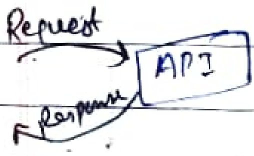
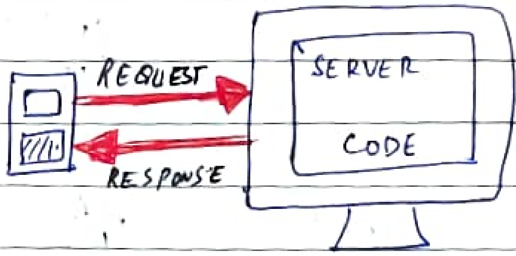
• अगर database में आपका signup nahi ho raha (Task 2) } controller → signup → await User.create kiya → as this not DB
signup.js का signup & signup.html का onsubmit = signup(event)
same hua change

1 | SYSTEM DESIGN BASICS : HORIZONTAL vs VERTICAL SCALING





Expose
INTERNET

SCALABILITY

1. Buy Bigger M/c - VERTICAL SCALING
2. Buy MORE M/c - HORIZONTAL SCALING

Horizontal Scaling	Vertical Scaling
[1] [2] [3] [4] [5]	Huge Box
1. Load Balancing Required	1. N/A
2. RESILIENT → if 1 m/c fails it may be transferred to other m/c	2. Single point of failure
3. Network calls (Remote procedure calls) → slow	3. Inter process communication → fast
4. Data Inconsistency → If there is transaction then we have to lock all the servers i.e all the DB which is impractical. we have some sort of loose transactional guarantee	4. Consistent → one systems where data reside
5. Scales well as USER increases	5. Hardware limit → we can not make m/c bigger & bigger what we want

we take Hybrid sol" is Horizontal Scaling where each m/c have big box.

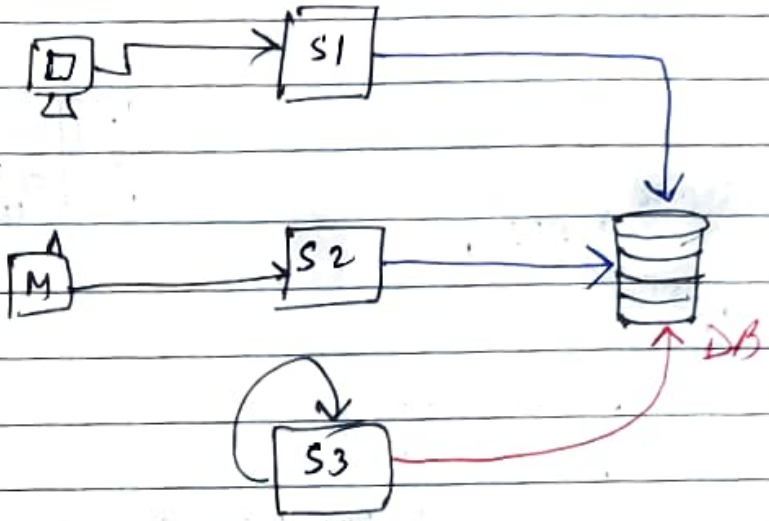
2 WHY DO DATABASE FAILS? ANTIPATTERNS TO AVOID

Frequent polling → load on DB / Long interval → Inefficient

Problems: polling interval

ANTI-PATTERN

DATABASES AS MESSAGE QUEUES



Part
- C → Create
- R → Read
- U → update
- D → Delete

"Using DB as message Queues"

"Using DB as message Queues" If,

↳ If we are using DB to send messages from one server to another that's considered as antipattern and we should avoid them

↳ DB is optimised for either reading or writing not both

↳ When we are going to do both operation at a time, it will have issues like locking, deadlock

↳ Some thing like CRUD model

↳ In this CRUD model we do Study that how much time do each of these operations take

↳ DB are rarely designed such that the read & write DB optimal Data structure. (Not possible to Design)

Here, interesting point is that what if our servers are talking frequently i.e set of message S1, S2 & S3 are sending to each other. So the DB will filled up with lot of entries

So, we need to store all the data or we can start clearing it. If we are nt going to delete the data then we need to update it & mark it as completed which is an expensive operation.

Or we have to delete it manually or using cron jobs. In both cases of deletion is going to be using script or using complicated logic

→ So 3rd problem comes for large system is scalability.

S4, S5 → eg: in this way DB cannot handle so many read operations. So if we are adding new DB which handle S4 & S5, even how will S4 talk to S2. For this we have to make complicated system which will act as broker b/w 2 DB's. There is no need to do all this when we have system design concept MESSAGE QUEUES

- ANTIPATTERN here is that we are using DATABASES AS MESSAGE QUEUES. Instead we will use specialized message queues when we have very large system.

Message Queue will avoid all these problems

→ polling interval is not going to come into consideration because message queues push the message to other server. so instead

→ So instead of server asking, message queue is going to give the message.

→ They are optimised so there's not too many read

operations. So writing to it is easy while reading is really the message queues ~~operation~~ responsibility. So it's going to do it in its own free time

Scalability is also taken care bcoz if we need more message queues we can just add them

- Drawbacks of message Queues.
  - → when system is not very large bcoz if we don't have those many servers then the DB will be able to handle loads that we throw on it
  - → Cost of setting up the message queue along with training is large
  - → Maybe overkill for a small service

Summary

- ~~DB~~ DB are often used to store various types of information, but one case where it becomes an as a problem is when being used as a message broker.
- DB is rarely designed to deal with messaging features, & hence a poor substitute of a specialized message queue. when debugging a system, this pattern is considered an anti pattern.

Here are the possible drawbacks

- → polling interval have to be set correctly. Too long makes the system inefficient, Too short makes the

DB undergo heavy read load

→ Read & write operation heavy DB. Usually, they are good at one of the two

→ Manual delete procedures to be written to remove read messages

→ Scaling is difficult conceptually & physically.

3 What is microservice architecture & its advantage?

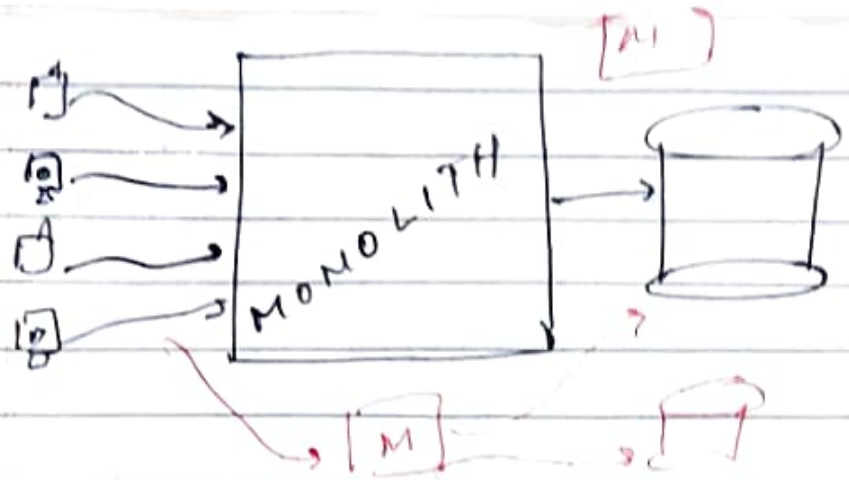
- we use Microservices instead of monolithic system. short answer is: Scalability.

- Advantage (Microservices architecture favorable)

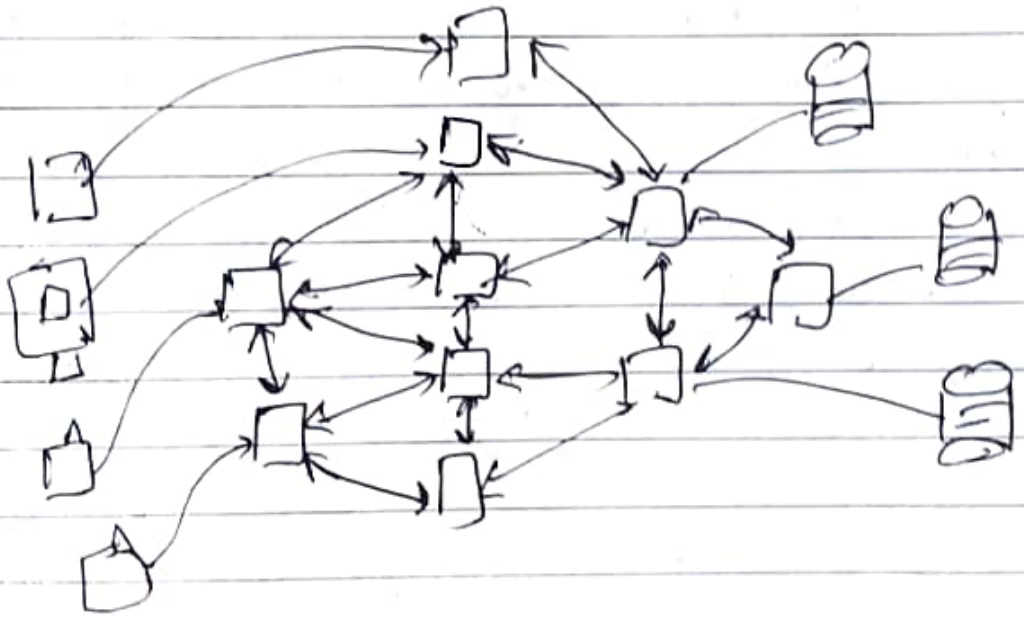
→ Microservice architecture is easy to design for complicated system
→ Allow new member to train for shorter periods & have less context before touching a system
→ Deployments are fluid & continous for each service.
→ Allow decoupling service logic on the basis of business responsibility
→ They are more available as a single service having a bug doesn't bring down the entire system. This is called single point of failure
→ Individual services can be written in diff languages
→ Developer teams can talk to each other through API sheets instead of working on the same repository, which requires the conflict resolution
→ New service can be tested easily & individually. Testing structure is close to unit testing compared to monolith.

- Disadvantage (Monolith architecture favorable)

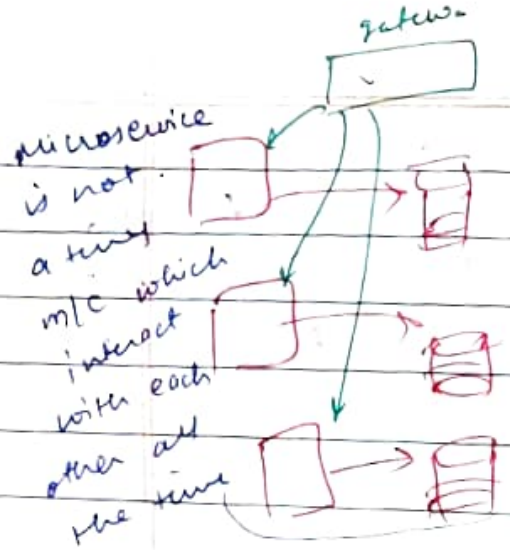
→ Technical/developer team is very small
→ Service is simple to think of as a whole.
→ Service requires very high efficiency, where network are avoided as much as possible
→ All developer must have context of all services



- Monolith doesn't need to be single in case of no. of m/c we are running bcoz there might be multiple m/c to the same monolith & these client can connect to them & they can connect to DB we can horizontally scale out even with monolith



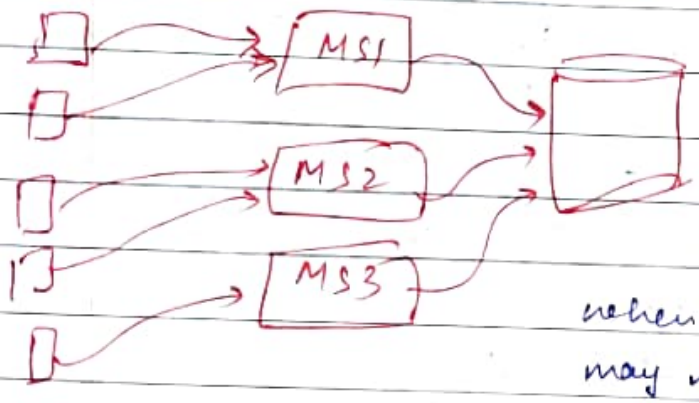
- Microservices is a single business unit, all data, all fn which are relevant to a service are put into one service. If we can seperate it with lot of concerns being seperated out then we probably want to seperate single service into pieces



There may be 3 microservices in our entire architecture depending on what our system is. & they usually talk to their own DB.

client may not be talking to microservices they might be talking to gateway. & this gateway might be talking to all the microservices

• ADVANTAGE OF MONOLITH



(1) Good for small team.
↳ when put under lot of load, monolith architecture scales out
when we have small teams we may not afford time & efforts to break them into small microservices. If our team is cohesive go for monolith architecture.

(2) Less complex
↳ lesser moving part around this architecture, we don't need to wonder that how we break into pieces & once we break into pieces how we maintain diff. servers at diff times. Deployment is easy as everything is same

(3) Less Duplication for every service we create
↳ we might have some code which are for setting up

(4) This is faster (Procedure call is faster)
↳ bcoz we are not making any call to network its not the RPC (Remote procedure call). Its actually in the same box. we just need to make the local call.

DISADVANTAGES

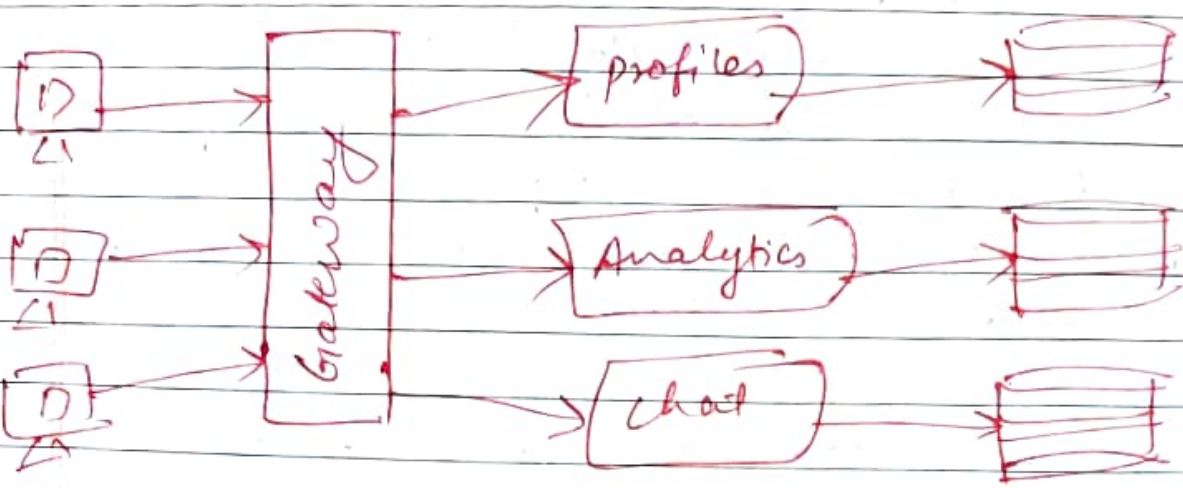
More Context Required

(1) If there are new members in team, they need to have lot of context on what they are developing To give them monolith which contains all the logic they have to go through and understands the whole system.

(2) Complicated Deployment
↳ bcoz any change in the code requires a new deployment. Here code is going to be deployed very frequently & it has to be monitored everytime that is being deployed

(3) Single point of failure
↳ There is too much responsibility on each server. If there is a mistake by of which server crashes all of these server will crash & whole system will collapse. ~~Instead if we have something called serving profiles~~

Microservices



(1) Scalability
↳ as we can look entire set of data as a set of services. Each concerns with only its data & interacting with each other, so its easier to design the systems in that way

(2) Easier for new team member
↳ whenever there is new developer coming in the team, we can assign them the task which concerns to the particular service, so they need to knows the content of 1 service for eg:- chat service instead of entire monolith

(3) Working in Parallel
↳ parallel development is easy bcoz there is lesser dependency for the chat developers on the analytics developer now bcoz they can develop at the same time in the monolith, maybe one fn is calling the other fn & its changing. so there is lot of tight coupling not just in code, but also in developer time

(4) Easier to Scale out

↳ There are lesser part which are redolent when we are actually deploying this service. If there is lot of load on chat server, we can easily scale that out by putting more m/c for that chat code. With monolith, it's more difficult to scale what is being used a lot & what is being used less. So we are more likely to put more server directly instead here we can have more streamlined approach for the problem.

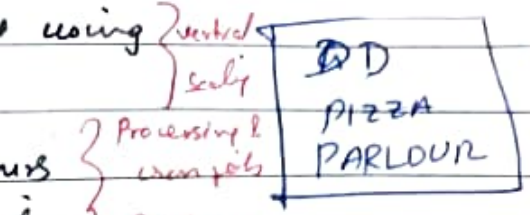
Disadvantage of Microservices

↳ Not easy to design so that are being broken into far more parts which are not required.

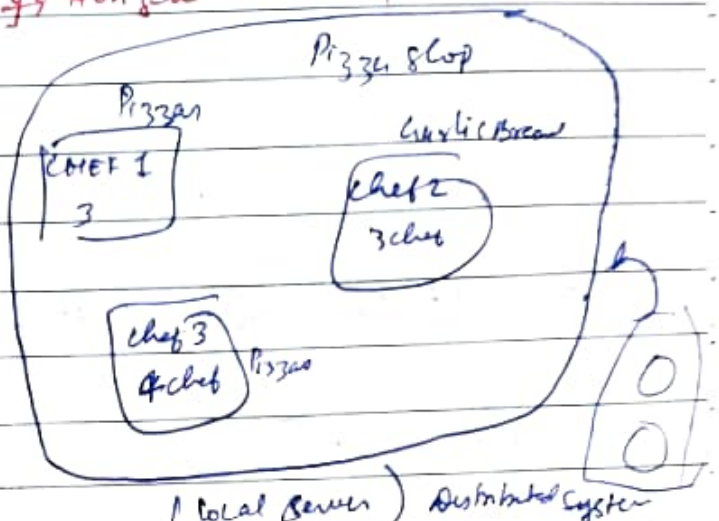
↳ Needs small architect to architect well for a microservice architecture

4 How to start with Distributed System

1. optimise process & increase throughout using same resource } Vertical Scaling
2. preparing before hand during non peak hours } Processing & cron jobs
3. keep backup & avoid single points of failure } Backup
4. Have more resources ie more m/c } Horizontal Scaling
5. Microservice architecture
6. Distributed System (Partitions)
7. Load Balancer
8. Decoupling
   - Separation of responsibilities
9. Logging & metrics calculation
10. Extensible

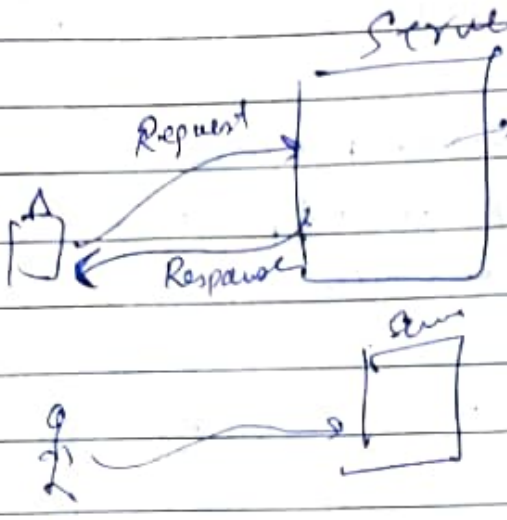


Socket + 1 backup slot



5 What is Load Balancing

2 WHAT IS LOAD BALANCING



we have N servers & we have to balance the load evenly on each server. concept of this balancing is known as load balancing

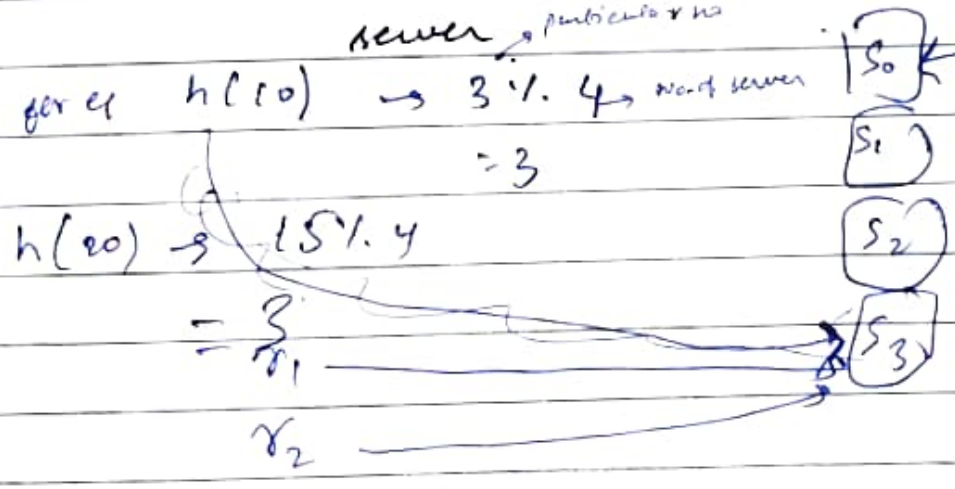
Concept of Consistent Hashing helps us to do that

- Request ID is generated from each server.

let we have Request ID → 0 to M-1

take this requestid. & hash it, we get particular no. & this no can be mapped to particular server → particular server no

$h(r_i) \longrightarrow [m_1 \cdot /. n]$



X request n load i.e X/n load & load factor to 1/n

fault tolerance means machine crashes &
request allocation means request to the server

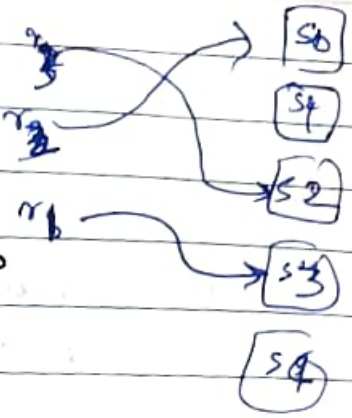
85

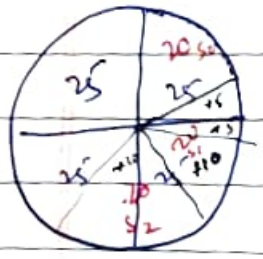
if we had 4 more Server

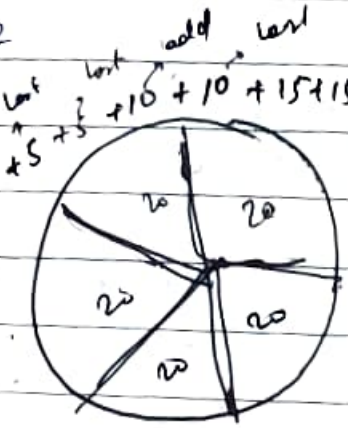
$r_1 \to h(\cdot) = 3\%.5 = 3$

$r_2 = h(15) = 15\%.5 = 0$

$r_3 = h(\cdot) = 12\%.5 = 2$

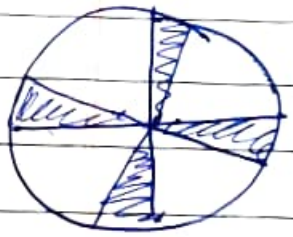






total change = 100 = M

2nd Case :- overall change should be minm



In new pie chart we will do minm change in all 4 such that adding them gives 20%.

In practice request ID is so rarely random, instead it encapsulates some user ID.

$h(r_1) \to$ This hash should give same result again & again

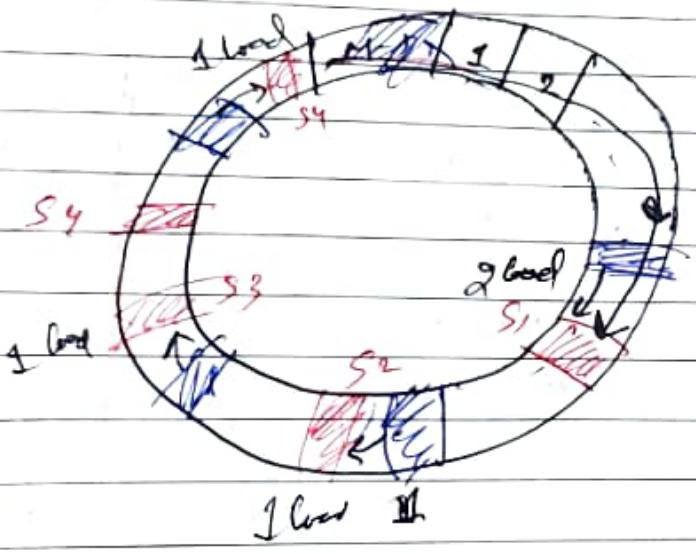
Depending on user ID we will send it to specific server once we send there we will store relevant information in form of cache but in previous one entire system changes & all useful information in form of cache is ~~used~~ useless as the no's what we were seeing before to server is completely changed. & we see huge change in range of servers we are seeing.

So old hashing don't work in this case & so we need more advance cases from where consistent hashing came

WHAT IS CONSISTENT HASHING & WHERE TO USE IT

- Problem is not the load balancing but adding or removing servers that we saw. which completely changes the local data that we have

Request ID → h(n)



It will always go in clockwise direction to see the nearest server to it

Instead of hosting in array we will do it in ring.

h(0) % M
h(1) % M
h(2) % M
h(3) % M
h(4) % M
h(5) % M

Load factor expected = $\frac{1}{N}$

On addition new server the change what we see is less.

If we lost S1 server then all load will go to S2.

Problem here is that theoratically load factor is 1/N but it is not uniformly distributed but it is skewed distributed.

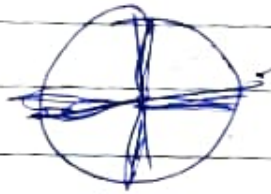
like here only we will see that half of the load is on S2 which is terrible.

what we can do ?

by make multiple hash fn. K hashes for that we have k points
& likelyhood of getting 1 server load more is much much lesser

if k=2 then 10%
k=3 — 15%

If we choose K value appropriately (eq :- log N), we have ~~change~~ almost eqn entirely removing skewed distribution chance.



→ add or remove so this increase EXACTLY change will be min as equally all server ☐ add ☐ remove hgn.

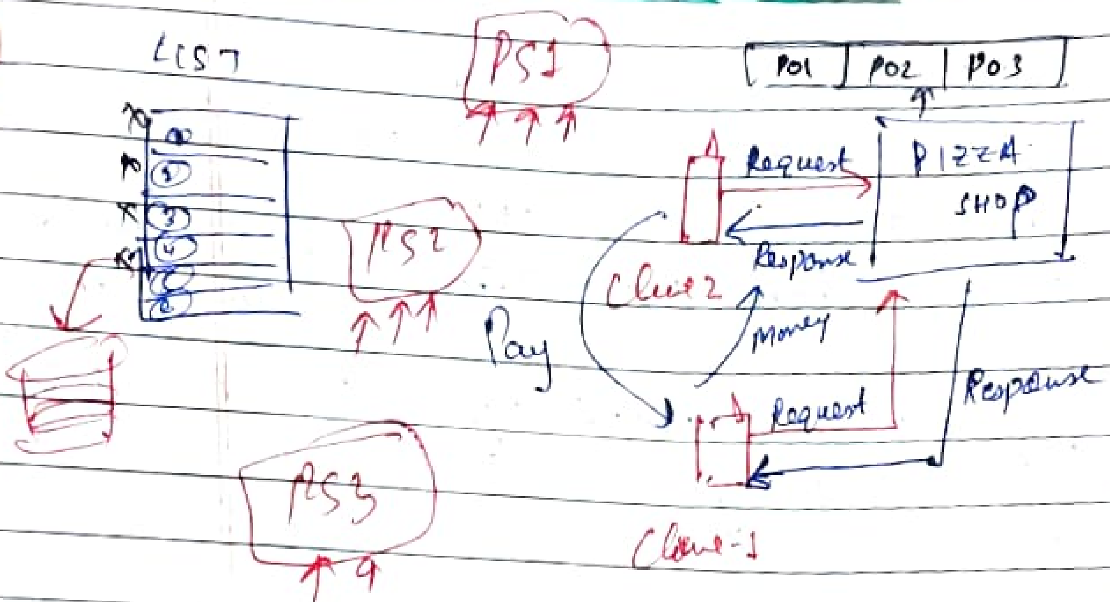
LOAD BALANCING is

Used in Distributed system extensively. used by Databases,

Consistent Hashing is something that gives flexibility & load balancing in very clear & efficient way. such that load is balanced & remain close to equal

• M.Q are widely used in asynchronous systems. Message processing is an asynchronous fashion allows the client to relieve itself from waiting for a task to complete & hence can do other jobs during that time. It also allows server to process its jobs in the order it wants to.

WHAT IS MESSAGE QUEUE AND WHERE IT IS USED



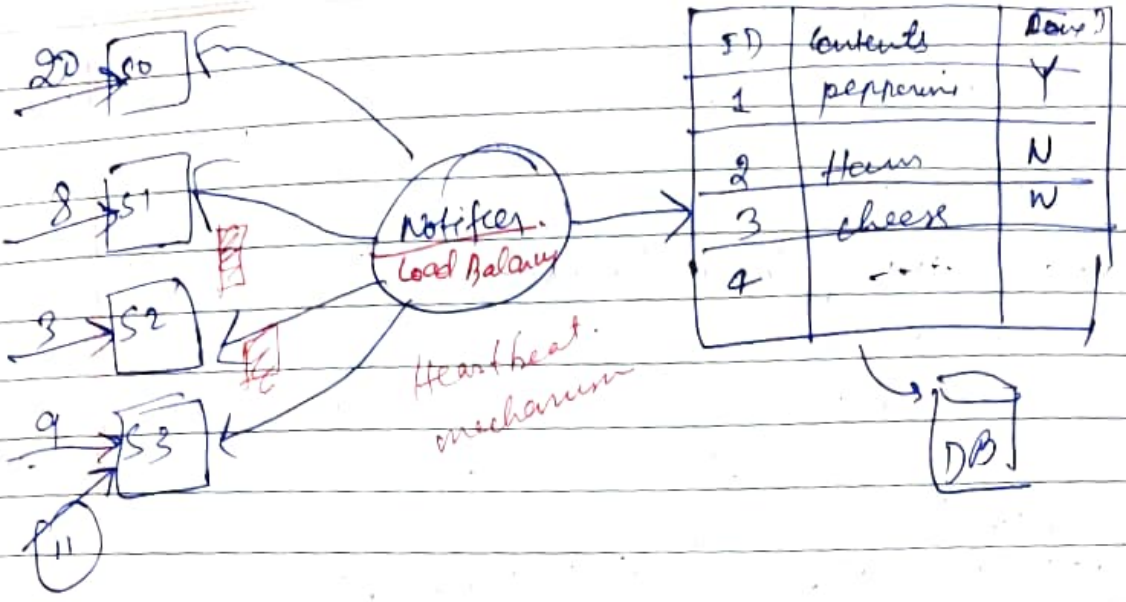
• Pizza bn rha h order ka aur jaise hi client ko pizza diya usne bola pay karne k money wo diya, also list se uska order hat gya.

↳ we can manipulate the queue like some work can be done instantly like (eg:- fetching coke). This can be done by Asynchronous processing

↳ like 3 pizza shop ka outlet h aur ek outlet kisi kaaran bnd ho gya. then the delivery of that order will be fulfilled by other outlet. This cannot be done by list for this we need some persistent in data for which we need DB. & this list will be stored in DB

Consistent Hashing is the technique by which we get rid of Hashing.

87



for eg S3 gets fault. one way is to check DB which orders belongs to S3 & we can note down the server ID which is handling the order everytime when it is making the entry. this is complicated

INSTEAD

Let there be Notifier which ↑ after every 10 sec (eg) is given to check some heartbeat on each server asks whether the server is alive if server will not respond notifier will assumes that server as dead & it all cant handle orders

↓

then it will query the DB to find all of those orders which are not done. once they are not done it picks those orders & distributes it among remaining servers.

Problem

what if there is duplication & for eg:- order 3 is not yet done & is picked by query that it is not done & get distributed to server 1

System having MQ can move to higher level requirements while abstracting implementation details of message delivery & event handling to the message queue.

Then order 3 will be in both server 1 & server 2. Then there will be big loss & lots of confusion

We can do

- Some sort of load balancing seems like sending right amount of load to each server.
  But principles of load balancing ensures that you don't have duplicates request to same server

Consistent Hashing
Has principles will take care of two things

(1) Balancing the load
(2) Not sending duplicates to same server

Reason is that S1 is going to handle some set of buckets S2 is going to handle some set of bucket.
Once this server crashes S2 won't lose its bucket. It will get new buckets added to it. S1 will also get new bucket added to it & therefore 3rd order will now come bcoz they belong to S2 even now & other orders like 9 & 11 come it

Through load balancing & some sort of heartbeat mechanism we can notify some sort of failed orders to newer server.

- If we want all features of assignment & notification, load balancing, a heartbeat & persistant in one thing. That would be a message queue / Task queue

- In case m/c crashes → Fault tolerance
- Scalability → m/c need to be added to process more requests (Primarily used in consistent hashing)

what it does it that it takes tasks, processes them, assign them to correct server & waits for them to complete & if it takes too long for server to give an acknowledgement, it feels that server is dead & then assign it to the next server.

Using messaging queue or task queue we get work done easily so that we can encapsulate all the complexity into just one thing.

eg:- Rabbit MQ (messaging Queue), Zero MQ, JMS (Java messaging Service)

MQ are really good encapsulation for complexities in the server side

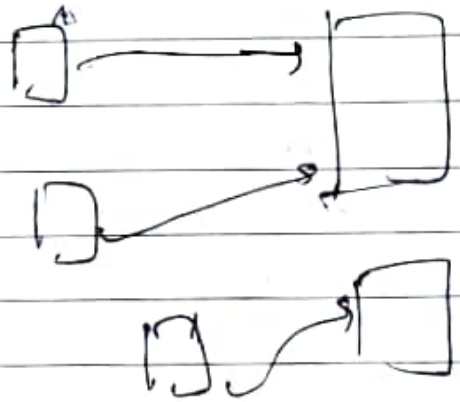
- Consistent Hashing allows request to be mapped into hash buckets while allowing the system to add & remove nodes flexibly so as to maintain a good load factor on each m/c
- Standard way to hash object is to map them to a search space & then transfer the load to the mapped computer. A system using this policy is likely to suffer when new nodes are added or removed from it.
- Consistent Hashing maps servers to the key space & assigns request (mapped to relevant buckets, called load) to the next clockwise server. Servers can then store relevant data in them while allowing the system flexibility & scalability.

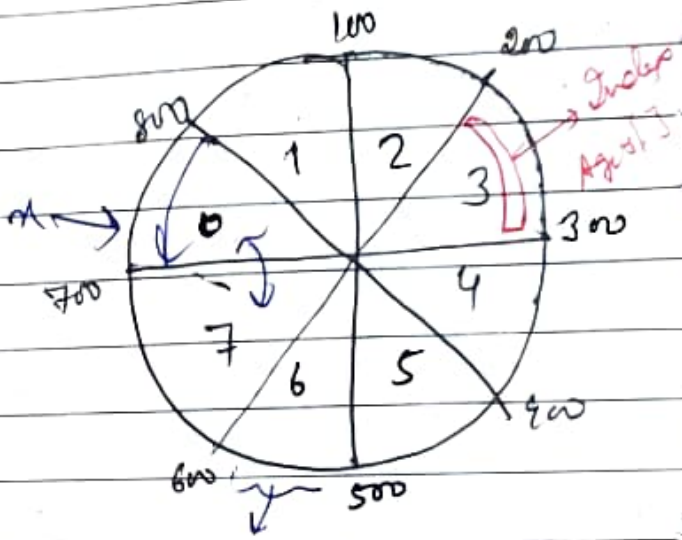
What is Database Sharding

→ **Sharding a Database** → Scalability strategy used when designing server side system. It uses concepts like sharding to make system more scalable, reliable & performant

Sharding is horizontal partitioning of data according to

Shard key → It determines which DB the entry to be persisted is sent to. Some common strategies for this are reverse proxies





Server 6

- partioning which uses some sort of key to break the data into pieces & allocate them to different servers is called Horizontal partitioning
- Horizontal Partioning depends upon 1 key which is the attribute of the data which we are storing to partion data
- Vertical Partioning which uses columns to partition data effectively

- Sharding (Horizontal Partioning)
  - Sharding is taking one attribute in the data & partioning the data such that each server gets one chunk. Server here is DB server

Sharding
↓
Database Server

- SHARDING
  - Consistency
    One of the key attribute of any DB that whatever data you persist in it is what we can read out of it later on & there is some sort of synchronization that if a person is going to make update then new request is going to read that update
  - Availability
    Means DB should not crash & should stay down but we want our application to be running all the time

like if X fall in 7th Shard, we just need to read through this shard which is what this DB Server 7 can do.

Advantage
- Shard is going to be smaller in size,
- Easier to maintain
- faster performance

Disadvantage

① Joins across shard

query needs to go to two diff. shards. They need to pull out the data then join the data across the network. & this is going to be extremely expensive.

② Shards are inflexible

but we want our DB server to be flexible in number. So one of the good algorithm is consistent hashing.

To overcome this problem we do is take a shard which has too much data in it & then dynamically break them into pieces. There will be some sort of manager for every particular shard which is going to map the request to the correct mini pieces in shard using this hierarchical sharding we can get rid of inflexibility

One smart things we will do is to create an index of these shards

assuming that our query requires this index would be completely different attribute compared to user ID

Find all people in New York ? 50 } → New York lies in server 4. we will find all users in this given range.

All our queries are fast is fast bcz our read pe & write performance goes up bcz all of our queries fall on one particular point.

what happens if shard fails

↳ we have something like master/slave architecture

↓

we have multiple slaves which are copying the master whenever there's a write request it's always on master while slaves continously pull the master & read from it

↓

If there is a read request it can be distributed across the slaves while if there is write request it goes to master

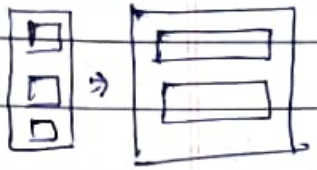
↓

In case master fails the slave choose one master among themselves so there is good single point of failure tolerance over here

Conceptually its very easy to take the data, break it into ~~pieces~~ ranges & then persist in different places but practically it is tough bcz consistency is difficult to do.

Two diff ways of ↑ing the capacity of system or application

Vertical Scaling



involves adding more resource to single m/c

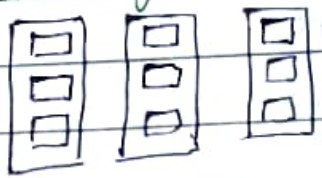
↳ adding more resource (such as CPU, RAM & storage) to single server or m/c. (also known as scaling up)

↳ eg:- adding up more RAM to server ↑s its processing power or we might upgrade its CPU to make faster

↳ can be quick & easy way to ↑ the capacity of system, but there is a limit how much a single m/c system can be scaled

↳ Problem: we can't fit enfinitely much CPU power into single m/c

Horizontal scaling



add more servers & merge data into one database (scaling out)

↳ eg:- instead of adding more resource to single server, we might add more servers to cluster or network to distribute the workload which allows system to handle more requests & users at the same time

↳ setup is more complex than vertical scaling

↳ offers unlimited scalability & high availability by distributing the workload across multiple m/c

→ involves adding more m/c to a system or applications