Page No. 13
Date / /

HIGH LEVEL DESIGN

HLD 101

Difference between HLD & LLD
HLD explains how the relationships between the components of a software system are designed.
LLD talks more about how the code inside these components is structured so that if any change needs to be brought in, minimum lines of code will be changed.

Vertical Scaling – When you replace the current hardware with better hardware.

Delicio.us

- Delicio.us was an extension (before Chrome) that was used to sync bookmarks across multiple machines and browsers using email-password authentication.
- This company was later bought out by Yahoo.

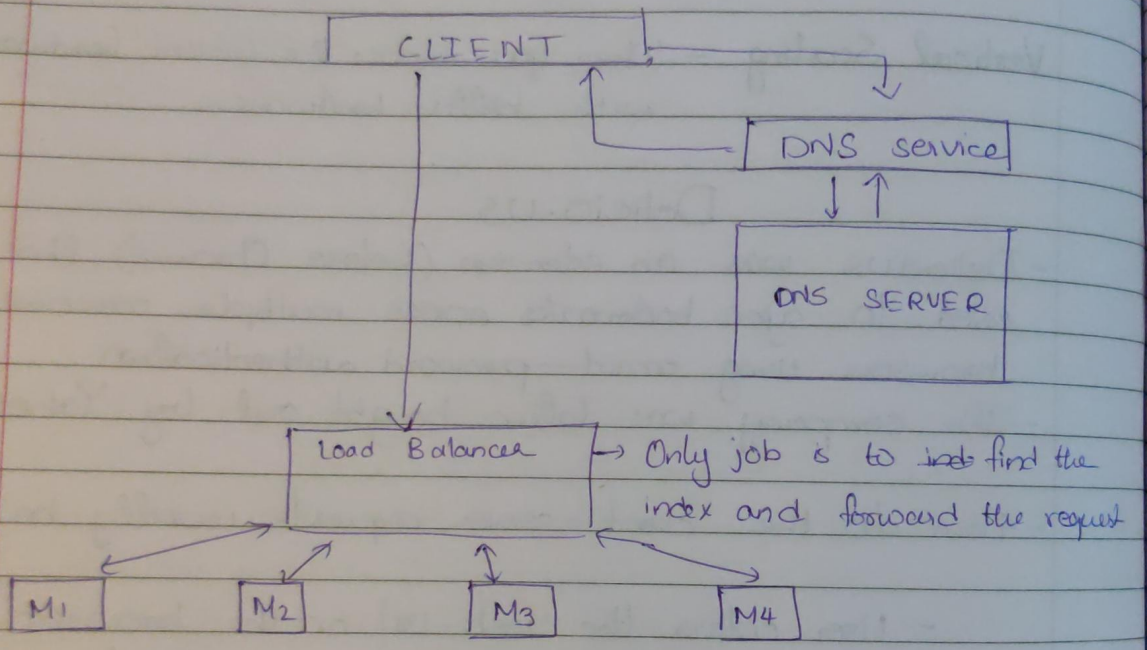
- How do the client-server requests usually handled?

- User enters the web-url on the browser.
- Browser does not understand the urls.
  So, it contacts DNS services - which have access to DNS servers.
  DNS Servers :- Converts urls to ip addresses / Finds the ip addresses of URLs/websites
  - ICANN maintains the url-ip repos.
- The browser with the IP address can now visit the site on the server.

- Now Delicious was hosted on a laptop in the beginning on a port (assume 80).
- After a while, the traffic started increasing.

Vertical Scaling:-

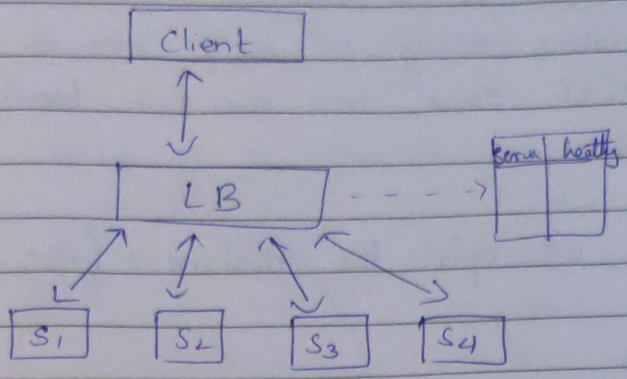
Option 1 - Increase the memory of the current machine.
Horizontal Scaling
Option 2 - Use parallel machines.
- for this, you might wanna use a load balancer to avoid some of the bottleneck.



- There are two problems with this.
  1. How does the load balancer know which server to send the request to?
  2. How do we split the DB and store it on multiple machines.

① Let's look into the ways we can utilize a load Balancer

- Consider a system (array of servers) only doing a summation (a+b)



- To route the requests, we can go with
  - I. Round Robin
  - II. Least Response Time
  - III. Least loaded Server.
  - IV. Weighted Round Robin
- Consider Round Robin for the time being.
- Here, the problem is that if one server goes down, every 4th request is failed.
- We can deal with that issue by checking the health of the server before sending it a request.
- For example, if S3 is down, we can skip over it and forward ~~it~~ the request to the next healthy server.
- How does the load balancer know if the server is healthy or not? - LB maintains a hashmap.
- This hashmap could be updated using one of the following:
  - ① Heartbeat (pushed by the server - ping once every few moments to let the load balancer know the server is alive)
  - ② Health check endpoint / url (If the server does not respond in time or if the server sends false as it has reached its capacity, we mark it as unhealthy)

- Consider Least Response Time

- Let's say one of the servers is responding very slow for some reason.
- This means every 4th user has a bad experience
- In that case, we can use Least Response Time model.

② How do we split the database? (Also called sharding)

Thoughts :-

I. prefix machineID with userID
- In this case, if the current system ran out of memory too and if we want to move the users data to new servers, we would want to change the user IDs which is not a good design. (cached apps might break).

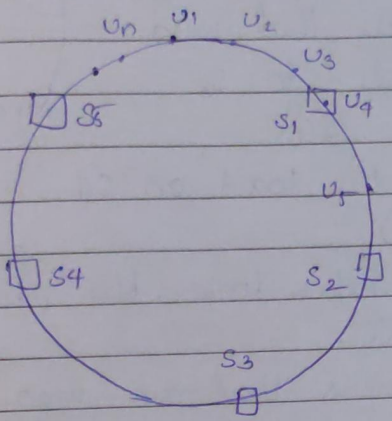
II. userID % no. of servers
- As soon as a new server is added, no. of server ie., the denominator is changed. So, almost all the users need to be moved.

Solution - Consistent Hashing
- Consistent Hashing is used almost everywhere in HLD

- Constraints we have
  - Load Balancer cannot have a lot of data
  - Load Balancer should somehow know which server to route the request to
  - Addition or deletion of servers should not cause a lot of data to move.
  - Cannot change userID

- We will have 2 hash functions.

Hash_1 (user_id)
Takes the user_id and converts/generates a long long int in the range of $[1, 10^8]$ inclusive
Hash_2 (server_id)
Takes the server_id & does the same.



- When we get a user id, we generate hash_1 & assign it to its next nearest server on the cyclic edge.

- Implementation

Arrange servers in a sorted order, sorted based on their hash values.

server_hashes - | 9 | 2800 | 4000 | 4300 | 98000 |
0 S1 1 S3 2 S2 3 S4 4 S5 5

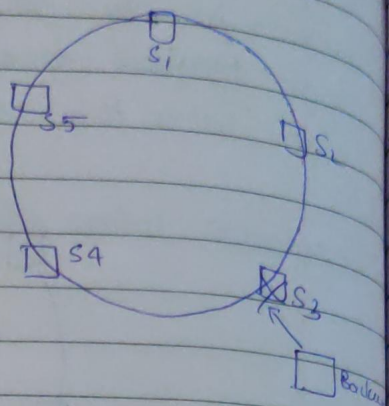
- When we get a user_id, we can use hash_1, get the value x & perform binary search on "server_hashes" to find the next greatest server and assign/route the request there.

Addition of the servers :-

- Let's say we added Sx server and it falls between S3 & S4.
- Now, we can ~~copy~~ move all the users that need to be present in Sx ~~from~~ S4 there, only some data is copied over. So, that is acceptable.

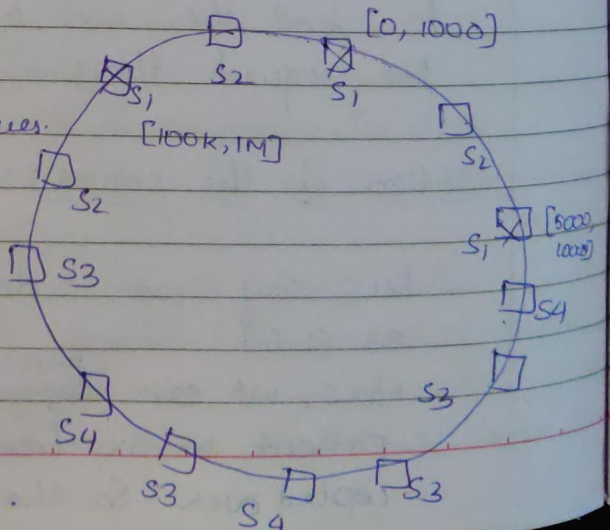
- The concern with this addition is that, its taking load off of only one server.

Deletion of Servers :-



- Let's assume that for some reason S3 failed.
- Now, all the users that need to be added / routed to S3 are diverted to S4.
- Now, effectively, the load on S4 is doubled (2x).
- Because of this extra load, the chances are S4 goes down as well.
- Now all the users that are meant for S3 & S4 are diverted to S5. Here, the load is 3x now.
- This will probably fail too.
- This phenomenon is called CASCADING ~~EFFECT~~ FAILURE
- To overcome the problem, we go with Modified Consistent hashing.

Modified Consistent Hashing



- We provide each server with multiple hash values. [Look at the figure]
- Let's say S1 died.
- All the user's who match with [0,1000] hash will be moved to S2.
- [5K - 10K] to S4
- [100K - 1M] to S2 again.

① Since the hash values are spread out randomly b/w $[1, 10^{18}]$ or some such, we can ensure that all the servers are getting about equal load.

NOTE: We do not have 3 physical servers representing each server. We just assign multiple hash values by using multiple hash fns.

server_id ——→ passes through 5 hash fns (or any)
user_id ——→ passes through 1 hash fn

- Having more hash fns spread out the data more evenly.
- But this will increase the amount of data stored on the load balancer.

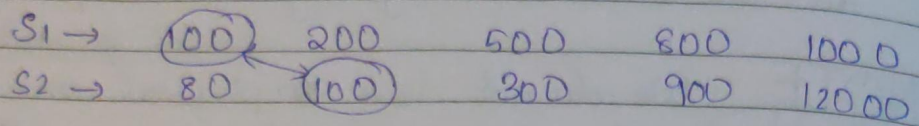
② When a new server Sx is added, it also has multiple hash fns and it will be plotted sparsely all around the cyclic edge.
So, the server takes off load from multiple (almost all) servers evenly.

Hypothetical Hash

- To make it fall in the range $[1, 10^{18}]$

```
hash(user_id):
    m = md5(user_id); num = 1
    for c in m:
        num = num * 36 + c
        num = num % 10^18
    return num
```

Hash Conflict :-
- What happens if two hashes collide?



- You can have a priority. Either S1 > S2 or hash_1 > hash2 or vice versa.

Stateless LB :- - It does not matter which server is contacted / requested.
- Round Robin with healthe check could be used.

Stateful LB :- It matters which server is requested for.
- When data is sharded.