Load Balancing is a key concept to system design. One simple way would be hashing all requests and then sending them to the assigned server.

The standard way to hash objects is to map them to search space, and then transfer the load to the mapped computer. A system using this policy is likely to suffer when new nodes are added or removed from it.

Some terms you would hear in system design interviews are Fault Tolerance, in which case a machine crashes. And Scalability, in which case machines need to be added to process more requests. Another term used often is request allocation. This means assigning a request to a server.

Load balancing is often tied with service discovery and global locks. The type of load we want to balance is that of sticky sessions.

For load balancing how is the load balanced?
I feel normal hashing won't be effective rather some other hashing or implementation

Load balancing is a critical component in distributing incoming network traffic across a group of backend servers, known as a server farm or server pool. This process helps to ensure that no single server bears too much demand. By spreading the requests across multiple servers, load balancing improves the responsiveness and availability of applications, websites, or databases. There are several strategies and implementations for load balancing, and while normal hashing (like consistent hashing) is popular, there are indeed other methods and enhancements to address its limitations. Here are some common techniques used for load balancing, including hashing-based methods and their alternatives:

1. Round Robin

Round Robin is one of the simplest forms of load balancing, where each server is selected in rotation. This method doesn't account for the current load on each server, making it straightforward but not always the most efficient.

2. Least Connections

This method directs traffic to the server with the fewest active connections. It's more sophisticated than Round Robin because it considers the current state of each server, making it suitable for long-lived connections.

3. Resource-Based (Least Load)

Traffic is directed to the server with the most resources available, which could include CPU, memory availability, or network bandwidth. This method requires real-time monitoring of the servers' resources.

4. Hashing Methods

Consistent Hashing

Consistent hashing is a popular method used in distributed systems to achieve load balancing. It hashes both the request (e.g., the client's IP address or request URL) and the servers to a consistent hash ring or space. This method helps in evenly distributing requests and minimally affecting the distribution when servers are added or removed.

Rendezvous Hashing (Highest Random Weight, HRW)

This is an alternative hashing method where each request is hashed with each server, and the server with the highest hash value is chosen. This method is efficient in ensuring that requests are evenly distributed among servers.

Maglev Hashing

Google's Maglev load balancer uses a consistent hashing mechanism to evenly distribute incoming requests across a pool of servers. It addresses some of the limitations of consistent hashing by using a permutation-based approach to distribute traffic more evenly.

5. Dynamic Hashing

Dynamic hashing schemes adapt the hashing mechanism based on the current state of the server pool, such as the number of active connections or the server load. This method can provide a more responsive and flexible load balancing solution.

6. IP Hashing

Traffic is distributed based on the hash of the client's IP address. This ensures that a client is consistently directed to the same server, which can be beneficial for session persistence.

Custom Algorithms

Besides these, custom algorithms can be developed to meet specific requirements of the network or application. For example, an algorithm might consider the type of requests, the geographical location of the client, or even the time of day.

In summary, the choice of a load balancing method depends on the specific requirements of the application, including factors like session persistence, the variability of server load, and the importance of evenly distributing connections across servers. Advanced hashing techniques and other strategies are continually being developed to optimize the distribution of traffic and improve the scalability and reliability of online services.

In these methods what happens when new server is added, our hashing would change? And in that case how does it work?

When a new server is added to a pool in a load-balanced system, how it affects the distribution of requests depends on the load balancing method used, especially in methods involving hashing. Let's focus on how this scenario affects hashing-based methods and how these methods cope with the addition of new servers

1. Consistent Hashing
Consistent hashing is designed to minimize the impact when servers are added or removed. In a consistent hashing setup, each server and request is assigned a hash value, which places them on a conceptual ring. When a new server is added:

The server is assigned a position on the ring based on its hash value.
Only the requests that map to the hash values between the new server and its immediate predecessor on the ring need to be redirected to the new server.
The majority of requests continue to be handled by the same servers as before, minimizing disruption.
This method greatly reduces the amount of reshuffling of requests, compared to traditional hashing methods where adding a new server would significantly change the distribution.

2. Rendezvous Hashing (Highest Random Weight, HRW)
In HRW hashing, when a new server is added:

Each request calculates its hash with every server and selects the server with the highest calculated hash value.
The addition of a new server might change the server selected for some of the requests, as the new server might compute a higher hash value for certain requests than the existing servers.
Only a portion of the requests will be redirected to the new server, similar to consistent hashing, minimizing the overall impact on the distribution.

3. Dynamic Hashing
Dynamic hashing methods are designed to adapt to changes, including the addition of new servers. The specifics of how the system adapts can vary:

Some dynamic hashing methods might redistribute some portion of the requests to include the new server.
They may adjust their parameters or the hash function itself to accommodate the new server while trying to keep disruption to a minimum.

General Impact and Mitigation Strategies

For methods where the hash space is directly tied to the number of servers (notably not consistent or HRW hashing), adding a new server would indeed change the hash distribution, potentially leading to a significant reshuffling of requests among servers. This can lead to what's called a "cache stampede," where many or all cache entries are invalidated and need to be recreated, putting a high load on the backend servers.

To mitigate such impacts, systems often implement strategies such as:

Gradual migration: Slowly redirecting traffic to the new server to avoid sudden changes.
Session persistence techniques: Using cookies or other mechanisms to ensure that a user session is not disrupted even if its requests are now handled by a different server.
Pre-warming: Loading the new server with necessary data and gradually increasing its load before it starts serving a full share of traffic.
Overall, the choice of load balancing method and the strategy for adding new servers depends on the specific requirements and constraints of the application, including considerations for session persistence, distribution fairness, and the acceptable level of disruption.