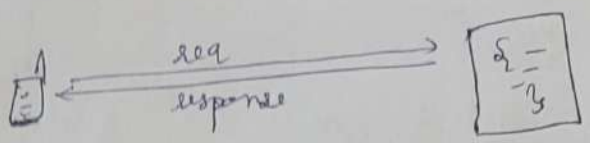
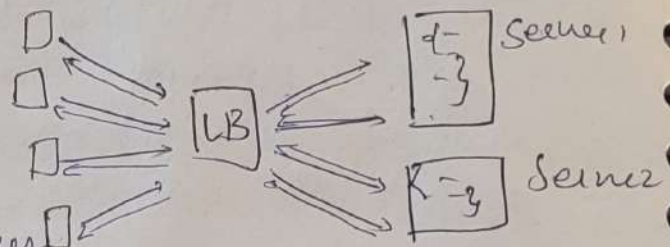
Vid 3 consistant Hashing



→ we have a code in server.
→ A person is wants to use our code & is ready to pay in place of code
→ client/person sends request, & we want our code to run properly & give response so we both should be happy
→ client is happy & he ~~wants~~ tells others to use as well
→ So our no of inp/client/ requests increases and one server is not able to handle many req
→ So we will add another server as we are getting money from other people

→ When we inc the no of servers, there should be something that balances the requests among our servers that is load balancer

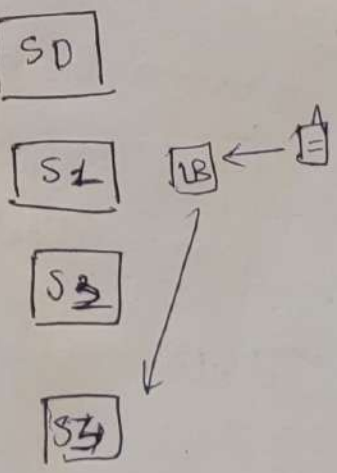


servers have load.

CONCEPT OF CONSISTANT HASHING

would help us to take the requests from clients and distribute the requests among servers evenly

① When a request is sent, a req id is generated from (0 - M-1) ~~M = no of servers~~ = r_1
② hash that request id. $h(r_1) = m_1 \% n$ after hashing we get a no as m_1
③ modulo hashing value ie m_1 with no of servers



Eg req no = 10
hash (req no) = hash(10) = 3 → random
3 % 4 = 3 so req would be sent to 3rd server

Que2 request no = 20, hash(req no) = 15, no of servers = 4
find to which server request would go

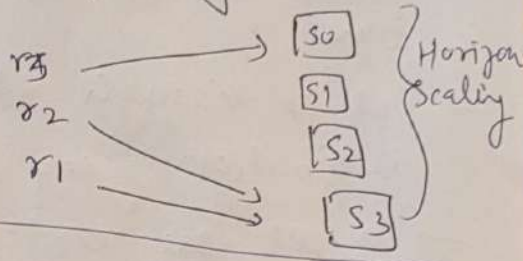
Ans req no $r_2 = 20$ no of servers $= n = 4$
hash $(r_2) = 15$
server req = hash(r_1) % 4 = 15 % 4 = 3

So our request would go to server 3

Que 3 ~~k~~ $r_3 = 35$ $h(r_3) = 12$, $n = 4$

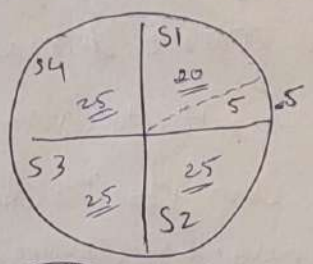
Ans $h(r_3)$ % n = 12 % 4 = 0 So req would go to 0th serv.

As req no & hash no both are random so we can say that load is always 1/n ie equally divided



OLD WAY OF HASHING

Pie diagram



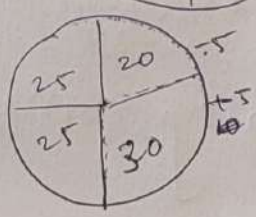
we have 4 servers each is taking 25% buckets

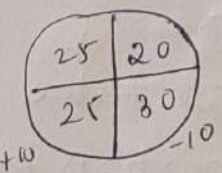
→ take 5 - from 1st server & merge with 2nd
2nd = 30

→ take 10 - from 2nd server & merge with third
3rd = 35

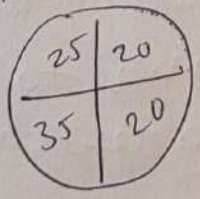
→ take 15 from 3rd server & merge with 4th
4th = 40

→ take 20 from 4th server & create 5th server

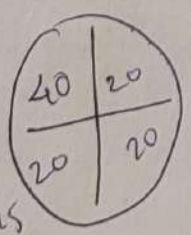


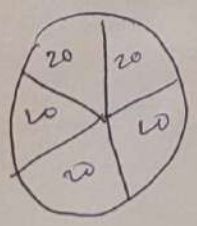


initially - 25 + 25 + 25 + 25 = 100 → for 4 servers
S1 S2 S3 S4



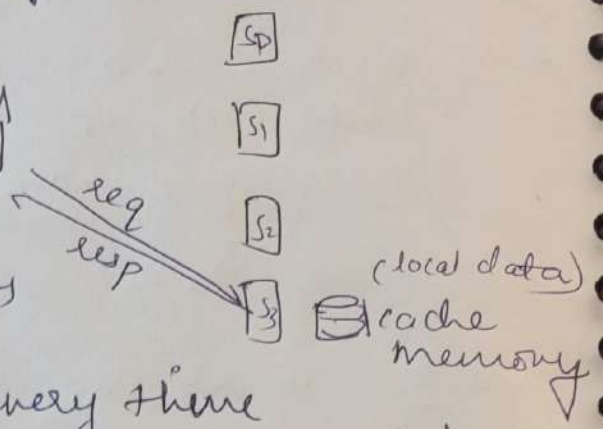
after ~~5 ser~~ = 20 + 20 + 20 + 20 + 20 = 100 → for 5 servers
S1 S2 S3 S4 S1





we have added 5th server.

Problem of old way of Hashing & needed the consistant Hashing



→ suppose a user sends req that goes to S3 (in case we have 4 servers)

→ Our S3 processes the req and shows the required info.

→ we know in case of 4 servers, every time the r1 request is sent, the same hash is generated & the req will go to same server

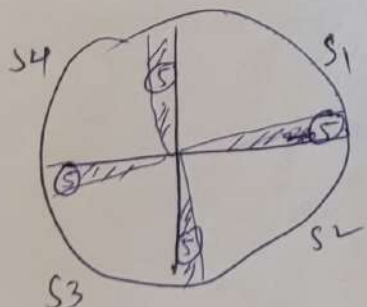
→ So instead of processing the same request, we can store the request in cache memory, so it will save time.

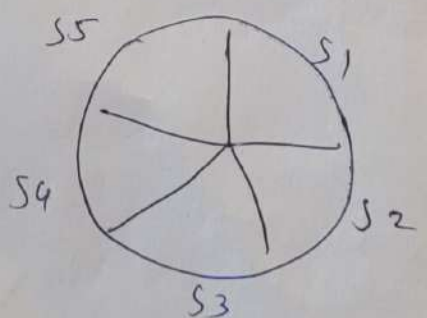
→ But if through old way of hashing → (inc no of server & distribution of req among servers) there was drawback.

→ as we are randomly sub space from one S1 & adding to other S1 due to this the there are more chances that request going to specific server, it doesn't have the processed data of req r1 in its cache memory → (every time server inc or dec, a new hash code needed to be generated

* So the concept of consistant Hashing comes here

→ here we take small portion of buckets from each ~~memory~~ server and create new server, we take small portion from each server to create a new one

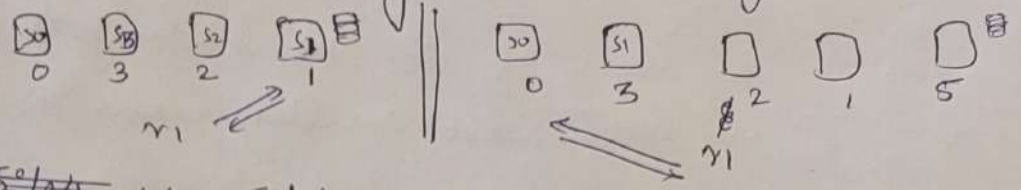




Video 4

→ In consistant Hashing the local data that resides in each memory will also get distributed

Eg

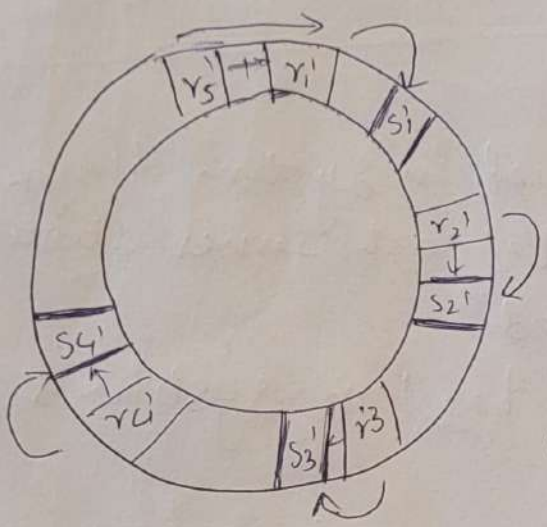


~~5%4=1~~ $5\%4 = 1$

req will go to S1
& data will be in S1 cache memory

$5\%5 = 0$

data is in S5 but req is going to S0 after consistant hashing



$h(r_1')\%n = r_1'$ $h(S1)\%n = S_1'$
$h(r_2)\%n = r_2'$ $h(S2)\%n = S_2'$
$h(r_3)\%n = r_3'$ $h(S3)\%n = S_3'$
$h(r_4)\%n = r_4'$ $h(S4)\%n = S_4'$
$h(r_5)\%n = r_5'$

So hash the server as well

So S_1' server will have 2 requests
S_2' server will have 1 req
S_3' server will have 1 req
S_4' server will have 1 req

~~Suppose S4 req~~ Here each request would go to the its nearest ~~req~~ server

→ in case S'4 breaks the request r_4' r_5' & r_1' will go to S_1'

→ It always works in clockwise direction

Theoritical load = 1/N

~~Suppose~~ now in above scenario when S4 breaks so half of load is going to be on S1

So what to do ?

→ buy virtual server

ie make multiple hash function

r_1'	h_1^1	h_1^2
r_2'	h_2^1	h_2^2
r_3'	h_3^1	h_3^2
r_4'	h_4^1	h_4^2

So if we do multiple hashing on same server.
so even if its one hash gets failed we would have another hash as bkp

Hence the chance of load on one server is very low

Where can be used ?

→ web caches
→ data balancing

CONSISTANT HASHING

LB → load balancing is key concept in system design that defines no of req sent to each server should be nearly equal or distributed.

→ one of the way to balance load in system is using consistant hashing.
→ Constant hashing allows the requests to be mapped to hash buckets while allowing the system to add & remove the nodes flexibly.
to maintain good load factor on each Machine
→ We do is hash the request & hash the server
each hash(request) would be mapped to its nearest hash(server) in clockwise direction, That server will store the req data
→ Suppose one one Server crashes, so its hash(server) would also be crashed, in that case the hash(req) would go to the next nearest hash(server) in clockwise.

→ But next hash(servers) would not store ~~pr~~ new req details already, it will process & then store req ~~req~~ details in its cache memory

→ So we ~~will~~ will do hashing of each server with different function.

→ So if one hash server gets failed the other would be there.

So consis Balancing → gives load balancing and allows to add & remove machine in efficient way.