Jahong Liu

System Design Basics 13: Consistent Hashing

Consistent Hashing

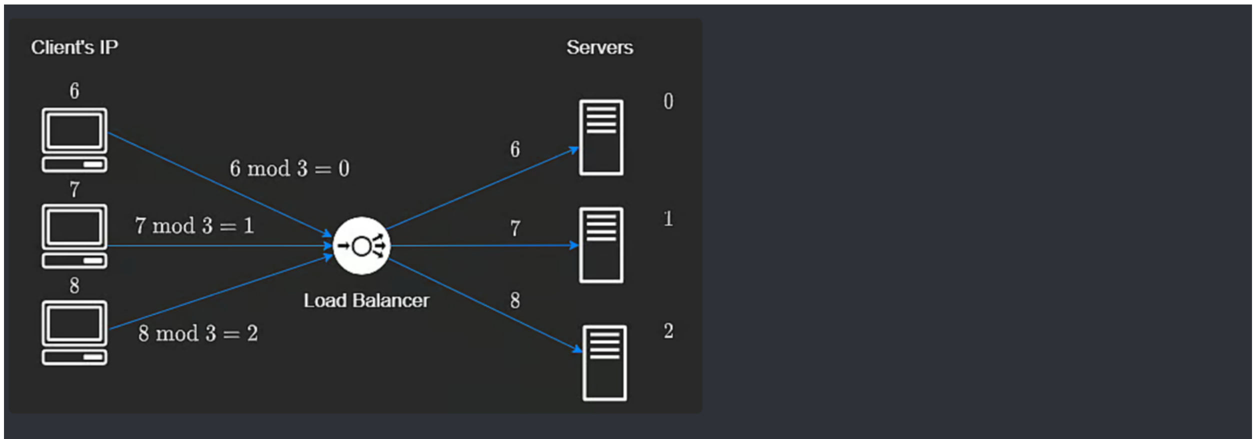
Hashing is another technique that can be used to map requests to servers in the context of load balancing. We discussed hash functions in the Data Structure and Algorithms. The concept is similar. Each request will have an IP address, user ID, or a request ID associated with it. If we can calculate the hash of this unique identifier and mod it by the number of servers we have available, the answer will allow us to assign the request to a specific server.

Suppose we have three servers, number 0, 1 and 2. If we take the IP address as the unique identifier of three incoming requests, hypothetically 6, 7 and 8 respectively, the mapping becomes:

```
6 % 3 = 0
7 % 3 = 1
8 % 3 = 2
```

We can probably notice that the same IP address will always be directed to the same server. The benefit of this is that each server can cache data that belong to a specific user.

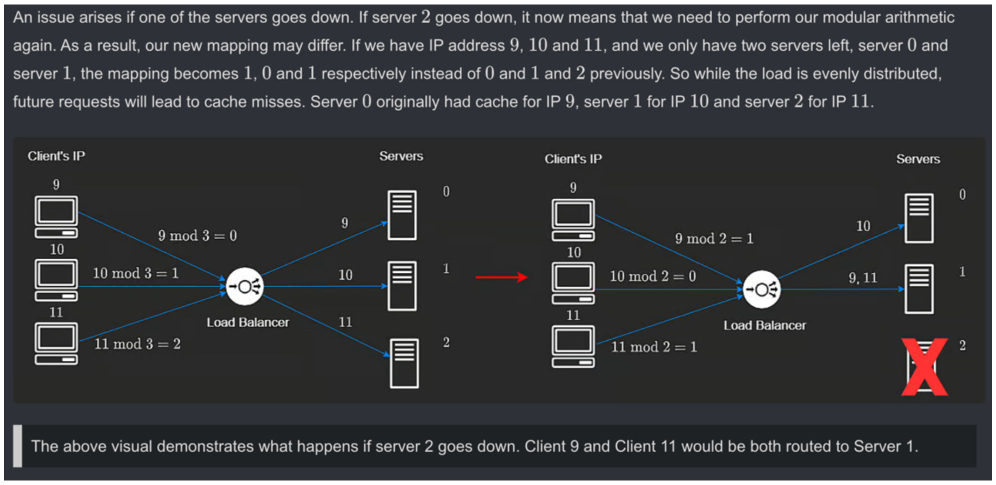
This makes sense because it is expected that the user with IP address 6 will always be directed to server number 0. You can see how this is different to round robin because the round robin technique is not consistent in assigning the user with the same IP address to the same server each time.



The above visual demonstrates the assignment of Client's requests among three servers using a load balancer.

Issue with regular hashing

An issue arises if one of the servers goes down. If server 2 goes down, it now means that we need to perform our modular arithmetic again. As a result, our new mapping may differ. If we have IP address 9, 10, 11, and we only have two servers left, server 0 and server 1, the mapping becomes 1, 0 and 1 respectively instead of 0 and 1 and 2 previously. So, while the load is evenly distributed, future requests will lead to cache misses. Server 0 originally had cache for IP9, server 1for IP10 and server 2 for IP11.

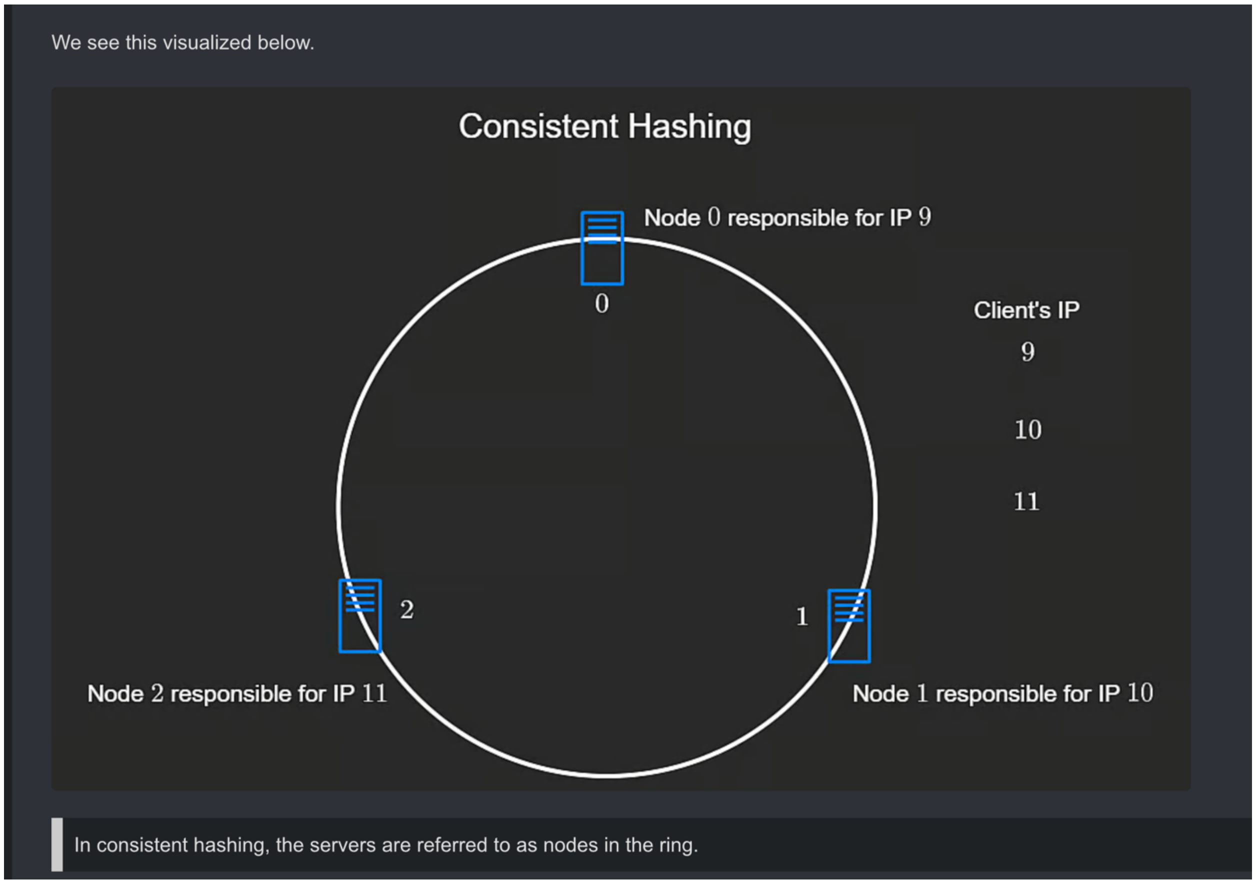

An issue arises if one of the servers goes down. If server 2 goes down, it now means that we need to perform our modular arithmetic again. As a result, our new mapping may differ. If we have IP address 9, 10 and 11, and we only have two servers left, server 0 and server 1, the mapping becomes 1, 0 and 1 respectively instead of 0 and 1 and 2 previously. So while the load is evenly distributed, future requests will lead to cache misses. Server 0 originally had cache for IP 9, server 1 for IP 10 and server 2 for IP 11.

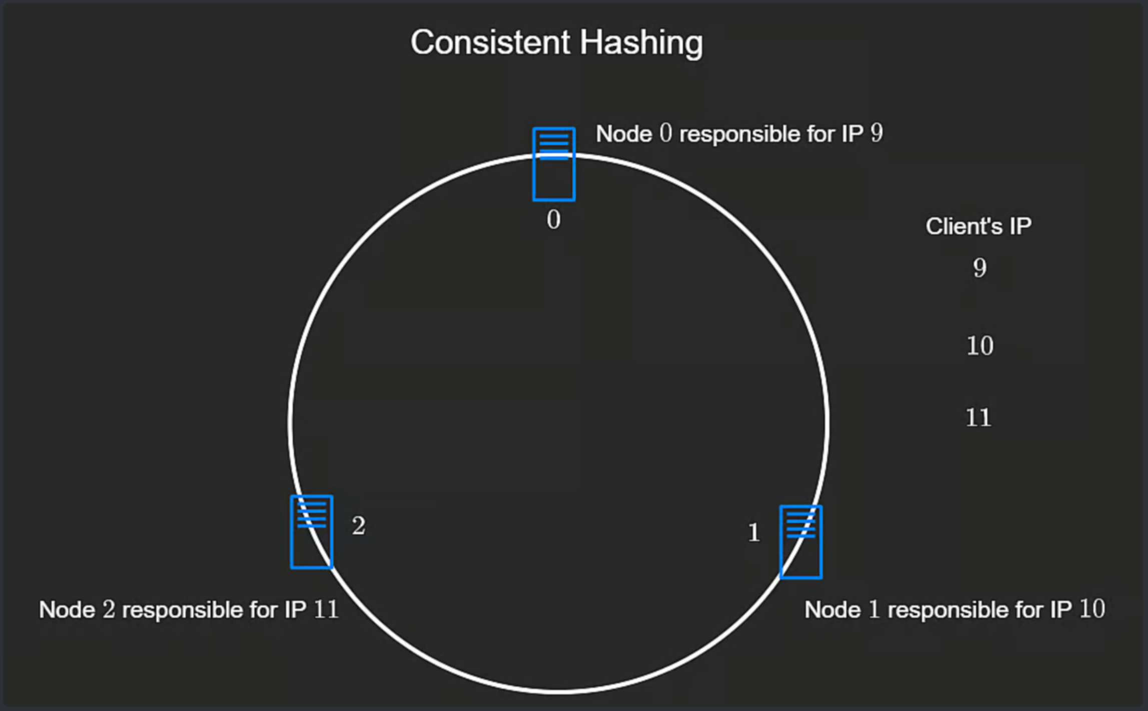
The above visual demonstrates what happens if server 2 goes down. Client 9 and Client 11 would be both routed to Server 1.

The solution

Consistent hashing is a technique that enables uniform load distribution across servers in a load balancing system. It uses a ring-based structure and a hash function to map requests to servers. If a server goes down, the requests it handled are reassigned to the next available server in a clockwise direction on the ring.

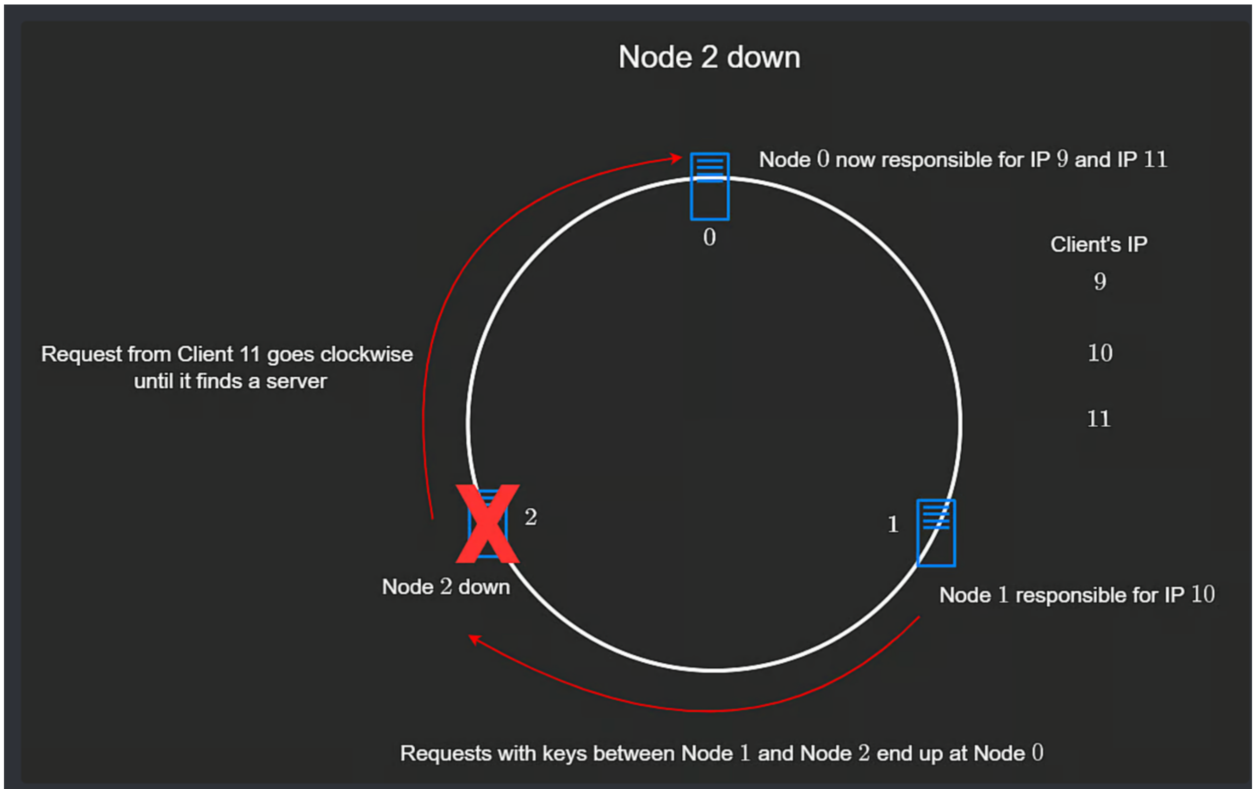
The ring represents the space of all possible values. Each server in the system is also represented as a point on this ring. The hash function takes the content of the request (IP address) and maps it to hash value within the range of the ring. When a request arrives, we calculate its hash value using the hash function, and locate the next server on the ring that is equal to or follows the hash value. This is the server that is responsible for handling the request. If a server goes down, its portion of the ring becomes unavailable. In consistent hashing, the requests that were previously handled by that server are reassigned to the next closest server in a clockwise direction on the ring.


We see this visualized below.



Consistent Hashing
Node 0 responsible for IP 9
0
Client's IP
9
10
11
2
1
Node 2 responsible for IP 11
Node 1 responsible for IP 10

In consistent hashing, the servers are referred to as nodes in the ring.


Node 2 down
Node 0 now responsible for IP 9 and IP 11
0
Client's IP
9
10
11
Request from Client 11 goes clockwise until it finds a server
2
1
Node 2 down
Node 1 responsible for IP 10
Requests with keys between Node 1 and Node 2 end up at Node 0

Notice above that if Node 2 goes down, all of the requests that would have gone to Node 2 now end up at Node 0.

For the sake of simplicity and demonstration, we have kept the Ips to single digits and the space as 360 degrees. Mathematically, we would use a sophisticated hash function to calculate the hash of the IP, and mod it by **M**, where M represents the solution space, which can be said to be the number of positions or partitions on the ring. This number is often equal to the number of servers in the system.

> The reason we perform the modulus operation is because we want to ensure that the resulting value falls within the range of available positions on our ring (0 to M-1). But also, hash functions can produce large integers values which might exceed the range of available positions on the ring. Taking the modulus ensures that we prevent overflow issues. This is crucial to the uniform distribution of requests.

Closing Notes

At a high level, the provided information covers the essentials of consistent hashing. It is important to note that consistent hashing is just one approach to load balancing and does not invalidate the usefulness of round robin or weighted round robin methods. Round robin can effectively handle load balancing when caching is not a concern.

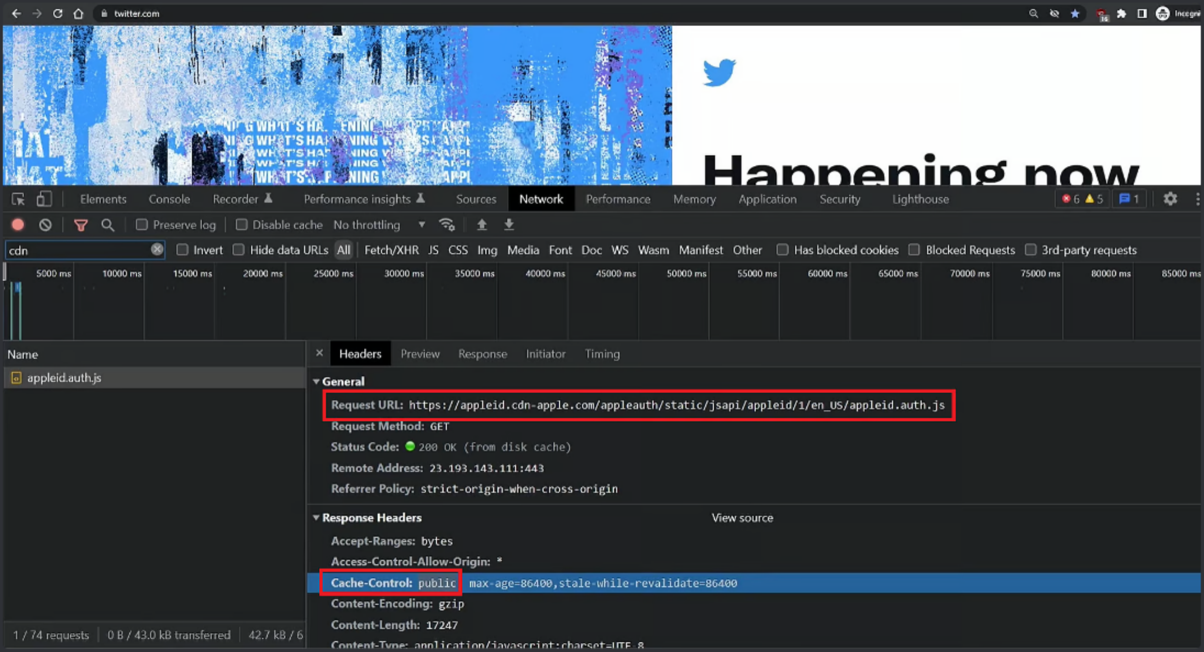
Consistent hashing finds applicability in scenarios such as Content Delivery Networks (CDNs), particularly when it is necessary to route specific users to the same cache servers in their respective regions. It can also be applied to databases where a user's data resides on a specific server, and consistent routing of that user to that server is desired.

By utilizing consistent hashing, CDNs and databases can ensure efficient and reliable routing, maintaining consistency for users accessing cached content or specific data.

Twitter Example

We can confirm that Twitter utilizes a CDN by visiting the Twitter website and inspecting the page. In the elements tab, we can observe that Twitter employs a CDN for Apple Authentication. By opening the corresponding URL in a new tab, we can view the JavaScript code associated with it. Regardless of how many times the file is refreshed, the code remains static. Twitter could have opted to host the authentication on their own server, requiring them to maintain a separate copy of the code. However, they have leveraged the CDN provided by Apple to handle this task.

It is worth noting the presence of the "cache-control" attribute in the visual representation below. When the "cache-control" value is set to "public," it signifies that the code should be cached by the CDN. On the other hand, if "cache-control" is set to "private," it instructs the CDN not to cache the content..



```
▼ General
  Request URL: https://appleid.cdn-apple.com/appleauth/static/jsapi/appleid/1/en_US/appleid.auth.js
  Request Method: GET
  Status Code: 200 OK (from disk cache)
  Remote Address: 23.193.143.111:443
  Referrer Policy: strict-origin-when-cross-origin
▼ Response Headers
  Accept-Ranges: bytes
  Access-Control-Allow-Origin: *
  Cache-Control: public max-age=86400,stale-while-revalidate=86400
  Content-Encoding: gzip
  Content-Length: 17247
  Content-Type: application/javascript;charset=UTF-8
```

> The visual substantiates the `Cache-Control` and the `Request URL` header mentioned earlier.

Closing Notes

The most crucial concept behind CDNs is their ability to bring content closer to the user. CDNs exemplify scaling, although it differs from horizontal scaling. While these two concepts may be interconnected, they are not identical.

Horizontal scaling aims to increase throughput, enabling the system to handle a larger number of requests. It involves adding more servers or resources to handle the increased load. On the other hand, CDNs focus on minimizing latency and improving content delivery by strategically placing multiple servers across the globe. By doing so, they ensure that content is geographically closer to the users, resulting in faster access and reduced network congestion.

> There is the indirect benefit of horizontal scaling, since CDNs can increase throughput, but that is not necessarily the purpose.

In summary, horizontal scaling aims to handle more requests by adding resources, while CDNs optimize content delivery by bringing it physically closer to the users through a distributed network of servers.