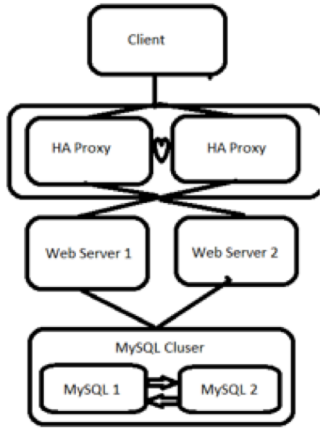
High Availability Clusters

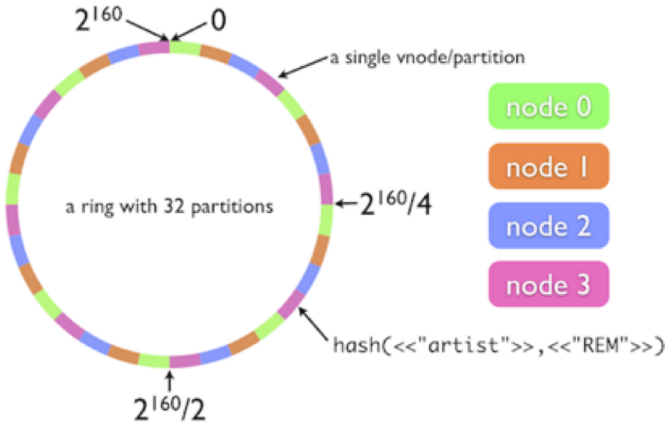


Question:
If a load balancer is used - isn't that the single point of failure?

Answer:
Yes, it will become a single point of failure if it is a single device.
So to avoid this, we need to use more than one balancer,
and we need to run them as a HA (High Availability) cluster.
- https://en.wikipedia.org/wiki/High-availability_cluster

There are multiple ways to achieve High Availability.
For example, consider simple fail-over mechanism with just two servers (master/slave).
They receive the same input, do the same calculations, constantly in sync, but only the master provides the output.
The slave server monitors the heartbeat of the master. If the master server dies (heartbeat stops), the slave server becomes the new master.

In the above simple master/slave architecture, the slave is mostly doing nothing.
In real life you may have many servers receiving same inputs and separating their responsibilities to achieve higher performance. Then if one of the servers dies, others can take over its responsibilities.
Common method of separating responsibilities (and doing fail-over) is called "**consistent hashing**".
- https://en.wikipedia.org/wiki/Consistent_hashing
Good explanation:
- https://dzone.com/articles/simple-magic-consistent
Original paper (1997):
- Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web – by David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin, Rina Panigrahy.



Consistent Hashing

Consistent Hashing Explained

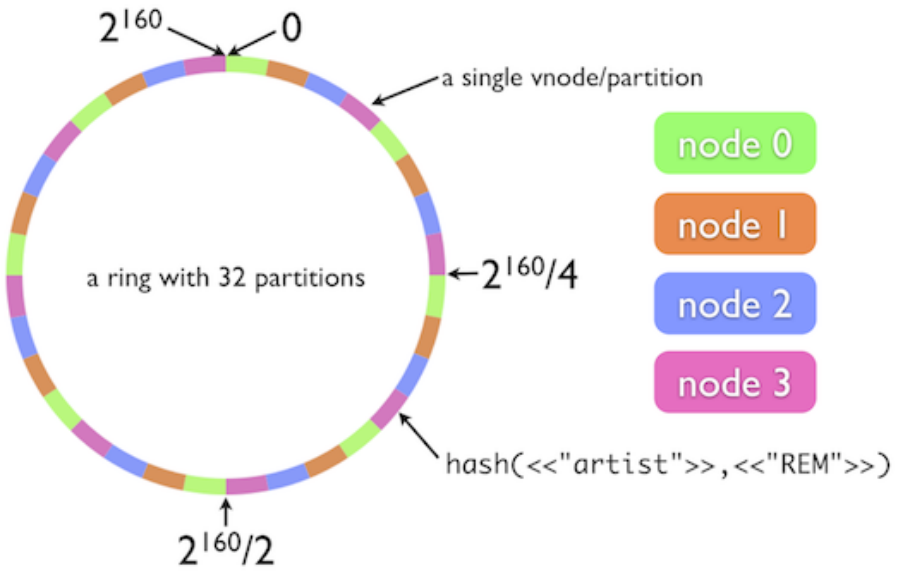
Adapted from:
- https://www.mikeperham.com/2009/01/14/consistent-hashing-in-memcache-client/

How do we distribute requests from clients between several servers.
Suppose each client has a key. Or we generate a key based on its IP address and port.
Now we can hash the key to an integer and do a modulo based on the size of the server set

```
server_idx = int_hash(key)    %    N_servers
```

So for a given key we get the same server every time.
But what if one of the servers dies ?
Here is a better algorithm.
Instead of using modulo, we can use random numbers.
Suppose N_servers=20, and for each server we want 200 random numbers.
Total: 200*20=4,000 values
Let's take a big range of numbers [0,2^160] and put them on a circle.
Let's randomly choose 200 numbers for each server (no sharing, please).
So now on the circle we have 4,000 positions which correspond to servers.

So now how we map a key to a server?
We use hash function to map key to some value in range up to 2^160.
We find corresponding position on the circle.
We move clockwise until we hit the first position corresponding to some server.
DONE.



Notes:
- Randomness helps to avoid hotspots
- Consistent Hashing Enables Partitioning
- Partitioning Makes Scaling Up and Down More Predictable
- Consistent Hashing and Partitioning Enable Replication
- There doesn't need to be a master for any piece of data
- Every node is simply a replica of a number of partitions
- Replication Reduces Hotspots (Even More by using load-balancing)
- Consistent Hashing Enables Scalability and Availability