From the vault of my engineering newsletter
"Systems That Scale"

Saurav Prateek's

Foundational concepts in System Design Part 2

Explaining the foundational concepts involved in **System Design**

Table of Contents

Chapter 1

Consistent Hashing

Let's discuss **Consistent Hashing**. It has been widely used to achieve an even load distribution across the distributed servers. In this article we will understand why we use Consistent Hashing, What problem does this solve and at the end discuss the **Skewed Load Problem** in Consistent Hashing.

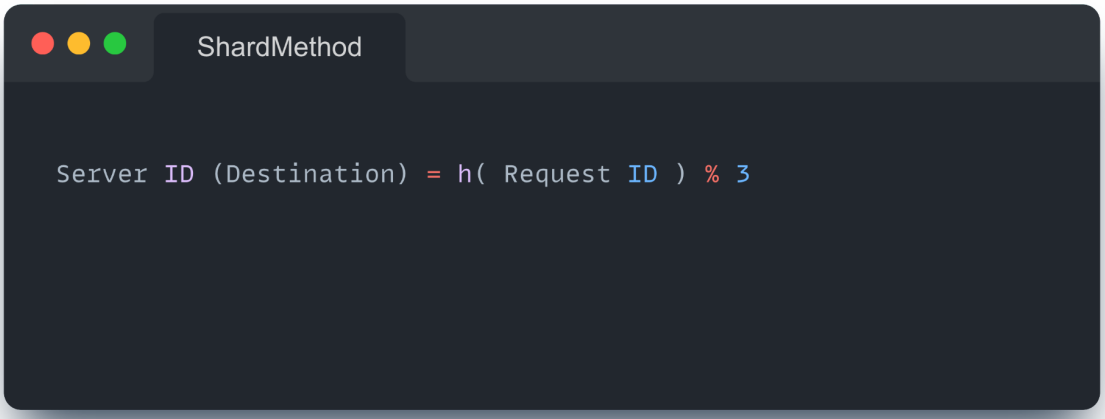
Introduction

Consistent Hashing is a concept extensively used in Load Balancing. In a typical Load Balancer we use hashing in order to map the requests to their corresponding Servers. Here when we had to add or remove any Server then the Hash Values of all the earlier requests got modified which caused our cache to become obsolete. In order to avoid this problem we use the concept of **Consistent Hashing**.

Using consistent hashing in our system we can avoid the change in the hash function of all the earlier Requests whenever any new server is added or removed. It allows only a small amount of requests to change hence making our cache very much in use.

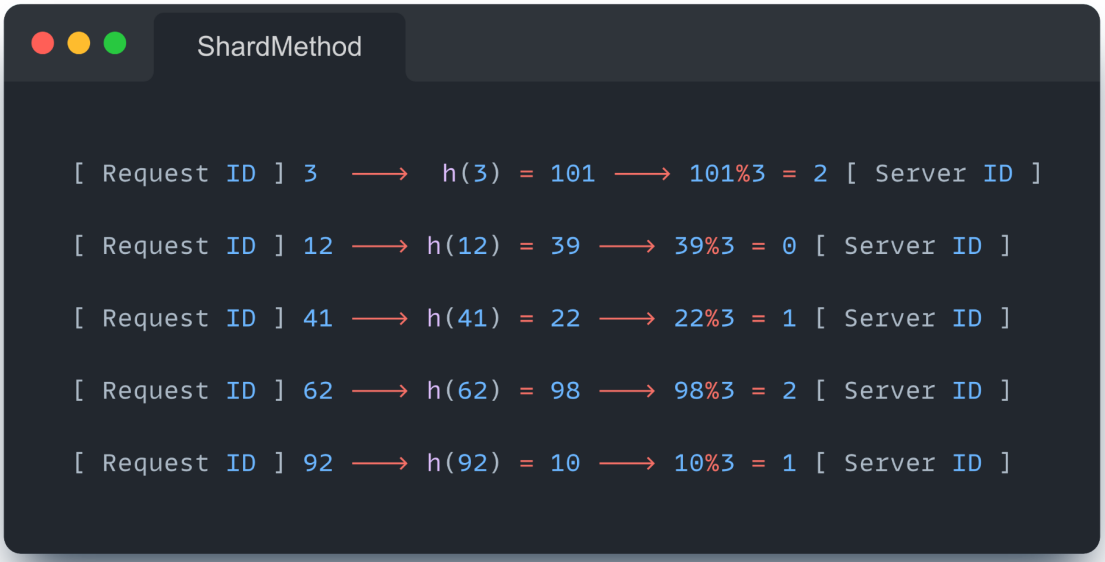
Understanding the Problem

Suppose you have a Load Balancer which distributes your requests among three servers : S1, S2 and S3. We have used a hash function to direct the requests to one of the servers in the system. The entire method used looks like this:



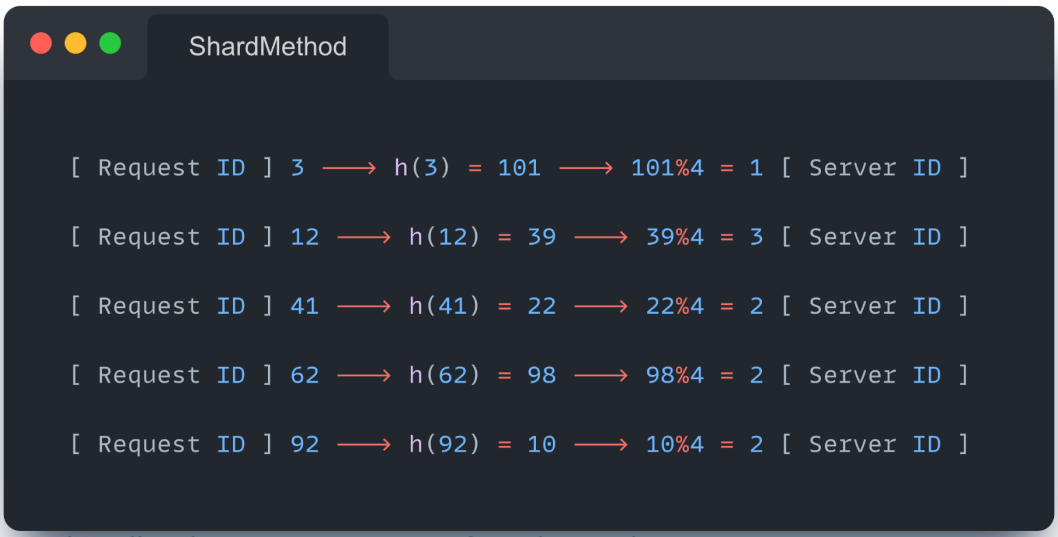
Here we can clearly see that the request id of the incoming request is hashed by a hash function (**h**) and then we did a modulo of the result with the total number of servers which is **3** in this case.

Suppose **5** Requests come initially then they will be directed in the following order:



As you can see, by using the earlier method we directed our requests to **3** Servers. Now what happens when an extra server is added to the System. Now we have **4** Servers with us and this will have a huge impact on the address of the destination servers of the earlier requests.

When the same **5** requests reach the System, it is now directed in the following manner.



We can clearly observe that the destinations of almost all the requests have been changed. Hence by adding or removing even one server will cause the requests to be directed to a completely different server. This can have huge drawbacks. Suppose if a server stored some critical data in cache about a certain request in the hope of re-using it when that request visits again. Now the entire data in the cache is obsolete.

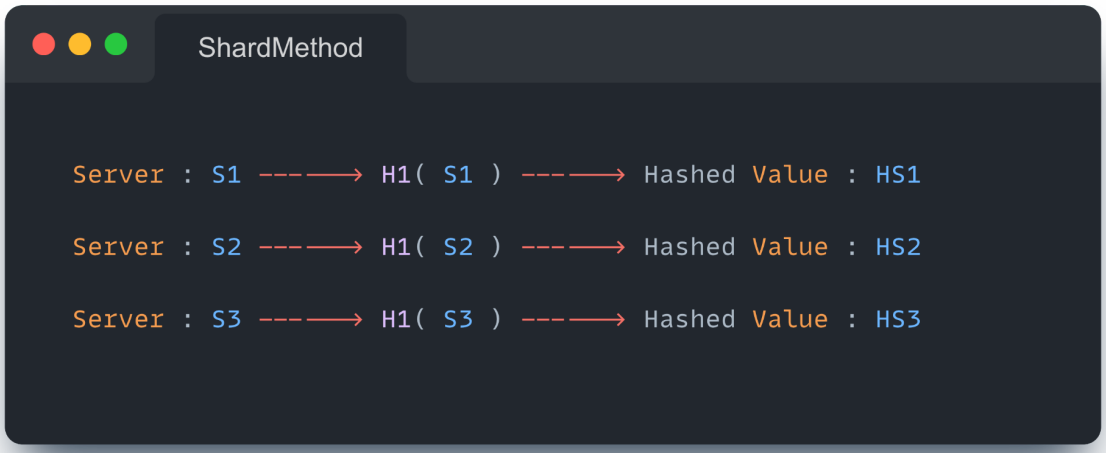
So our current method of Hashing is not quite reliable in order to direct the requests to our multiple servers. We need something which could cause a very less amount of change in the destination of the requests when a new server is added or removed. Here, Consistent Hashing comes into play. Let's see how it solves our current problem.

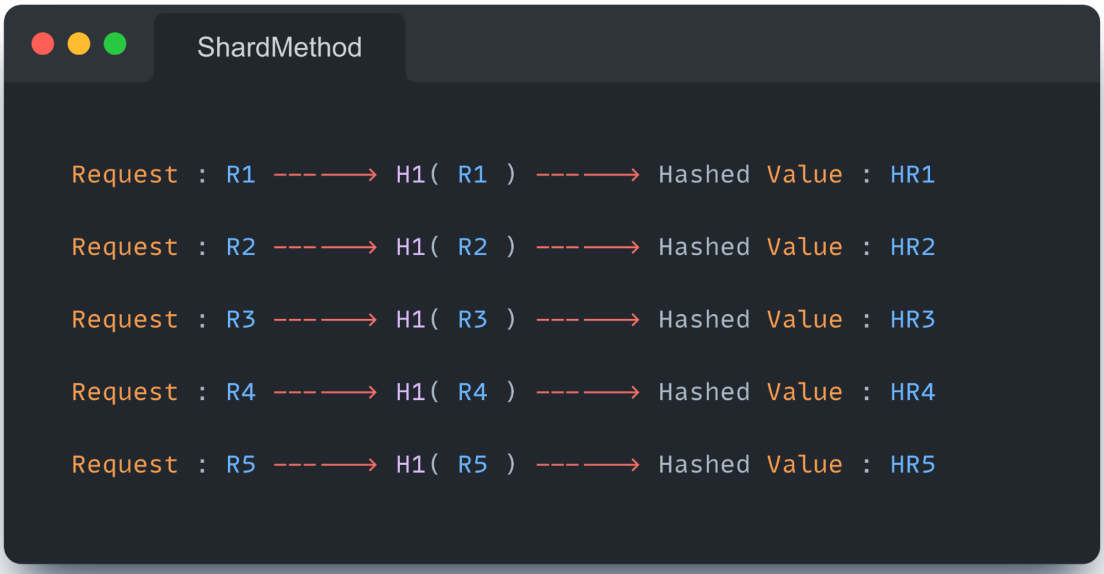
Discussing Consistent Hashing

It is a method which is independent of the number of servers present in the System. It hashes all the Servers to a hash ID which is plotted on a circular ring. This ring-like structure allows a very minimal change in the requests when a server is added or removed.

Suppose our Hash Function returns a value in the range of **0** to **N**. The ring starts from **0** and ends at **N**. Now each Server present is hashed using this function and is plotted over the ring. Suppose we have three servers **S1**, **S2** and **S3** and have **5** requests coming to the system.

We used our hash function say **H1** and hashed every server and request to get the values like this:

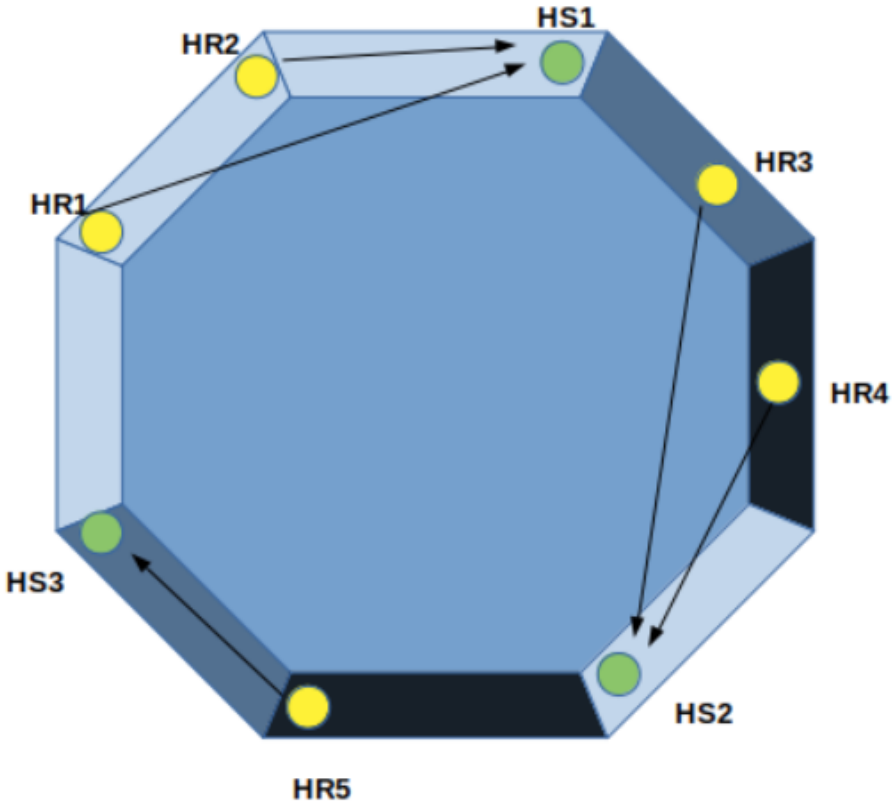




Now they can be plotted on the circular pie as described below:

7 - Saurav's handbook on System Design foundational concepts Part 2

Hashed Servers : **HS1, HS2. HS3**
Hashed Requests : **HR1, HR2, HR3, HR4, HR5**



Here every Request is served by the Server which is adjacent to it when we move in a clockwise manner. Hence Requests **R1** and **R2** will be served by Server **S1**, Requests **R3** and **R4** will be served by Server **S2** and Request **R5** will be served by Server **S3**.

Suppose Server **S1** got shut down due to some issue. Requests **R1** and **R2** which were earlier served by **S1** will now be served by server **S2** leaving all the other requests completely unchanged. This caused only those requests to change which were earlier served by the obsolete server. Rest of the requests were completely unaffected by the change in the number of servers. This is one of the major advantages of using Consistent Hashing in a system.

Skewed Load in Consistent Hashing

Earlier when we removed Server **S1** from the system Requests **R1** and **R2** were served by Server **S2**. Now requests **R1**, **R2**, **R3** and **R4** are served by Server **S2**. That means **4** out of **5** requests are served by a single server. The entire load is skewed on a single server and this can cause that System to wear out or shut down.

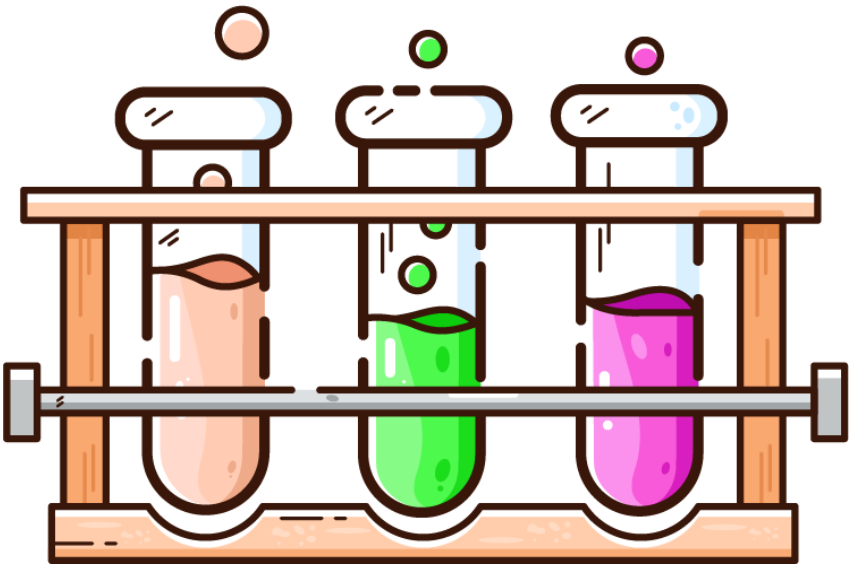
What can be done to avoid this problem here? Instead of using a single hash function we can possibly use multiple **Hash Functions** to hash the servers. Suppose we used K hash functions. Now each server will have **K** points on the circular ring and then it will be less likely to have a skewed load over a single server.

We can select the value of **K** accordingly in order to reduce the chance of having skewed load to none. Possibly **log(N)** where **N** is the Number of available Servers.

Chapter 2

Microservices and Monolithic Architecture

Let's talk about **Monolithic** and **Microservice** architectures, their pitfalls and advantages. The edition covers the concept of **Technology Heterogeneity** in the Microservice architectures and achieving **Resilience Engineering**.



Introduction

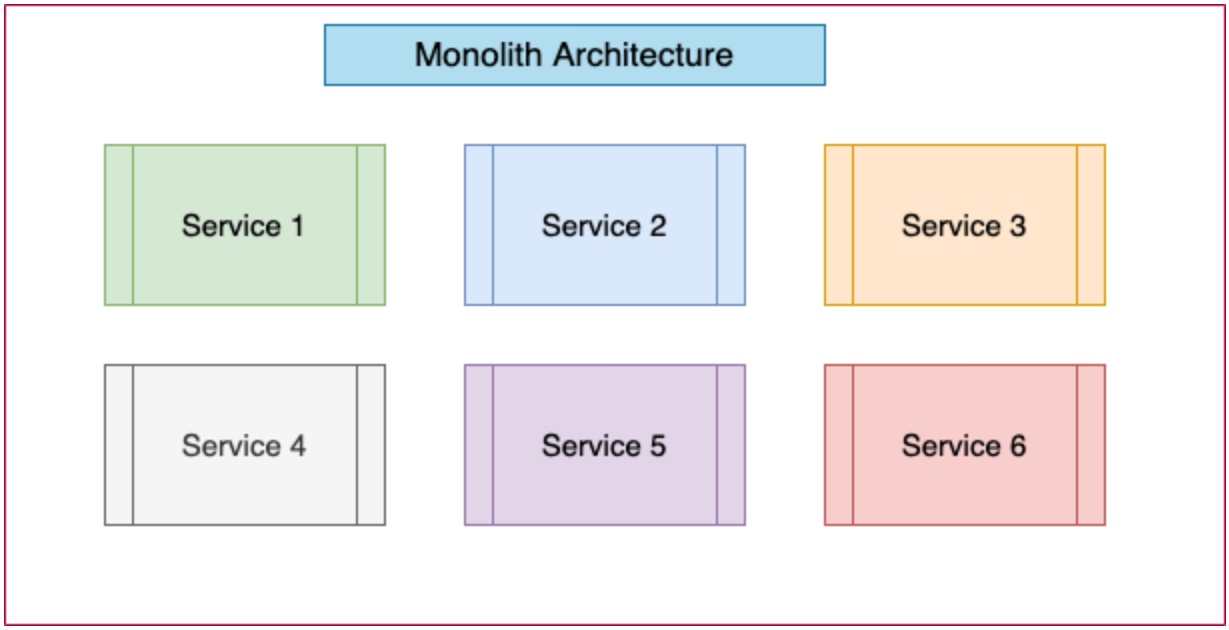
Systems are becoming large and complex with time. Now a single system or an application is responsible for performing multiple functions, providing users with multiple features and services. Such a large system can get complex sometimes as they have multiple components functioning altogether. Hence, writing modular code at the backend will not be enough to ensure a structured design. Apart from that, the system should also be structured at a higher level. There are multiple ways to keep your system structured so that it can be easy to maintain and understand.

A System having multiple components can keep all their components at a single machine or at a single place and such an architecture is known as a **Monolithic Architecture**. While there can be other cases where systems may store their different components individually at different systems all of them connected through network calls, such an architecture is known as a **Microservice Architecture**. We will be discussing both the architectures in detail, looking into their advantages and disadvantages. Let's take an example to discuss both the architectures in order to get a crystal clear understanding of these concepts.

Suppose we are designing an E-commerce Application having multiple components. Let's see what are some expected characteristics when we design it through a monolithic architecture and via microservices.

Monolithic Architecture

A system implementing Monolithic Architecture stores all of its components at a single place or tightly coupled and highly interdependent on each other. Suppose we are designing an E-commerce Application which has multiple components like : Product Catalogue, User Profile, User Cart and many more components.



Being a Monolithic Architecture the system stores all its components at a single machine. Such a system can have following characteristics:

- **Fast** : A system following a monolithic architecture can be comparatively fast as no network call or Remote Procedure Call (**RPC**) is required between multiple components. As the components are located at a single machine, hence a simple function call can get the job done. And these function calls are much faster than the network calls, hence the entire system can be fast.

- **Large and Complex Systems**: As all the components of the System are stored at a single place or a single machine, hence the system can get large. And if there are large components in a system then the entire system can become complex and hard to understand. This can cause several maintenance issues in future.

- **Hard to Scale**: Systems having a monolithic architecture can be hard or costlier to scale. They can take up unnecessary resources which can be saved by adopting a **Microservice** architecture. Earlier we had an E-commerce application with multiple components. Suppose there is a sale going on and this has a huge load over the **Product Catalogue** section. A large number of

users are trying to access the Catalogue. So you need to scale that component so that it can respond to the increasing load. You will be increasing the number of servers according to the increasing load. But in this architecture this will be a costly approach as all the components are stored in a single system and increasing the number of servers will indeed scale all the components present in the System. This is one of the major drawbacks with Monolithic Architecture. The component which is not facing a spike in the load is also scaled along unnecessarily.

- **Hefty Deployments**: Suppose you made a small change in the code base of the User Profile component of the E-commerce application. At the time of deployment you will need to deploy the entire system which can be costly and time taking as well. This is also a major drawback as all the components reside in a single machine.

- **Hard to Understand**: Suppose a new developer joins your team and is assigned to work on one of the components of your system. As your system implements a monolithic architecture, hence all the components are tightly coupled and that new developer has to go through the entire system in order to get a clear understanding of a single component. This makes a monolithic architecture hard and time taking to understand. The system may appear to be complex to a new individual.

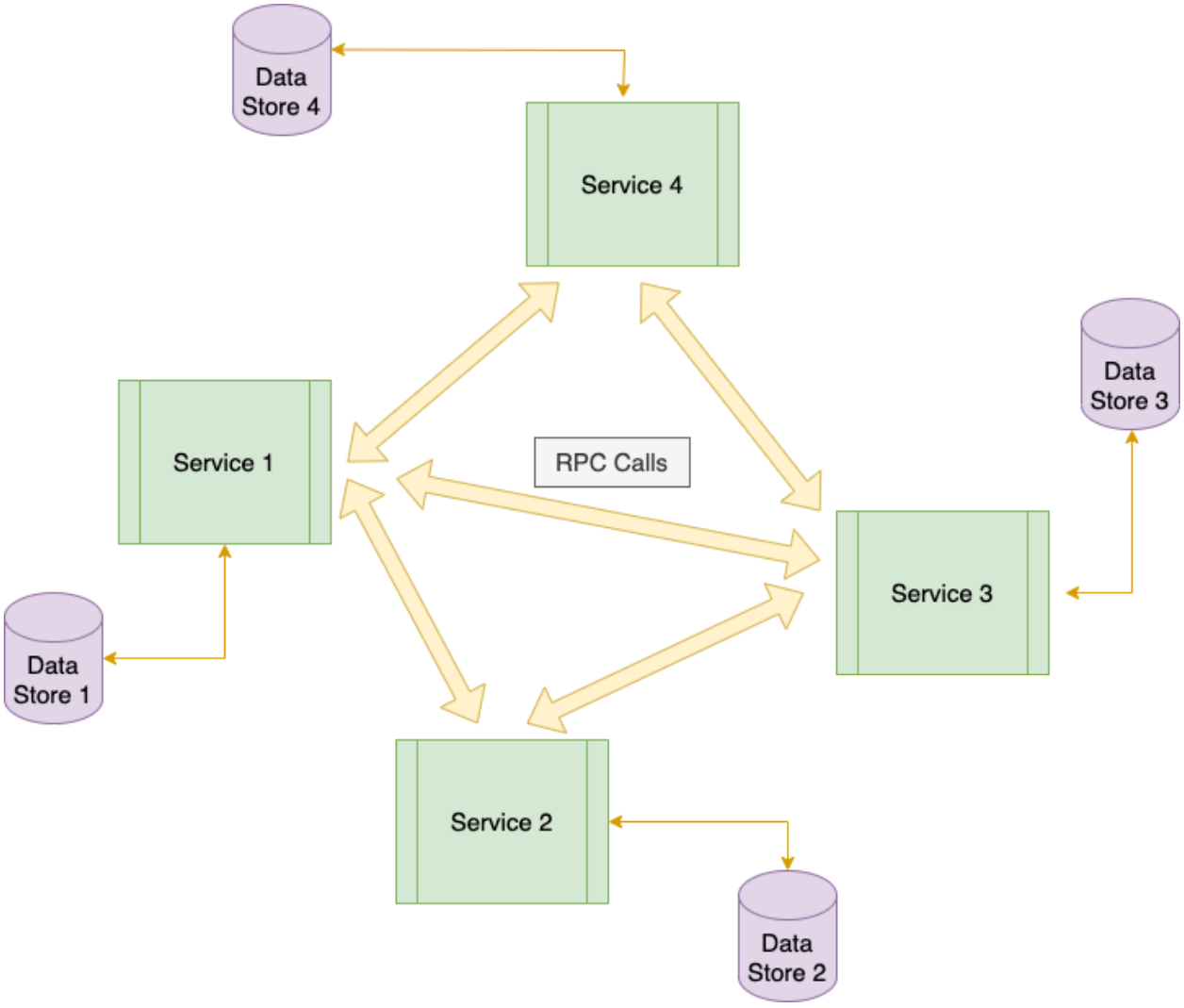
- **Complex Testing process**: A monolithic architecture can be complex to test. Suppose you have performed a small change in one of the components and have deployed the system. Then you will be needed to test the entire system as there is a chance of having a bug introduced anywhere in the system as all the components are tightly coupled and highly interdependent.

Microservice Architecture

A system implementing **Microservice Architecture** has multiple components known as microservice which are loosely coupled and functioning independently. Suppose we have an E-commerce Application having multiple components as

discussed earlier. In a microservice architecture all these components are deployed on separate machines or servers which communicate remotely with each other through a Network Call or a **Remote Procedure Call** (RPC). These components function individually and solely as a separate entity. They come up together to form an entire system.

Basically a large system is broken down into multiple components which are then deployed on distinct machines and are set to operate independently from each other. All these components function as an independent and complete system.



A **Microservice Architecture** can have following characteristics:

- **Network Calls required**: In a microservice architecture all the components are stored in different machines and hence requires a medium to communicate. These components communicate with each other through a set of **APIs** via **RPCs** or **Network Calls**. An individual component receives the request from the network call, processes it and returns the response again through a network call.

- **Easy to Scale**: A microservice architecture can be easy to scale. As all the components function individually hence it is easier to scale a particular component according to the requirements. In an E-commerce application when there is a sudden spike in the load on the Catalogue component during the time of sale then we can easily scale that particular component only. As these components are stored in separate machines, we can increase the count of the machines which hold that particular component according to the increasing load. The rest of the components which didn't receive any load hike are kept untouched. Hence using the resources efficiently.

- **Easy to Test**: A system implementing a microservice architecture can be easy to test. Suppose we need to make a change in the User Profile component of the E-commerce application. In that case we can make the change in that system and test it independently for various test cases and once we are satisfied with the testing we can deploy the changes. We are not required to test the entire system in case of a microservice architecture. We only need to test that particular component or microservice in which the change has been made.

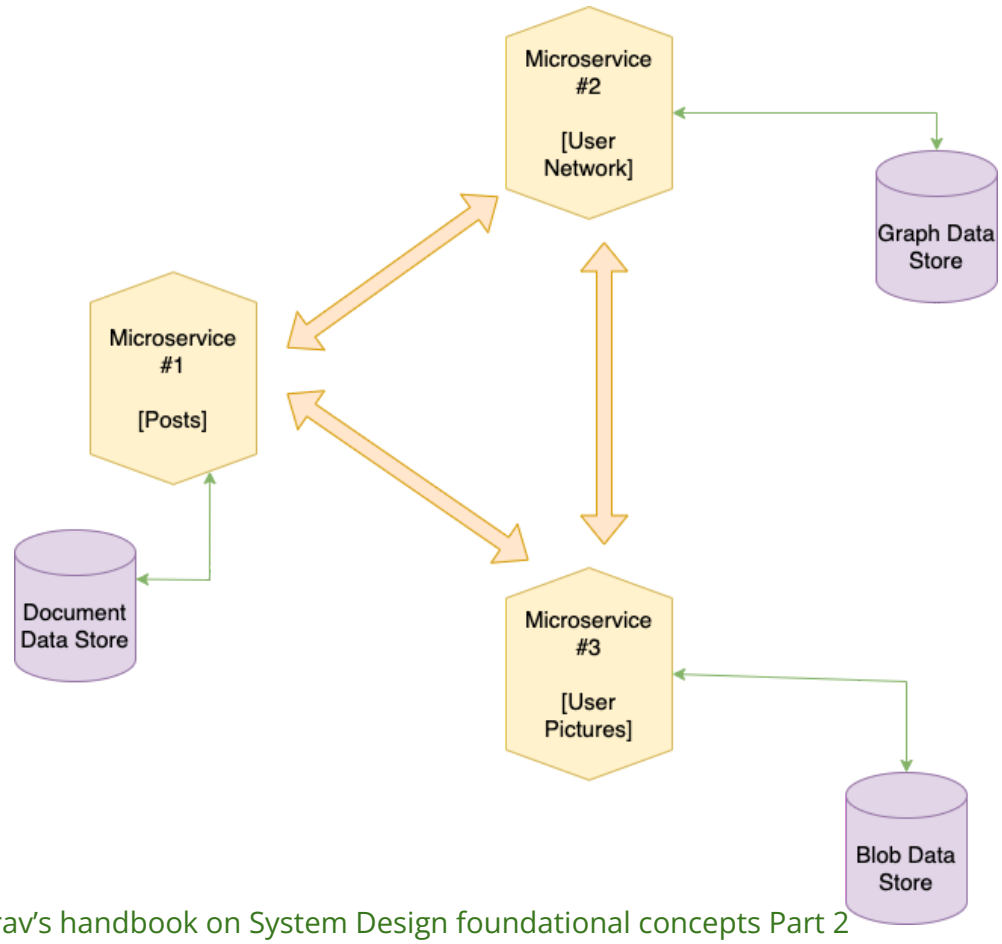
- **Easy to Understand**: A microservice architecture can be easy to understand. As all the components function independently and are not dependent on others in any way. Hence when a new developer comes up and has to work on a particular component then in that case he/she is not required to go through the entire system. They can simply understand the working of that particular component or microservice which they have been assigned to

work on. As these components are loosely coupled, allow the developers to work independently on the services.

Technology Hierarchy

In a **microservice architecture**, we have multiple services working independently. In these architectures, we can use different technologies in different services. This also helps to pick a more suitable and correct tech stack for each service.

Along with different tech stacks, we can also change the way we store the data for different services. We can store data according to our data needs in different microservices. For Example: For a social network we might store the users' interaction in a Graph-oriented database to reflect the highly interconnected nature of a social graph. But perhaps for the posts that users upload over the service, we can store them in a simple Document-oriented data store. This gives rise to a **Heterogeneous Architecture**.



With microservice architecture, we can also adopt new technologies more quickly. One of the biggest barriers in trying out new technologies in a monolithic architecture is the high risk associated with it. With a Monolithic Architecture, if we want to try a new Programming Language, Database or a Framework this will impact a huge percentage of the system.

But with the systems having a Microservice Architecture, we have a lot of independent services where we can try out a new piece of technology and may also experiment with it. We can pick a service with perhaps the lowest risk and use the technology there knowing that failing to adopt will have the lowest negative impact on the system.

Resilience Engineering

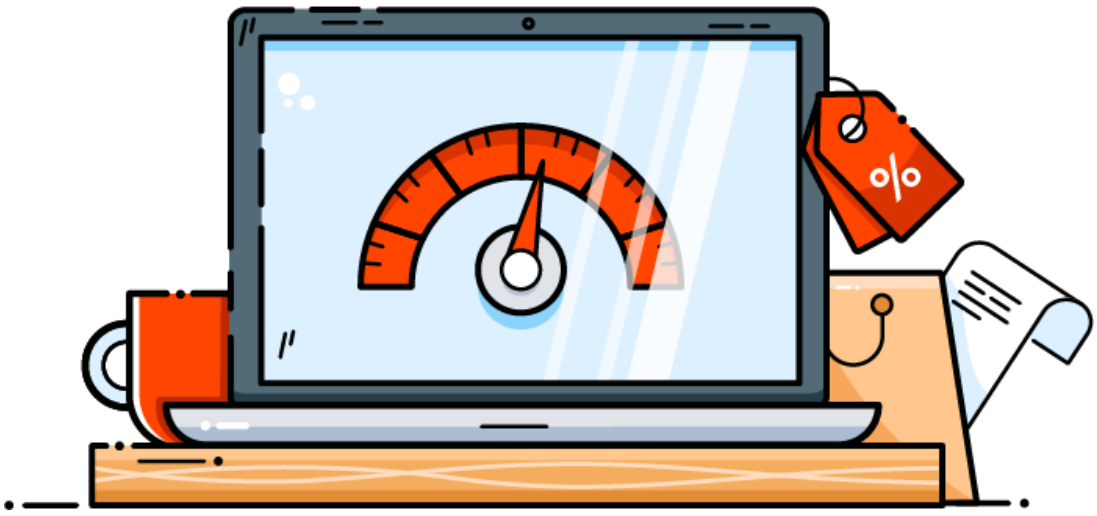
In a **Microservice Architecture**, if one component/service fails then it doesn't impact the entire system. Rest of the other services can still function well while that one service is down. But in a **Monolithic Architecture**, if the service fails then everything may stop working. Hence generally with a Monolithic Architecture, we run services on multiple machines to reduce the chance of a **Single Point** of **Failure**.

But in order to ensure that our Microservice Architecture embraces the improved resilience, we need to understand the new source of failure that Distributed systems have to frequently deal with. Yes, these are **Network Failures**. Since in a microservice architecture, we have multiple services working independently, these services generally communicate with each other through **RPC** calls. These RPC/Network calls might fail sometimes and in that case we should know how to handle this and what impact (if any) it should have on the end user of our services.

Chapter 3

Cracking the System Design interview

Preparing for the System Design interviews can be overwhelming sometimes. This chapter involves tips and resources which might help in clearing your **System Design** interview rounds.



Introduction

If you are a college graduate preparing for the Software Development roles of a Product based company then System Design rounds might not be a part of your Interviews. But one or two years down the lane when you are looking for switching to a different company then this might not be the case. At that time there can be 1 to 2 rounds of Interviews focussed around System Design scheduled for you. How should we tackle that? What are the things to avoid? What are the things to remember? What concepts should we know? Do we need to know everything?

If you are planning to prepare for these rounds, you can get tons of content on the Internet which is free. And sometimes we can get overwhelmed by this huge amount of content present around this topic. One problem can be solved in multiple ways here. Say, you pick a question on how to design a chat application like Whatsapp, for this single problem you can find multiple video explanations over the internet and there can be a chance that all of these explanations may look very different from each other. Some may use one technology while others may use a different technology. Some might follow one architecture while others might follow a different one.

So what is the scope of this chapter? This chapter will focus on some best practices you can follow to prepare for these rounds and some mistakes you can avoid in these interviews. I will also share some resources at the end which you can follow to prepare for these rounds.

Let's start with some points to remember while attending a System Design Interview.

Communicate with your Interviewer

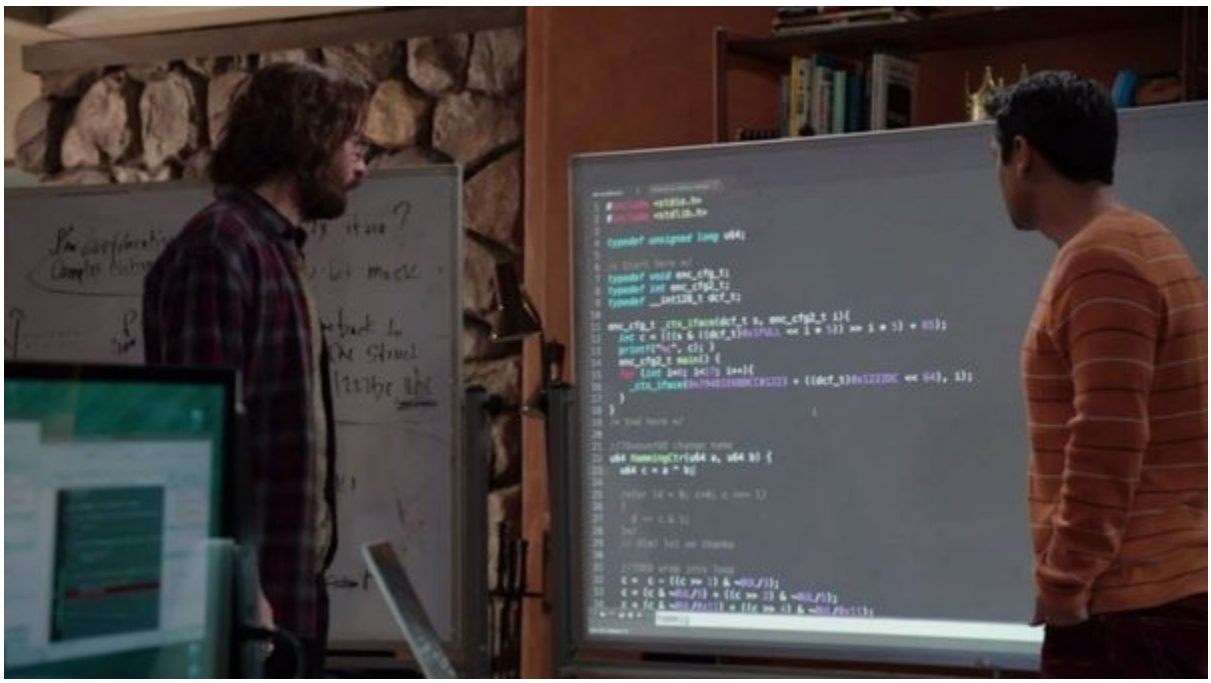
One of the major things to keep in mind while you are giving a System Design interview is to keep the Interviewer in loop. Suppose when you are given a Problem statement then don't just start framing your entire solution and go on explaining it. Instead ask your interviewer about what he/she demands. What load should you

expect your system to handle? What features should your system provide? Obviously it's not possible to design a full fledged system in a span **45** to **60** minute interview. You need to simplify a few things. You should ask your interviewer about what features you are planning to include. And will it be okay to exclude the others? This will not only simplify things up but since you're already asking your interviewer about this, you won't be missing out any essential feature which your interviewer would have expected you to include in the first place.

"See, communication is the key"

While designing your system, you can always let your interviewer know about the decisions you are taking and why. Like, whether you're going with a SQL or a NoSQL database, Will you be implementing a Cache and if Yes, then Why. Letting your interviewer know about these decisions and the reason behind those at the time you take them can keep things clear as you move ahead with the Solution.

Consider this round more as a discussion and less as an Interview



System Design interviews can be overwhelming sometimes and it's completely okay to be nervous. But one thing which you can do to reduce the stress of these interviews is instead of thinking of them as an Interview, think of them as a technical discussion which you will be having with your interviewer for a span of 45 to 60 minutes. Adopting this mindset can reduce stress and nervousness a lot.

Break your problems down to multiple solvable sub-problems

Since you have been asked to design a complete System your solution will be huge. Don't get overwhelmed thinking about how to design such a large system in this short span of time. Instead, try to divide the Problem into multiple small sub problems. Suppose you have been asked to design a particular System, then try to come up with all the features you are planning initially to include in your System. Enlist all of them on a doc or a whiteboard available. Now start implementing those features one by one. Take one feature at a time and give your best 5 to 10 minutes implementing that, don't think about the rest of the unimplemented features.

Say you are given a problem to Design a Chat application like Whatsapp. You can list multiple features you could include such as: End to End messaging in real time, Online/Offline status feature, Delivered/Read message status feature, Encrypting your messages while sending them and much more. Once you have listed all the essential features then you can move ahead and implement them one by one.

Be confident and have some points to back your solution

Having some points to back up your solution or the technology you are using while designing your System is always a green flag in these Interviews. Having a reasonable explanation behind the services you are using or the technologies you have adopted in your solution will reflect that you have a deep understanding of these technologies and services. It reflects that you have a clear knowledge of when one should use and when one should avoid a particular design or tech. This also

shows that you have a clear understanding of the Problem you have been asked and what it demands.

This knowledge of when you should use or avoid a particular design or technology comes with time and experience. As you keep learning about the concepts and tech, you will get to know their plus points and drawbacks. Eventually you will know in which scenario you should pick which tech stack and design. You will know what will help your system to scale in the long run.

Perform estimates and calculations about your System

Performing calculations and Estimating things while designing your System can be a plus point. You can take multiple information from your interviewer beforehand and on the basis of that you can further calculate and estimate multiple things about your System.

If your interviewer informs you about the number of requests your system will be handling per hour or per day, then you can easily estimate multiple things like: How much data your system will be generating, How much servers you should have to distribute the load, Will you be needing a Cache service or not, What can be the throughput of your System, Number of Database calls or API calls you will be required to make and much more. Performing these calculations beforehand while designing your System gives you a nice idea about what you are actually dealing with.

You don’t need to know every technology

One of the major things which I struggle with myself as well is the fear of not knowing enough. And this is okay, it helps us to learn and explore more. But on the other hand don’t let this demotivate you in any way. You are not expected to know every technology out there. There are a ton of them. Instead focus on the core concepts which these technologies implement. And knowing one or two technologies or services following a particular concept or design is enough.

A few resources which you can look up to and it’s Free!

While I was preparing for my System Design interviews these resources helped me a lot. Especially during my System Design round at Google.

1. Gaurav Sen's System Design Playlist: This playlist covers most of the hot topics and includes some very interesting case studies as well.

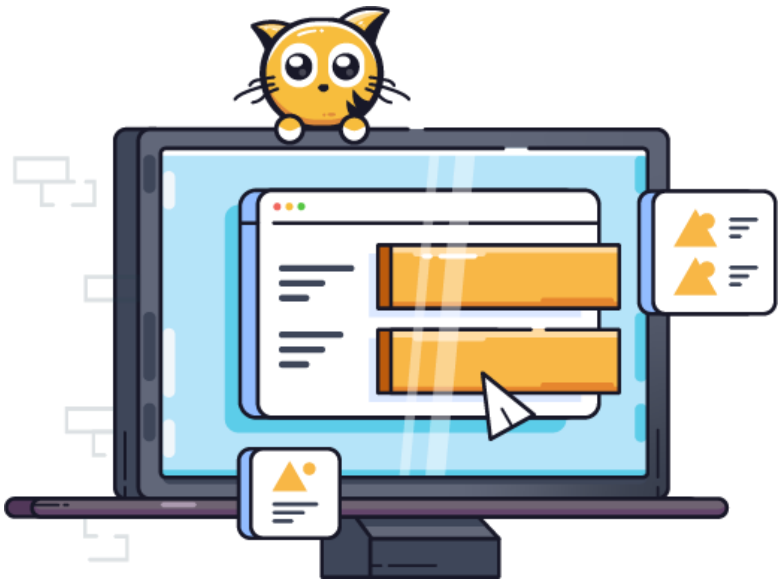
2. Distributed System Playlist (MIT OCW): This includes thorough content on Distributed Systems. It’s from MIT OpenCourseWare and the lectures are lengthy and a bit advanced. But it’s very detailed and thorough. A must watch!

3. Success In Tech : System Design Interview Questions Playlist: You can use this playlist for practising System Design problems. It has a decent collection of problems and the solutions are very well explained.

4. This is My Architecture by Amazon Web Services: It is a video series that showcases innovative architectural solutions on the AWS Cloud by their customers and partners.

5. Software Architecture Playlist by InfoQ: Contains a very informative collection of Software Architectures and Practices followed by the tech companies.

These were the few concepts which I always keep in mind while participating in a System Design interview or sometimes in a general interview as well.



Thank You! ❤️

Hope this handbook helped you in clearing out the foundational concepts of System Design.

Follow me on Linkedin
Do subscribe to my engineering newsletter “Systems That Scale”