTP protocol introduces the idea of having machines communicate with one another following this request/response paradigm where one machine sends a request to other machine and that other machine returns a response to the first machine.

This request/response paradigm with a set of accompanying rules makes it very easy for developers to create robust and easy to maintain systems.

This is why modern day systems rely on http protocols for communication.

With HTTP, we completely forget about IP packets, we forget about TCP.

All we deal is with HTTP request and responses.

Requests are gonna be things that machines that want to interact with other machines send and these requests are going to have a lot of properties defined by HTTP protocol.

Visualize what these requests and responses look like. We can think of them as objects with important fields and properties that describe them.

Various type of http requests like GET, PUT, DELETE, POST describe the purpose of the request.

If we have a request that has ‘GET’ as its method, then that request is going to retrieve data from the server.

If we have a ‘POST’ request, then that request is going to provide data to the server.

‘DELETE’ request asks the server to delete some data.

We, as the developer of an API or a system that relies on HTTP, **we can implement our server’s logic however we want**.

These methods and their intended use case are just guidelines. How we are gonna use them will depend on our server.

A server might have multiple paths for different services and clients are going to issue requests to these various paths.

Depending on the path, different business logic is going to occur.

Path separates out logic on the server.

We can think of headers as a collection of key-value pairs that contain important metadata about the request.

“content-type” defines the type of the body.

“content-length” defines the length of the body.

In the response, statusCode describes the type of response this is.

Specific status codes are suppose to mean specific things.

Code the client-server in both java and javascript.

“**curl**” command: helps to send data to servers and retrieve data from servers using different protocols.

By default, it uses http.

IP and TCP were only for transportation of data. HTTP introduces the opportunity to add a lot of business logic which is what we want if we are developing a large scale system or any kind of system.

The end points that we define, we can define a lot of business logic which is what HTTP gives us.

HTTP will help to implement business logic in our system.

**Lecture 5**: **Storage**

Storage concepts

Any system that we want to design requires some form of storage.

This is where databases come into play.

Databases server 2 main purposes: Store and retrieve data.

Storing, retrieving data

Reading, writing data

Setting, getting data

Recording, querying data

All these word pairs can be used interchangeably when we are talking about storing and retrieving data in the context of system design interviews.

Database allows us to store and retrieve data.

What a database almost always is just a server.

Our machine can serve as a database for a system. That might not be the best decision to make for our system. The point is that it can be.

Another concept with databases is persistence.

Persistence of data that we store in the database.

It is not always the case that data in a database will persist if there is a power outage, some network issue, etc.

This leads to 2 fundamental things in storage: Disk and Memory.

If we have a database server and that database writes data to a disk, that data will persist even if the database server goes down.

Writing data to a disk is what happens when we save a file on our computer.

If we shut the computer afterwards or if the computer crashes, the file that we saved is still going to be there unless there is some catastrophic issue with the hardware or some exceptional issue, the file is still gonna be there.

In contrast, if the database writes data in memory (that is we might store data in an array, hash table, etc), if the database server goes down and then gets booted back up, the data structure in which we might have stored data is no longer going to have that data.

This is what essentially means to store data in memory.

Reading data from memory or writing data in memory is much faster than reading data from disk or writing to disk.

‘fs’: Javascript library that lets us write data to a disk and data from a disk.

We can specify that the server expects json format from the requests.

By default, http requests made through curl command are ‘GET’ requests.

Can write post routes to write data to a file on the disk and also store it in memory in a data structure.

If we turn off the server and try to send get routes to disk and memory again, then the data only persists on the disk and not on memory.

Storage is complex.

Google cloud platform offers 8 different storage products.

There are a lot of different things that databases can offer us.

There could be a structure imposed by databases in which databases store data.

Database could be a critical part of our system.

Distributed Storage

Could need to store on multiple machines.

Consistency issues (Up to dateness).

**Lecture 6: Latency and Throughput**

Latency and throughput are often misunderstood.

Key is understanding what these 2 words mean.

Latency and throughput are important measures of a performance of a system.

Latency is how long it takes for data to traverse a system, that is how long it takes for data to get from one point in a system to another point in the system.

We might be talking about latency of a network request. That is how long it takes for a request to go from client to a server and then back from server to the client.

If a machine is reading data from memory or disk, the time it is going to take to read this data is also going to be referred to as latency.

Different things in a system have different latency.

There is a trade-off between different ways that a system is built.

Certain things have high latencies and other things have lower latencies.

Important to grasp the order of magnitudes of different latencies and how they compare to each other.

If we are reading 1MB from memory, that is going to take roughly 250 microseconds.

Reading 1MB from a solid state drive, that is going to take 1000 microseconds.

Sending 1MB data over 1Gbps network, this takes 10,000 microseconds.

We are assuming that we are sending this 1MB of data over the network to a machine or computer that is right next to us.

If we are making a network request, if we are making a web application for instance we are likely to be making API calls, making network requests.

When we are reading from memory, we can think of reading a variable in our code.

Reading 1MB over a hard drive, HDD is going to take 20,000 microseconds.

If we are sending a packet (lot smaller than 1MB) over a network from California to Netherlands and back to California, this is going to take 150k microseconds.

**Note**: Packet is considerably smaller than 1MB.

Sending stuff around the world takes a lot longer than reading stuff from memory or disk or sending over network without counting physical distance.

Electricity has to travel and it takes time when it has to travel long distances.

When designing systems, we would like to optimize systems by lowering the overall latencies of the system.

Some systems won’t necessarily warrant optimizing for latency as much as others.

Certain types of systems might care about having low latencies. A good example here is video games.

Multiplayer games have high latencies.

‘lag’

This might literally be because the server that we are playing on is located halfway across the world from us and it takes a while for our computer or client to make network requests to the video game server. If we are sending a lot of data over the network, then this 150k micro seconds quickly add up.

If we need instantaneous action in our game, then this will be a really bad experience for the user.

High latency: Poor system

Some websites might rely on accuracy and may have high latency.

We have to think of these kinds of trade-offs or the priorities we have when we are designing a system.

Latency is definitely a thing which we want to consider.

**Throughput**: How much work a machine can perform in a given period of time ?

How much data can be transferred from one point in a system to another point in a given amount of time.

We measure this throughput in Gbps or Kbps or Mbps.

Suppose we have got a bunch of clients. All of these clients send requests to a server.

The throughput is going to be how many of these requests can this server handle in a given amount of time.

If we reduce these requests to bits, how many bits can this server handle or let through per second?

We can only fit a fixed amount of bits in a given amount of time (seconds).

How do we increase throughput or optimize a system for throughput?

Simple answer is: We pay to increase throughput.

The thing that determines the number of bits that can go in or out of algo expert’s servers at any given time or during a second is determined by their cloud provider, GCP. Algo Expert can pay Google to just increase their throughput.

Just increasing throughput does not necessarily fix a potential problem that we may have in a system.

The system that is getting requests from many clients could be Google Search, Facebook Messenger that might expect to serve thousands of requests, perhaps even millions of requests per second.

Just trying to blindly increase throughput on this network probably won’t make sense because we still have some bottleneck here.

To fix this simple system would be to have multiple servers handling all of these requests.

Even though latency and throughput are very much related and they are a very important measure in a system’s performance, they are not necessarily correlated.

We may have parts of system that have very low latency but some other part of the system may have very low throughput.

We cannot make assumptions on latency and throughput based on the other. They are not necessarily correlated things.

**Lecture 7: Availability**

SLA/SLO Nines Availability.

When we are evaluating a system, 2 of the things that we want to think about the system’s latency and throughput.

Another thing we want to think about is the system’s availability.

Availability is how resistant a system is to failures. For instance, what happens if a server in our system fails or what happens if the database fails. Is the system going to completely go down or the system is still going to be operational ?

This is often described as system’s fault tolerance.

How fault tolerant a system is.

Another way to think about availability is the percentage of time in a given period of time (in a month or a year or atleast operational enough such that all of its primary functions are satisfied.)

Availability is an important thing to think about when we are evaluating a system.

Most systems almost have an implied guarantee of availability. An example of this would just be AlgoExpert.

When we purchase AlgoExpert, the main thing that we purchase is access to the platform, access to the content, the ability to watch the videos. That is the main thing we get when we purchase algo expert.

There is implied guarantee of availability that comes with the purchase of AlgoExpert.

If we go to algoexpert.io, we expect the website to be up and expect the website to be fully operational.

If the website ever is not fully operational, we are not going to be happy.

To the designers of algo expert, availability matters a lot.

System designers care about the system’s availability because there is an implied guarantee of availability.

There are varying degrees of availability that we might expect from different systems. In case of Algo Expert, if the website were down for a few hours or if the parts of the platform were not operational for a few hours, it would be upsetting for customers and bad for service providers.

But it would not be the end of the world or most unacceptable thing to ever happen.

Now imagine that we were dealing with a system that supported airplane software. The software that allow an airplane to properly function when it is in flight.

If the software goes down when the aeroplane it flying, that would be absolutely unacceptable.

In this system, we expect extremely high amount of availability.

Any amount of complete downtime in the system would just be unacceptable.

If youtube goes down, that would be really bad.

If parts of these cloud provider system ever go down, that would be really bad.

It then affects all of the businesses, customers that rely on their services for their own platforms.

Availability matters a lot.

How do we measure availability?

We can measure availability as the percentage of the system’s uptime in a given year.

50% availability would be really bad and the products may not survive in the market.

Even an availability of 90% is not really great.

In the industry, most services or systems aim for high availability.

We often end up measuring availability not exactly in percentages but we call Nines.

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

If we have a system with 99% availability, that is it is up 99% of the time during a year, then in the industry we say that the system has two 9s of availability.

If the system has 99.9% availability then we say that it has 3 9s of availability.

99.99%: four 9s of availability.

99% availability sounds really good but for a lot of services being down more than 3.5 days in a year is still really bad and unacceptable.

For systems that serve billions of users (Fb, Youtube), 3.5 days of downtime a year is not good.

99.999% (five nines) translate into 5.26 minutes of downtime per year.

Five nines of availability is typically regarded as gold standard of availability.

If a system has 5 nines of availability, then we can say that it is a highly available system.

HA: abbreviation

A lot of systems aim to be highly available systems because they really care about availability.

Availability is really important and it is something that *matters a lot both to the end users of the system and to system designers*.

In fact it matters so much that for certain systems we don’t have an implied guarantee of availability.

We have an explicit guarantee of availability.

Many services providers have what are known as SLAs.

**SLA**: Service Level Agreement

SLA is an agreement between a service provider, basically people who are behind a service or a system that is being sold or provided and the customers or end users of this service or system.

An agreement on that system’s availability amongst other things.

Many service providers have written SLAs that basically tell customers about a guarantee of some amount of availability (guarantee about some percentage of uptime in the system).

SLO is related to SLA but is not the same thing as SLA.

**SLO**: Service Level Objective

We can think of SLOs as components of an SLA.

A service level agreement is made up of service level objectives.

If we provide a service to a customer and we guarantee a percentage of uptime for that service. That percentage of uptime guarantee would be an SLO.

All the major cloud service providers like GCP or AWS have very clear cut SLAs.

We can see them online on their product pages. These are very important when we are considering buying their services because depending on our need for their services, we might require some percentage of availability.

We will want something clearly and unambiguously specified in an SLA to give us that piece of mind.

Cloud Spanner is a database that Google cloud platform offers.

If GCP is unable to maintain 4 or 5 nines of availability, then they back customers a portion of their monthly bill.

SLAs are very serious and availability is taken very seriously.

Availability is very important in systems design.

Availability may not be always super important when designing a system or it is not always the case that we need to have five nines of availability because having high availability comes with

trade-offs.

It is difficult to achieve high availability.

We as a system designer have to think whether or not our system should be highly available.

We have to think long and hard about what parts of our system should be highly available or don’t need high availability.

Stripe provides payment services for businesses. For Stripe, their core services of handling payments and charging customers, that is likely a highly available service.

If this service would ever go down then stripe and their customers could lose 1000s$ of revenue.

This part of the system has to be highly available. But other parts of their system might not have to be highly available.

For example, they have a dashboard that businesses can use to monitor their sales. This dashboard is not as critical as the core payment service that Stripe provides.

If the dashboard goes down, then still it is pretty bad, but it is not the end of the world.

When we are going to be designing systems in our system design interviews or even on the job, we have to gonna ask ourselves, What parts of our system are absolutely critical to the point that they require high availability and what parts of my system do not require high availability

What parts of my system would be ok to fail?

We will have to think about this.

How do we actually improve the availability of a system?

We have ensure that our system does not have single points of failure.

Single places in our system that if they fail cause an entire system to fail.

How can we eliminate single points of failure?

Using redundancy

Redundancy is multiplying certain parts of our system.

If we have a very simple system in which clients interact with a server and then our server interacts with a database, we can clearly see that the single server is a single point of failure.

If this single server gets overloaded or if there is a problem with it for whatever reason and it dies then our entire system fails.

We will want to make part of the system (server) redundant and we can do so by adding servers.

If we want to add servers, we would want to have a load balancer in between our clients and servers in order to distribute the load of our clients across the three servers.

Now our load balancer is a single point of failure.

What if our load balancer becomes overloaded?

We can add redundancy at the load balancer layer in the system.

AlgoExpert uses 5 load balancers.

We can add redundancy in any part of our system literally just by adding machines to those parts of our system.

Multiple servers and multiple load balancers: Passive redundancy

Passive redundancy is loosely speaking when we have multiple components at a given layer in our system.

At any point of time if one of the component dies nothing is really going to happen.

Example of passive redundancy

Airplane engines,

When we dealing with a twin engine airplane, an airplane that has 2 engines, both engines are being used.

But if one engine fails the airplane can still fly completely smoothly with just one engine.

On the other side, we have active redundancy.

Active redundancy is bit more complicated. It is when we have multiple machines that work together in such a way that only 1 or few of the machines would be typically handling traffic or doing work.

If the machine handling the traffic fails, the other machines will somehow know that the other machine failed and they are going to take over.

If in our system, if we had a special service that was handled by 5 machines, suppose 5 servers were in charge of one specific service.

Only one of these 5 machines at any given point of time were responsible with the main duties of this service in our system.

And if that 1 machine was responsible for the duty of a service, and imagine that it died then the other 4 machines will re-configure themselves and take over the job of that failed machine, that would be active redundancy.

We will also need to make sure that we have rigorous processes in place to handle system failures.

System failures are gonna require human intervention. If servers in a system crash, we will need a human to bring them back up and we will need to have processes in place to ensure that happens in the proper time frame.

**Lecture 7: Caching**

Caching is one of the important techniques in system design.

We may be using caching in almost all of my system design interviews.

Avoid doing computationally complex operations that may take a lot of time multiple times.

We use caching to improve time complexity of our algorithms to speed up our algorithms.

Caching is used to improve latency of a system.

Caching is a way to design a system in such a way that if we originally going to be using types of operations or data transfers that take a lot of time, like for example, network requests, we will design the system in such a way that we don’t have to do those network requests and we can do different types of operations, different types of data transfers that are going to be faster.

Caching is going to be storing data in a location that is different from the one where the data originally is such that it is faster to access data from this new location.

We can use caching in a bunch of different places in a system. We can cache at the Client level.

Client can cache some data value so that in longer has to go the server in order to fetch it.

We can also cache at the server level. Maybe, we always want the client to interact with the server but the server does not always need to go to the database to retrieve data.

Maybe, it only needs to go to the database once and we can have some cache at the server level.

We can also have some cache in between 2 components in a system.

We can have a cache in between a server and a database.

We can even have caching at the hardware level. In modern day computers there is some caching that happens at hardware level.

There are CPU caches at the CPU level. They make the retrieval of data from the memory faster.

Caching can occur by default at many different levels of the system.

The instance in which caching will be helpful is when we will be doing a lot of network requests and we want to avoid doing all of these network requests.

The client can make a request to a server and the server might make a request to a database. The database is going to return stuff to the server.

The server is going to return stuff to the client. And finally, the client might be able to do stuff with the data that it requested.

Perhaps, we want to speed up this operation by caching the result of this network request.

If the client ever has to do that network request again, it will be cached and we won’t have to go all the way to the database.

Here, we can put the cache at the client level or the server level.

Another instance where caching is very helpful is when we are doing some very computationally long operations.

We might want to avoid some other computationally long operation. When a client sends a request to server, then the server might perform very complex algorithm, maybe an algorithm which has poor time complexity.

And then server returns the result to the client. Maybe we just want to cache that.

Goal behind caching is to speed up the system.

Another instance where caching is going to be very useful is: Imagine we have multiple servers instead of just 1 server and other clients that are communicating with these servers and all of these servers are hitting the database.

They all hit the database with the same network request that the first server does.

We have got a bunch of clients doing the same thing. All making individual requests to different servers to get a certain Instagram profile for instance.

Suppose there is a popular celebrity and there are million users who are trying to view that celebrity’s profile.

Here we can use caching not so much to increase the speed of each network request when it is trying to get the profile because that is not something we want to optimize on. Individual network requests would be very quick.

But maybe we don’t want to read from the database a 1000 times or a million times because that might overload the database for instance.

So we will use caching to not have to read from the database that many times.

We may have a cache somewhere in between the server and database. The servers would communicate with the cache instead of the database. Or may be each server may have its individual cache and we can store the result of the network request, that is the Instagram profile in memory at each server.

And this would prevent the system from doing so many reads at the database level.

We can do caching to:

* Avoid doing a network request altogether.
* Avoid doing a network request multiple times.
* Speed up an operation because it is computationally long.
* We have an operation that is done tonnes of times. We don’t need to speed them up individually but we just don’t want to perform it that many times because it might affect our system in other ways and we might want to use cache in there.

If we go on Algo Expert and go on Questions List, we may notice that first time we go on that page there is going to be a Loading icon. Depending on how fast our Internet connection is, we might see that loading icon for just a split second or for a few seconds.

If we go to another page afterwards, we do not close our browser or the tab and if we go back to the same questions list, we will notice that the icon is no longer there.

And the questions list is pre-loaded, that it is there on the page immediately.

We actually cache the questions list on the client. We know that the questions list is a static piece of content. (ignoring marking the question as completed, etc).

The questions list is static content.

Instead of having the user or the client make a network request to our server everytime they go on the page to see the questions, we just cache the result on the client after the first time we make the network request. And that way we just get a faster experience on the website.

Another example of caching would be when we are running code on AlgoExpert. Running code on Algo Expert on average takes about 1 second.

When people run code on AlgoExpert with the solution provided on the website, they don’t need to do that 1second computation everytime.

What we can do is cache the results for running code with our solution (AlgoExpert’s).

When users do that, we just return the values that are in our cache. That is going to be much faster (order of milliseconds rather than 1 full second or even more).

How do we cache the results of running code?

When we are running code, we are making a network request.

We can store the cache at the server level or detached from the server into some independent component.

Here we can use something like Redis, which is a very popular in-memory database. It is actually a key-value store.

The way that this caching would work is the client, we users would go on AlgoExpert on the coding workspace, we would run the code with one of AlgoExpert’s solutions.

This would send an HTTP request to the AlgoExpert server/s.

This request would be of a bunch of bytes.

The server would hash the request/these bytes down to a single signature, very much like a hash table. It might hash it down to an Integer.

And then we check our cache which might either be in-memory on server or in some detached component like Redis.

If the ‘key’ is in our cache, we take the value associated with that key and we return it to the client. We have avoided doing that additional 1 second computation of running the code. We just used the value that we had in the cache.

Can write a simple program for server with database. Can show the difference with using a cache and not using it.

The examples of caching that we have covered so far involve storing data that only needs to be read.

Not data that needs to be written.

Let’s imagine that we are designing a system or a web application where users can read and write posts. Users can edit their posts.

We can imagine writing Facebook or LinkedIn posts.

The client who is writing a post makes a request to the server to write the post. Then the posts are stored in the database.

If we want to cache those posts, let’s say we are caching the posts at the server level in memory.

We now have 2 sources of truth. Our posts are stored both in the database and in the server.

The client made a network request with our new post and the server made a network request to the database and stored the post in the database.

And then we want to display the post on the page and somehow the post got stored in the cache.

How do we deal with these 2 sources of truth ?

How do we know when to write to cache for instance and when to write to database ?

We do that at the same time or do not do that at the same time?

Here we will cover two popular types of caches.

First one is a Write-Through cache.

Write-through cache is a type of cache in a system where when we make an edit to a piece of data or write a piece of data, our system will write that piece of data both in the cache and in the main source of truth at the same time, that is in the same operation.

We have our post stored in the database and also in the cache and we want to make an edit to the post as a user.

We are going to make a network request to the server and the server is going to overwrite whatever is in the cache.

And then it is also going to make the request to a database and overwrite what is in the database.

This way, the cache and the database are always in sync.

Downside of this is that, we still end up having to go to the database.

Whereas some of our previous example of caching, we were able to avoid some of the network calls or certain operations.

Here, with this Write Through caching, every single time we are going to overwrite something in the cache or in the database, we are going to be doing 2 things and we are still going to be go to the database.

Other popular type of cache is Write Back cache.

If a user is editing a post, he will be making a network request to the server and the server is going to update only the cache and it will immediately go back to the client.

Cache will be out of sync with the database.

Behind the scenes, the system will asynchronously update the database with the values stored in the cache.

This can be done in several ways, in certain intervals, every 5 seconds, every 5 minutes, every 5 hours, etc.

Or it may follow some other schedule like for example, when the cache gets filled up and we have to evict stuff out of the cache.

With write back caches, whenever the user makes a network request to the server to edit their post, only the cache will be updated and then asynchronously database is going to get updated.

Downside: If something happens to our cache and we lose the data in the cache, for instance before the database has been updated asynchronously, then we are going to lose data.

With large systems with many different components, suppose we have to design YouTube comments system. And we have decided that we have got a bunch of servers.

Every server caches, in memory the comments on a single video and we have got clients that communicate with the servers.

When they are reading comments, they just read them from the caches in the respective servers.

Suppose our first client has posted a comment on a video.

Now, the second client goes to the video.

The server goes to the database first to fetch all the comments and stores them in the cache.

And then some time elapses, the clients do whatever and eventually the previous client goes back and edits its comment.

The second client goes back to the video but this time the server does not go back to the database because it already has all the comments already stored in the memory (cache).

The 2nd client sees the older version of first client’s comment and not the newly edited one.

The 2nd client responds to the old comment and not the newly edited one.

This will be unacceptable for YouTube comments.

Concept of staleness

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

In this system, clients will be dealing with stale caches.

In this system, the solution would be to move the cache out of the server and put a single cache in the middle which could be Redis.

We will have a single source of truth for the caching mechanism.

On the other hand, for certain features/parts of our system that we are trying to build out with our system we might not care about staleness or

non-staleness of the data in our caches.

Lets take ‘view count’ on YouTube videos. View count is not necessarily an important piece of information on a youtube video.

If one user sees a slightly stale version of ‘view count’ on a video, then that is probably not going to be the end of the world.

This is stuff we have to keep asking interviewers and ourselves when we are in a systems design interview.

What are the things that we are trying to build out?

What are the requirements?

Do we care about accuracy of data that much?

Caching has a lot of pitfalls. Watch for those pitfalls.

If the data that we are dealing is static or immutable data, like for instance the questions list on AlgoExpert, then caching is beautiful.

It typically works very easily. But if we are dealing with data that is mutable, then things are going to be trickier because we will have 2 different locations where data exists. We have to make sure that these locations are in-sync otherwise the data might become stale and depending on our use case, it may not be good.

As a rule of thumb, we should definitely consider using caching if we are only storing mutable or static data.

We should consider caching if there is only single thing reading or writing that data because the second we introduce multiple things, things becomes a bit more complex.

If we do not care about consistency, if we do not care about staleness of data, then we can totally consider caching because we might not have to worry about potential pitfalls of caching.

Or if we are able to design our system in such a way that we can properly invalidate or get rid of stale data in our cache/s, specially in a distributed manner, if we are dealing with a distributed system, then caching is something we would like to consider.

Eviction policies with caching

We cannot store infinite amount of data in a cache but also sometimes we are gonna be left with stale data in our cache and we will want to get rid of that stale data. This is where we get into eviction policies.

What policy or rules do we follow to get rid of data in caches ?

**LRU policy**: Get rid of least recently used pieces of data in a cache and we have some way of tracking which piece of data is least recently used.

We make the assumption that the data which was used least recently is likely the one that we no longer or least care about.

There is also a least frequently used policy where we have a bunch of data in our cache. The least frequently used data, not necessarily the least recently used is the one that we get rid of.

We can also get rid of data from a cache in FIFO or LIFO basis or even just randomly. There are lots of ways to evict data from a cache.

This will depend on the use case.

Talk with the interviewer to figure out about what things are valued.

**Lecture 8: Proxies**

Important fundamental concept in systems design.

Reverse proxies are used in tv shows and movies during a hacker scene, where people are like “We can’t see the hackers because they are hiding behind a proxy”

In the remaining video, we will refer to a proxy as a forward proxy. In the industry, people use the term ‘proxy’ pretty loosely and when they use ‘proxy’, they often mean ‘forward proxy’.

A forward proxy is a server that sits in between a client or a set of clients and another server or a set of other servers.

A forward proxy is a server that acts on behalf of a client or clients. We can think of forward proxy as being on the client’s team or on the client’s side of an interaction between a client and a server.

In practice what this means is that, if a client wants to communicate with a server. Assuming that the forward proxy has been configured correctly by the client, when a client is going to issue a request to the server instead of going directly to the server, it is first going to go to the forward proxy which then is going to forward the request to the server.

We can think of it as a client does a request that is meant to go to the server but first goes to the forward proxy.

Client is asking the forward proxy to communicate with the server on its behalf. The server gets the request from the forward proxy and not directly from the client.

When the server responds, it is going to gives its response back to the proxy and the forward proxy is going to give the response to the client.

To reiterate, if a client wants to communicate with a server but has a forward proxy configured or setup then the client is going to ask the forward proxy to communicate with the server on its behalf.

A forward proxy can serve as a way to hide the identity of a client that is requesting something from a server because when a client issues a request that goes to the forward proxy, the forward proxy then forwards the request to the server.

But the source IP address is going to be in the request that is going to be sent to the server from the forward proxy.

The source IP address is going to be IP address of the forward proxy.

The source IP from the initial request would be the client’s IP address and it can get removed and replaced by forward proxy’s IP address.

There are some types of forward proxies that might still make the client’s IP address retrievable or visible in some way to the server but typically the original client IP address is not going to be visible.

This is basically how VPNs work.

VPN hides our identity.

A client may be able to access restrictive servers which it otherwise cannot access with help of forward proxy.

Using this, we can access websites unavailable in our country or are restricted by our organization.

Reverse proxies are a bit trickier.

Reverse proxies act on behalf of a server in an interaction between a client and a server.

If a client wants to interact with a server, that is client wants to send a request to a server. If the reverse proxy has been configured properly by the server or by the entity that owns this server then when the client is going to issue a request to the server, the request is actually going to go the reverse proxy.

Client won’t know that its request is going to a reverse proxy.

Client will think that it is sending the request to the destination server. But it will actually be going to the reverse proxy.

The arrows will look similar to that in case of forward proxy but the meaning is different.

The client is going to issue a request to a server but if the reverse proxy has been configured properly, the request is actually going to go to the reverse proxy. Then the reverse proxy is going to forward the request to the server.

Then the server returns a response to the reverse proxy and finally reverse proxy returns the response to the client.

The key thing here is that the client thinks that it is interacting with the server. To the client, there are no two other entities here. It thinks that there is just one entity.

The client thinks that the one entity is the reverse proxy.

If AlgoExpert.io or google.com used a reverse proxy and have it configured correctly, what is going to happen is that the DNS query is going to return the IP address of the reverse proxy and not one of the actual server.

When a client sends in a request, the destination IP for the client is the reverse proxy. Reverse proxy forwards or re-routes the request to server with IP address ‘B’.

The response goes back to ‘R’ and then eventually back to ‘A’.

Client has no idea that there were 2 entities that it effectively interacted with.

In forward proxy, the server has no idea that the client and the forward proxy are one and the same things.

Reverse proxies are really helpful.

When we will be designing complex systems, reverse proxies are definitely the tool that we want to have in our toolbelt.

We can configure reverse proxies such that it can filter out requests that we want to ignore.

Maybe for our system, we don’t want our servers to ever deal with certain kinds of requests. We can have the reverse proxy to filter them out.

We can configure reverse proxy to take care of logging for our system. If we want to log stuff, gather metrics, maybe our reverse proxy can do that.

If we want to cache certain things like html pages, we might be able to do that at reverse proxy layer. That way our server may not get bothered too much.

Perhaps the best use case of a reverse proxy is using a reverse proxy as a load balancer.

A load balancer is a server that is gonna effectively distribute or that can distribute load like request load between a bunch of servers.

If we are designing a very complex system we might have more than 1 server. The load balancer can decide to what servers all of the incoming requests from the clients go to.

The reverse proxy can distribute the requests across the servers following a specific pattern.

This might also have security ramifications.

Imagine that we had a client, that is a malicious client that wanted to bring down a server, maybe by issuing a ton of requests to a given server.

The reverse proxy can act as a shield for that because it will distribute again as a load balancer.

Distribute the requests, maybe evenly among various servers such that no single servers gets all of the requests and gets taken down by the malicious client.

**Nginx**: web server that can be used as a reverse proxy.

nginx.conf file

**Lecture 9: Load Balancers**

Used in almost all of the complex systems.

Look at a simple use case of a load balancer.

Suppose we had multiple clients issuing requests to a server.

One of the client might be issuing more than 1 request. Our single server has limited resources.

Our system has limited throughput.

Our single server can handle limited amount of requests in a given amount of time.

The more requests that are being sent to the server either from many clients or either all from a single client, the more requests are being sent to our server, the more likely our server is to become overloaded to receive more requests that it can handle.

It might lead to a failure in our system or might make our system very slow.

The simple answer is: We scale our system.

And we have a couple of ways of doing this.

The first way is to vertically scale our system meaning to increase the power of our server.

There is also limitation to the increase in power or performance of a single server, of a single machine.

Second option is to horizontally scale our system, meaning to add more machines or to add more servers to our system.

If we have many servers, we can handle requests from clients in a balanced way, assuming all the servers have the same resources, have the same power, our system will be able to handle n times previous load with only 1 server.

We are assuming that all of our clients are going to be issuing requests to our servers in a balanced way.

How did our clients know to issue their requests or direct their requests to the servers in this way ?

How come none of the clients issue the requests to the same server leading to the same problem that we had before?

This is where load balancers come into play.

A load balancer is going to be a server that sits in between a set of clients and a set of servers.

Can also place load balancers at other places in a system.

Load balancer balances workload across resources.

Re-routing requests of clients to the servers in a balanced way.

Load balancer redirects the requests in a balanced way or in some pre-configured way to the servers.

Now we have achieved the goal of horizontally scaling our system as it grew.

Load balancer also improves the throughput of the system. If we have multiple servers that are not being overloaded, then the throughput of the system is going to go up.

Our system might naturally get better latencies because the servers are going able to respond to requests faster because they won’t be clogged or bogged down.

And we might make better use of our resources because as we are adding servers as we are horizontally scaling our system, the load balancer is going to know to make use of the new resources by distributing traffic to these new resources.

We will able to make use of them and alleviate the load from other servers.

We can think of load balancers as reverse proxies most of the times.

Because load balancer acts on behalf of the servers and sits in between client and servers.

This is typically what a reverse proxy does.

Load balancing can happen at a lot of different places in our system.

We may have another load balancer in between our servers and databases.

When we are dealing with a website, we may even have load balancing at the DNS layer.

DNS query is made to get the IP address of the website we are looking for.

DNS Round robin.

DNS round robin is a kind of load balancing that happens at the DNS layer where a single domain name gets multiple IP addresses and when a DNS query is made to get the IP address of that domain name, the multiple IP addresses that are associated with that domain name are going to be returned in a load balanced way.

“**dig**” command takes in a domain and returns IP address of that domain.

If we do “dig google.com” from 2 different terminals, we get 2 different IP addresses for google.com.

If we curl both of these IP addresses, we get google.com for both of them.

This is great example of load balancing at play where basically the same domain name “google.com” is associated with different IP addresses.

The 2 terminals acting as 2 different clients here got served 2 different IP addresses.

**How does the load balancer actually distributes the load/traffic to the servers**?

Does it follow an algorithm or does it randomly ?

There are different types of load balancers. Some load balancers are software load balancers and others are hardware load balancers.

Hardware load balancers are physical machines that are dedicated to load balancing.

We can kind of do more with software load balancers. We have more power over customization and scaling.

With hardware load balancers, we are limited to the hardware we are given. And hardware is often expensive.

We will be looking at software load balancers and techniques that software load balancers use to distribute traffic to servers.

How does a load balancer even know that it has servers to distribute traffic to ?

If a new server gets added, how does it know to also redirect the traffic to this new server ?

People in charge of the system can configure the load balancer and the servers to know about each other.

When we add a new server or remove an old server, it registers or de-registers itself with the load balancer so that the load balancer knows about its existence.

As for how load balancers actually select servers to send traffic to, there are a lot of different ways that they can do that. The first one is pure random redirection.

We can configure our load balancer such that it will redirect traffic to all of the servers following a purely random order.

This might be fine but it could cause problems. It could be possible that one server would by chance get overloaded.

Another way that load balancers often select servers is following a round robin approach.

Round robin is a method that goes through all of our servers in one order.

We sort of guarantee that we will be evenly distributing traffic across our servers.

Another round robin strategy is “Weighted Round Robin” where we place weights on specific servers.

The order of redirecting requests will be similar.

The load balancer can redirect a couple more requests (a couple more traffic) when it is at some server.

And then it redirects to other servers.

We want to do that if one of the server is more powerful than the others.

Weighted round robin server selection strategy.

Would place more weight on a particular server.

Another way for a load balancer to select servers is based on performance or based on load.

Load balancer performs health checks on the servers to know how much traffic a server is handling at any given time, how long a server is taking to respond to traffic, how many resources a server is using and based on that it will redirect traffic accordingly.

If the load balancer sees that a particular server is not handling requests well, maybe it is getting overloaded somehow, it will not redirect requests to this server.

If suppose the load balancer notices that some other server is handling requests really well and is not being overloaded at all whereas the other servers are starting to get a lot of load, then it will redirect a lot of traffic to that server.

This could be a good way for a load balancer to distribute traffic depending on our use case.

IP based server selection strategy

When load balancer gets requests from clients, it is going to hash the IP address of the clients and depending on the value of the hash, it will redirect the traffic accordingly.

IP based load balancing can be really useful if we have got ‘caching’ going on in our servers.

If in our system we are caching results of our requests in our servers, it would be really helpful to have a request from a specific client always be redirected to the server in which the response of that particular client’s request has been cached.

This can be accomplished through IP based load balancing.

IP based load balancing can help to maximize cache hits.

Path based server selection strategy:

With this strategy, a load balancer distributes requests to servers according to the path of the requests.

This strategy is used on AlgoExpert. We (algoexpert) distribute some requests to servers according to their path.

All requests related to running code on AlgoExpert are going to be redirected to a specific server or a specific set of servers or requests related to payments on AlgoExpert, meaning purchasing the product are going to be redirected to another server or to another set of servers.

If we ever want to deploy a big change to a service, suppose we want to deploy a big change to Code Execution engine, this deployment will only affect the servers handling the code execution engine requests and all requests that are related to payments on algo expert would get routed to their own specific set of servers.

If code execution servers start dying for whatever reason, all other services will remain unaffected.

We have to pick server selection strategy according to use case.

When we are designing a system, it might be helpful to have multiple load balancers that use different server selection strategy.

We can have multiple load balancers in different parts of our system and which we will likely have.

And we will likely have multiple load balancers at the same part of the system.

What happens if a load balancer starts to get overloaded ?

One of the main problems that we were trying to solve was to prevent a single server from getting overloaded with requests.

What happens if our single load balancer gets overloaded with requests ?

In this case, we can have a 2nd or even a 3rd load balancer.

These load balancers can communicate with each other and depending on that, reroute traffic accordingly.