Consistent Hashing

In horizontal scaling, one problem we faced of redistribution of user ids when some server fails or a new server is added to the system.

Hash space

In consistent hashing, we have hash values in a ring to which user ids and server ids are mapped. The hash function used should be **uniformly distributed**.

If we use hash function → SHA-256 then ring space goes from `xo = 0` to `xn = 2^256-1` in a ring.

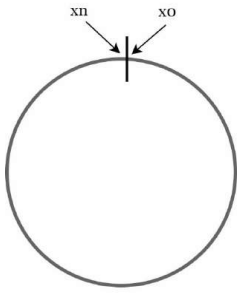
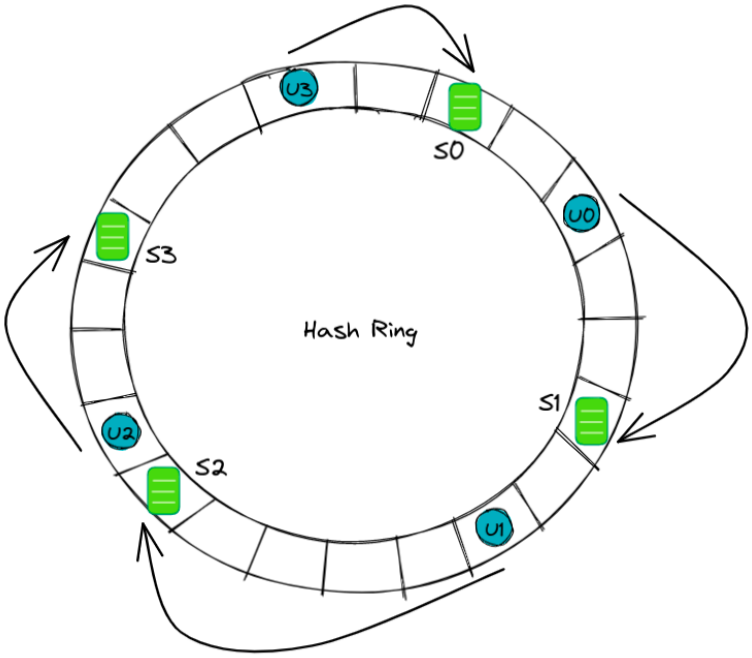


Figure 5-4

Hashing servers and users

Now, using the same hash function we map our servers on this ring using their IP or name.

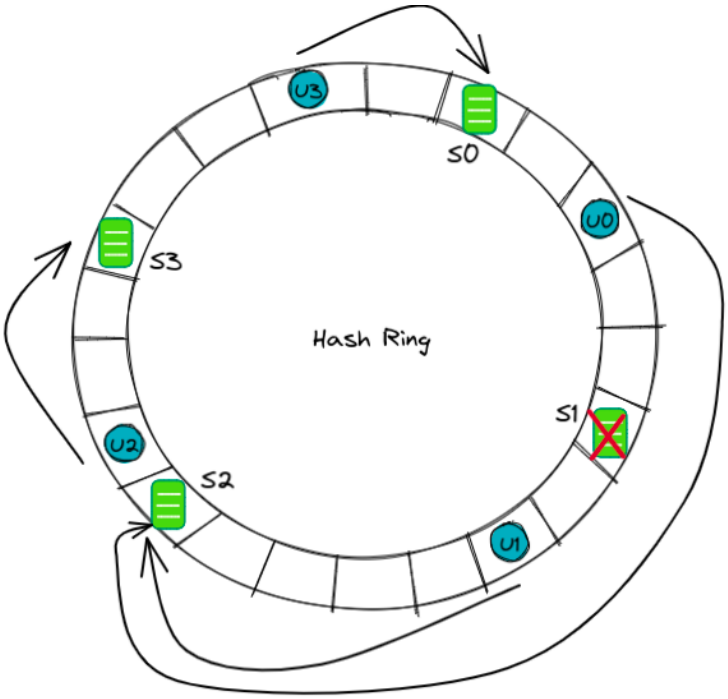


Now, the mapping of a user to the server is done by moving in a clockwise direction starting from a particular user till it sees any server.

Here, u0 → s0, u1 → s1, u2 → s2 and u3 → s3.

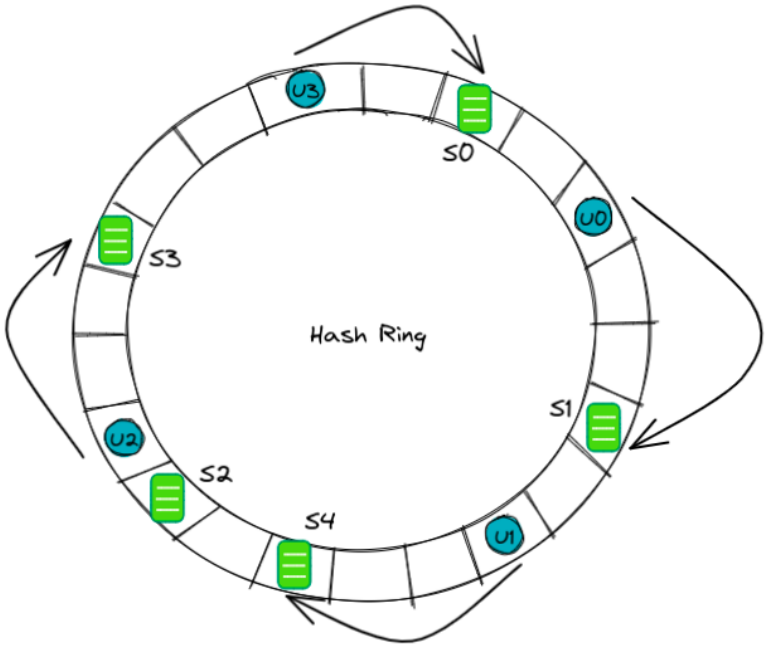
Remove server / Server fails

Now a server fails, all its users are mapped to the next server which it sees while moving in the clockwise direction in the hash ring.



Add new server

Now we add new server s4, then users mapped between hash (s1, s4) will get mapped to server 4.



Problem:

1. When a server fails, the gap between all servers becomes non-uniform, which can lead to non-uniform distribution of users to servers. This, in turn, causes a high load on a particular server.

This can be solved using **virtual servers.**

Virtual nodes

Instead of just one hash, we calculate k different hashes for the same server. So particular server can have location at any of the hash from a set of k hashes, which can be used whenever this is an unequal distribution of users.

> 💡 The outcome of an experiment shows that with one or two hundred virtual nodes, the standard deviation of gaps is between 5% (200 virtual nodes) and 10% (100 virtual nodes) of the mean.

Conclusion

1. Minimum server hashes are to be distributed on server failure or new addition of server.
2. Prevents excessive load on a single server by evenly distributing users' requests (load).
3. Real-world use cases: Amazon Dynamo DB, Discord chat, etc.