Load balancers are critical to the infrastructure of most applications deployed on premises and in the cloud. Whether the applications need scaling up or scaling out, load balancers play a key role in efficiently routing traffic to make best use of the available infrastructure while ensuring a fast and reliable experience for end users.

Load balancers distribute traffic based on specific rules and conditions captured in a load balancing algorithm.These algorithms encapsulate the logic for routing or distributing loads, making your choice critical as they directly influence your application's performance and user experience.

This article compares six widely used load balancing algorithms to help you choose the best one for your use case. Each algorithm is reviewed based on the following factors:

- Distribution methodology (rules and conditions for load distribution)
- Performance
- Scalability and flexibility
- Resource use
- Complexity and maintenance
- Affinity and session persistence
- Fault tolerance and recovery
- Suitability for specific applications

Static Load Balancing Algorithms

There are two broad categories of load balancing algorithms: static and dynamic. Static algorithms don't depend on external feedback to distribute the load, while dynamic algorithms do. The first three load balancing algorithms in this comparison are static.

Round Robin

The round robin load balancing algorithm evenly distributes traffic among servers in a cyclical and rotating fashion. It's stateless, meaning it's not reliant upon any previous states, IPs or external dependencies. Its uncomplicated distribution logic makes it simple to understand and implement.

This simplicity allows for easy setup and low operational overhead. However, it also precludes the ability to have finer control over how your traffic gets distributed.

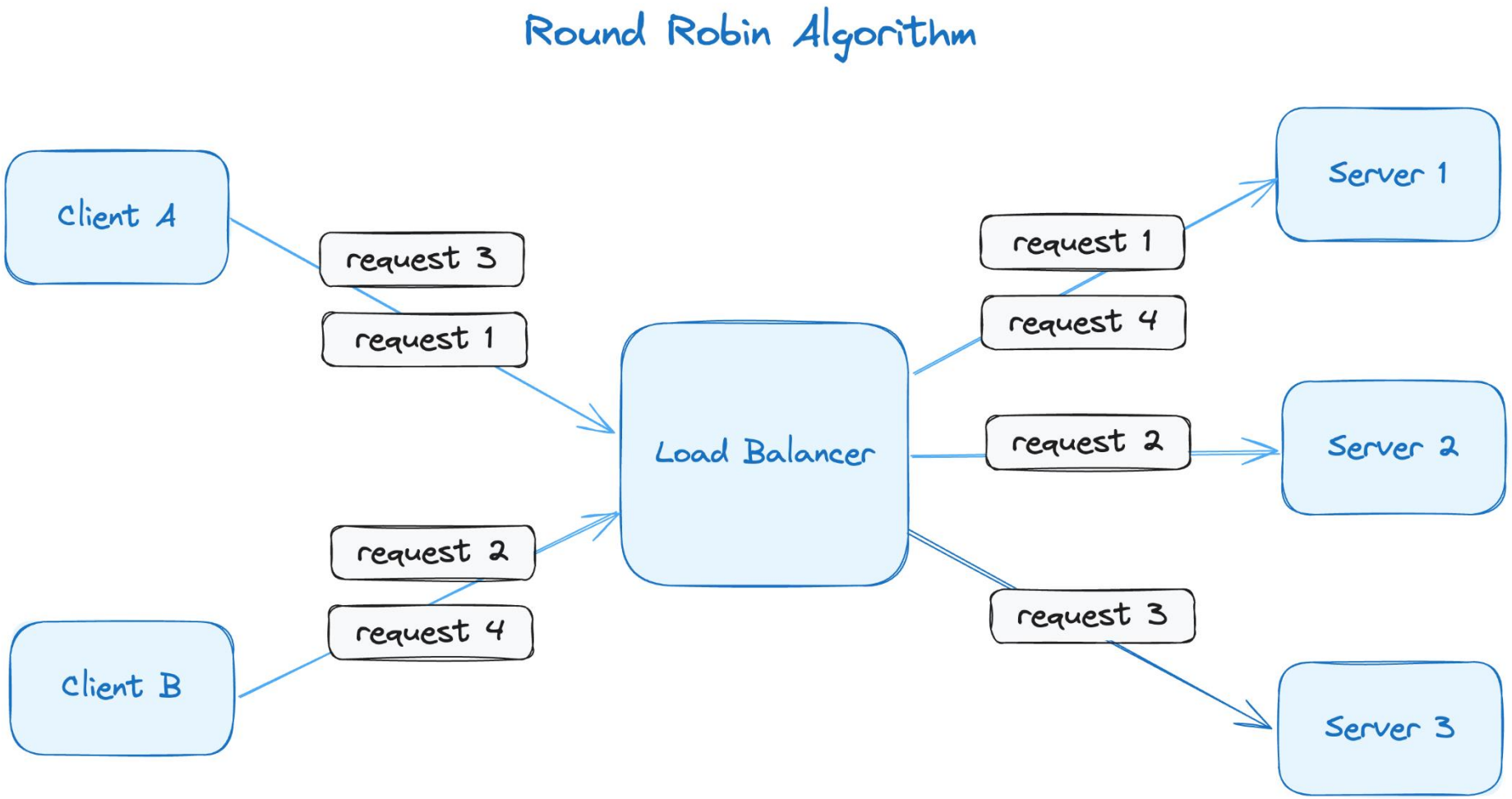

Diagram by Kovid Rathee

In this image, requests 1, 2 and 3 are forwarded to servers 1, 2 and 3 in sequential order. When the load balancer gets to request 4, it repeats the cycle and sends the fourth request to server 1. Even if the servers have different capacities, they'll get an equal proportion of requests.

More on load balancing:

- How Load Balancers Differ From Reverse Proxies, and When to Use Each
- What Load Balancers Do at Three Different Layers of the OSI Stack
- How API Gateways Differ from Load Balancers and Which to Use
- How to Speed Up and Secure Your Apps Using DNS Load Balancing

The core feature of the round robin algorithm is it distributes traffic evenly across servers, making it ideal for scenarios where you have identical server capacities (CPU, memory, etc.) in your server pool. It's not intended to be efficient when scaling up a subset of your server pool. Scaling out, however, is not a problem and usually doesn't require major changes to the load balancer configuration.

Disadvantages

One disadvantage of the round robin algorithm is that it cannot make decisions based on session information (that is, it can't support session affinity and persistence). This means requests from the same client session might go to different servers, which isn't suitable for some scenarios (more on that in the next section).

The round robin algorithm also has minimal fault tolerance and recovery aspects. Other than simply checking whether the target machines are available, your options are usually limited.

In addition, this algorithm doesn't consider any other external variables before distributing load. For example, the purpose of having a CDN is to accelerate content delivery, but what if there's a server that is more proximate to a user than another server? To address this issue, variations on the round robin algorithm have been created, including the weighted round robin algorithm.

Use Cases

The round robin load balancing algorithm is useful for content delivery networks (CDNs), API gateways, database read replicas and DNS servers, to name a few.

For example, it's recommended to split reads and writes in an application so that they can be scaled separately. This involves creating read replicas, which facilitate horizontal scaling for read operations and are typically of similar size. Routing random read requests to these replicas is efficient and incurs minimal overhead.

None of these scenarios require the load balancer to be stateful, and almost all rely more on scaling out than scaling up.

Weighted Round Robin

Unlike the round robin algorithm, the weighted round robin load balancing algorithm allows you to assign a weight to each server. That means servers with higher weights receive more requests than those with lower weights.

While this algorithm slightly increases the overhead of load distribution decision-making, it allows you to distribute the load more in line with what the servers can handle, which helps optimize your resources.

Weighted Round Robin Algorithm

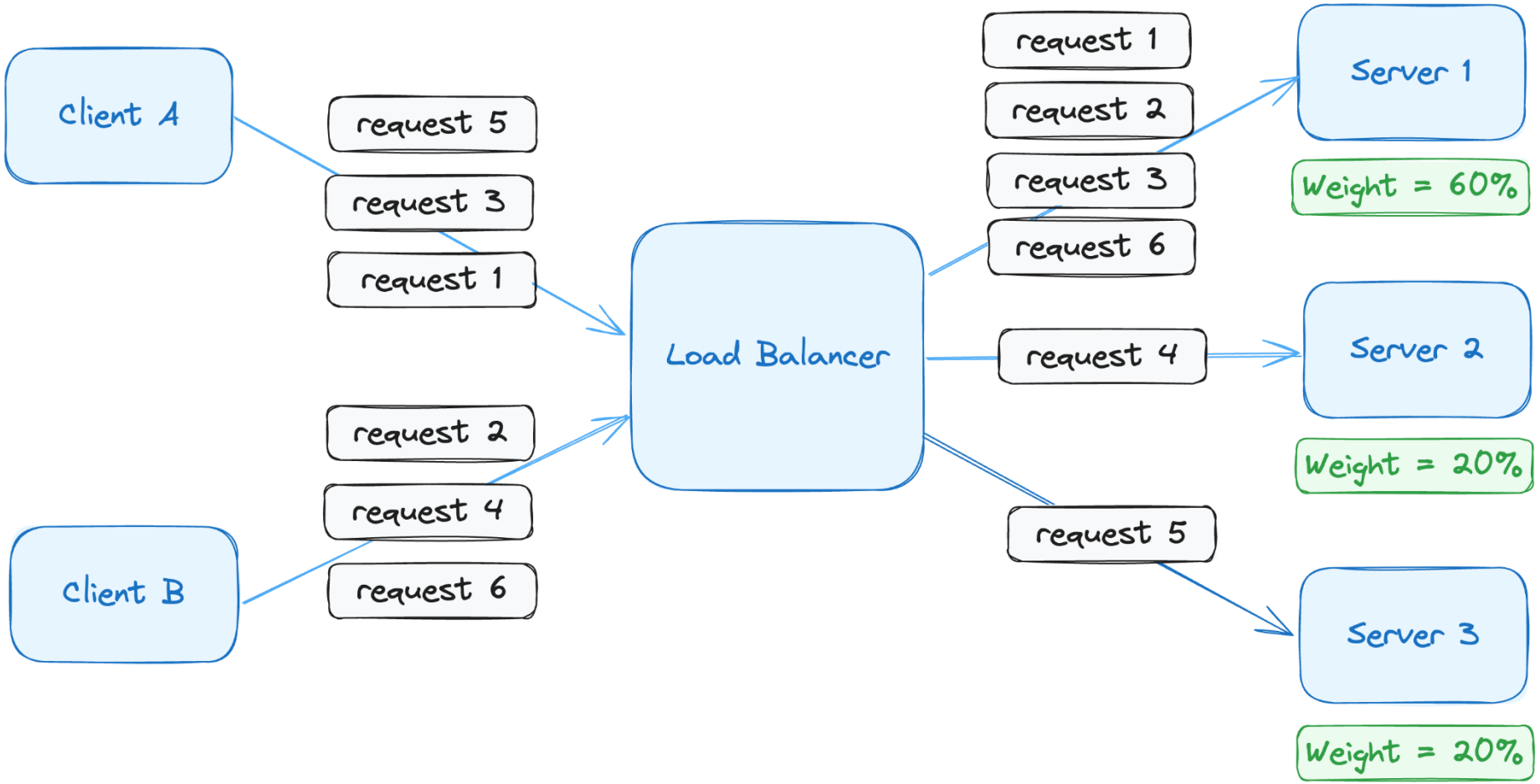

Diagram by Kovid Rathee

In this example, server 1 has three times greater capacity than servers 2 or 3. The pool of servers can address five requests in one cycle, where server 1 gets requests 1, 2 and 3 while server 2 and server 3 get requests 4 and 5. When the cycle repeats, server 1 gets the next request (request 6).

Disadvantages

Like the traditional round robin algorithm, this static algorithm doesn't take into account any external factors, such as server conditions, connection count or resource utilization. This means that the weights assigned to each server must be manually configured and don't change automatically if you scale up one of the servers in the pool, which adds a layer of manual upkeep. In addition, it can't support session affinity and persistence.

Despite the overhead, the weighted round robin algorithm allows the load balancer to distribute the load in line with many real-world scenarios where not all servers have the same capacity of CPU, cache or memory. While highly scalable, the algorithm requires a fair amount of manual intervention to manage server weights.

Use Cases

The use cases for this algorithm are very similar to the use cases for the round robin algorithm, but with weighted round robin, you can distribute the load for a CDN among edge servers with different capacities and read requests among database replicas of different capacities.

IP Hash

An essential criterion for load distribution is session affinity and persistence, which refers to the ability to send requests from the client session to the server where the client connection was first established. This isn't possible with either of the round robin algorithms.

The IP hash load balancing algorithm allows you to use the hashed value of the client's IP address to decide which server fulfills the request, thus uniformly mapping client sessions to servers and making the number of requests within the session irrelevant to the algorithm.

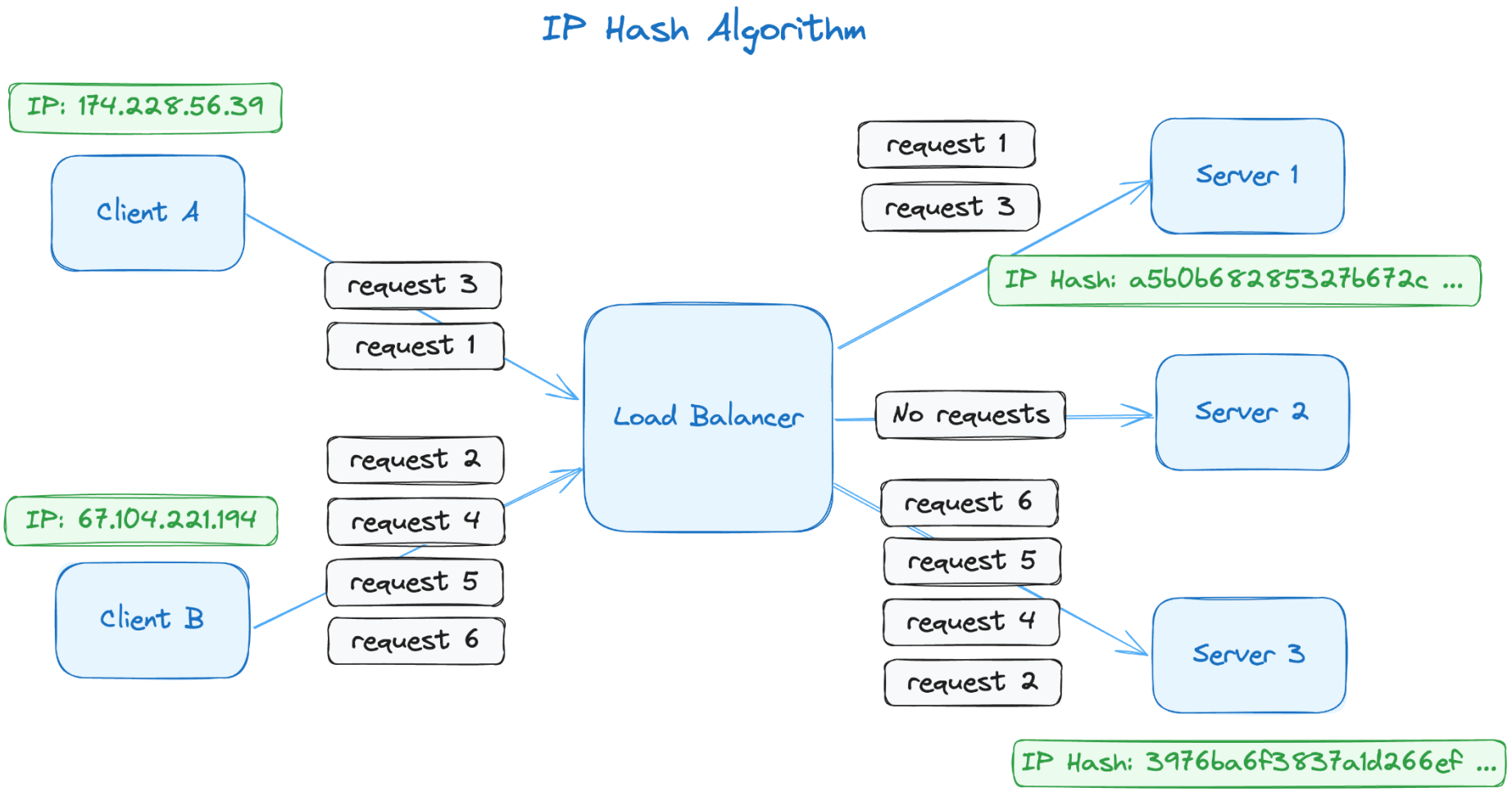

Diagram by Kovid Rathee

This image depicts two client sessions: one from the IP address 174.228.56.39 and one from 67.104.221.194. Since the IP hash value is assigned to a server, all requests from that IP are sent to the same server, and server 2 doesn't receive any requests.

Disadvantages

The IP hash algorithm struggles in highly dynamic cloud environments that frequently scale in and out based on demand, as this induces significant overhead in the load distribution process.

Another issue with the IP hash algorithm is that client requests can be highly inconsistent; some clients might send hundreds of requests while others only send a few. This disparity of traffic from different clients can undermine the effectiveness of the algorithm.

Use Cases

A lot of applications on the internet require you to maintain state across multiple requests from the same client, such as social media platforms, banking platforms, e-commerce websites or real-time communication applications. The IP hash algorithm works well for these types of applications since session persistence is core to their functionality.

This algorithm also has some fault tolerance and recovery features. It allows the load balancer to continue serving requests from other servers in case of a failure of the server that was handling the client's request from a particular IP address.

Dynamic Load Balancing Algorithms

Now that you've looked at three static load balancing algorithms, let's take a look at a few dynamic load balancing algorithms.

Least Connections

One of the most widely used dynamic load balancing algorithms is the least connections algorithm, which constantly monitors existing server connections and distributes incoming requests based on the server with the fewest connections. This enables the load balancer to conduct fair and sustainable load distribution without adding unnecessary pressure on a single server.

Least Connections Algorithm

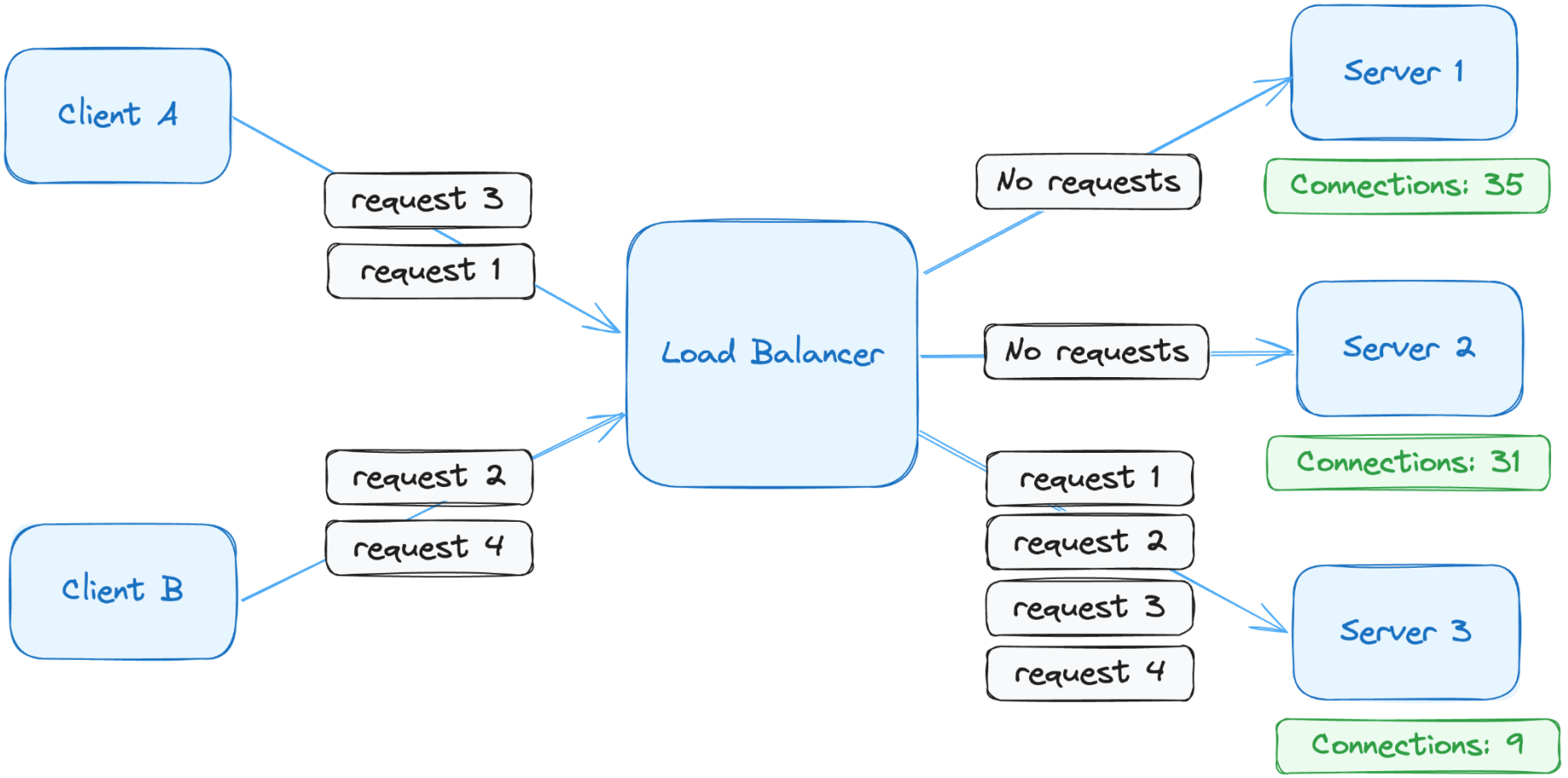

Diagram by Kovid Rathee

As you can see, the connection count for each server is included in this diagram (35, 31 and 9). Following the least connections principle, any new requests will be forwarded to server 3. In this scenario, request 1 from client A gets forwarded to server 3, and the number of connections for server 3 will rise from 9 to 10. However, because it still has the lowest number of connections, the next request will also go to server 3, making the connection count 11.

This algorithm is highly effective when your application has many connections with a variable connection time. The least connections algorithm can adapt whenever a session gets terminated or a new one is created. This algorithm uses the connection count maintained by the load balancer to ensure equal and efficient distribution of requests to servers. It's fault-tolerant because it can redirect traffic based on the server's response to active connections. If one of the servers is not responding with an active connection count, the algorithm will direct the load balancer to redirect the request to another server.

Disadvantages

The least connections algorithm doesn't support session affinity and persistence unless combined with a session persistence method, like prioritizing requests from a specific session to a particular server.

For example, you'd want a banking application customer to have a persistent session so that all their requests during the banking session go to the same server, ensuring a seamless and optimal banking experience. The least connections algorithm has no inherent properties that allow you to maintain stickiness. However, you can create a duration-based sticky session to achieve that. This has disadvantages, as sticky sessions could overload a single server while the others remain underutilized.

Use Cases

Well-implemented database servers usually limit the number of connections, as every connection takes up memory and CPU, even if it's stale. Database connections can be of varying active times; some sessions last only for a single query, while others might go on for hundreds of SQL queries that last hours. This makes the least connections algorithm a good option for balancing database load.

Least Response Time

The least response time algorithm tracks the server response times to fulfill requests. Some servers might have higher response times, depending on various server and network conditions. This algorithm aims to reduce the load distributed to the servers with high response times and concentrate the load on the servers with low response times.

Least Response Time Algorithm

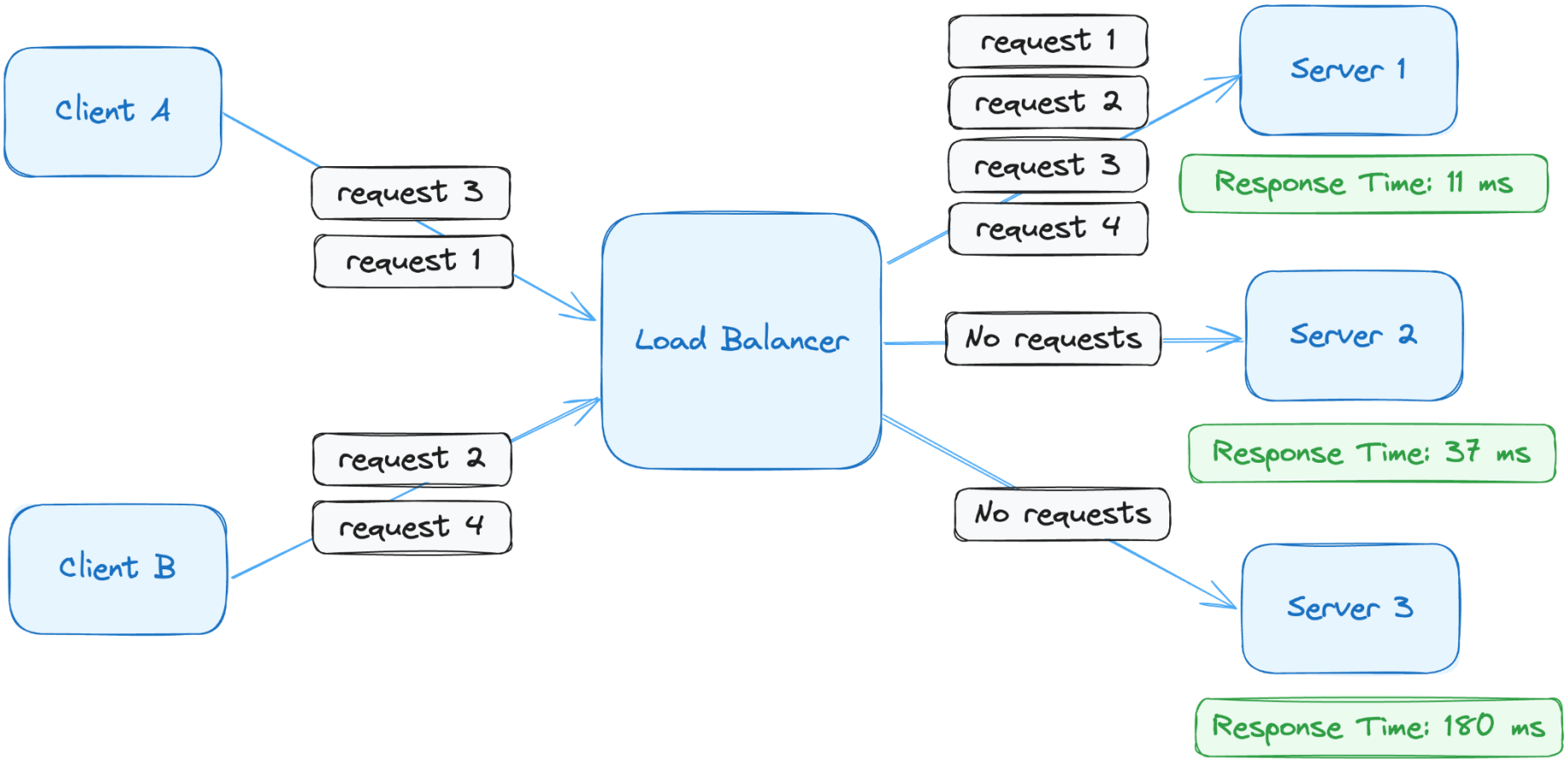

Diagram by Kovid Rathee

Here, servers 1, 2 and 3 have response times of 11 ms, 37 ms and 180 ms, respectively. That means all requests will be directed to server 1, assuming that the response time, usually a rolling average, remains the lowest among the three servers.

Disadvantages

While this algorithm improves performance by adapting to changing conditions, ensuring server resource utilization and allowing for automatic failovers, there's a risk of overloading some servers based on decisions that aren't valid anymore. This often happens in highly dynamic environments where server conditions vary significantly in a short period. For instance, a high-engagement e-commerce website might get tremendous traffic because of a flash sale. The incoming traffic might change server usage differently than expected due to session lengths, network latency and order placement queueing. In such conditions, getting a low-latency update on server resource usage becomes essential for making efficient routing decisions.

It's also difficult to use the least response time algorithm when dealing with a global distribution of loads because it becomes increasingly difficult to accurately measure and compare responses across geographies, especially where distances are larger.

Use Cases

The least response time algorithm is useful for distributing load for low-latency and real-time applications, such as online gaming, video streaming and high-frequency trading platforms. However, keep in mind that this comes at the additional cost of constantly monitoring response times.

Resource Based

The resource-based algorithm takes into account the real-time resource usage metrics from various servers in the pool. These usage metrics represent CPU usage, memory usage or a combination of both. This algorithm is often more resource-intensive than the other dynamic algorithms because of the added load that comes with tracking utilization metrics continuously.

Resource-Based Algorithm

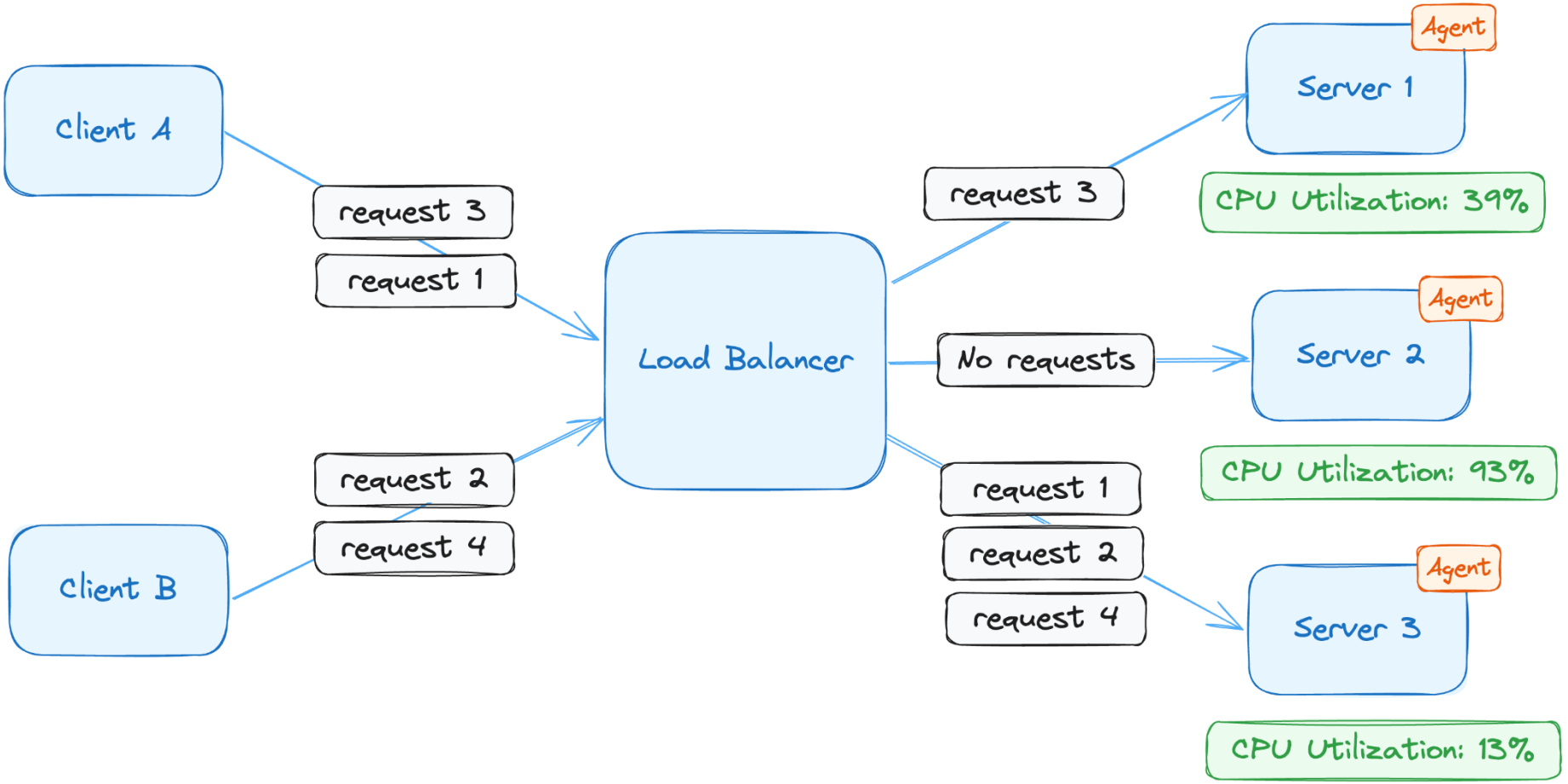

Diagram by Kovid Rathee

This diagram shows that an agent has been installed on all three servers that expose CPU utilization data to the load balancer. The load balancer uses the utilization data to make routing decisions. As server 3's CPU utilization is the lowest, requests 1 and 2 are routed to it. However, after request 2, the CPU utilization isn't the lowest anymore; server 1 now has the lowest CPU utilization, which is where request 3 is directed, and the process continues.

The resource-based algorithm is easy to scale and can be used in extremely large deployments. This algorithm also ensures that the load balancing is fault-tolerant, as it takes the availability and utilization of servers into consideration.

Disadvantages

By default, the resource-based algorithm doesn't allow you to maintain session affinity and persistence, but you can integrate it with session persistence methodologies to do so. Some of those session persistence methodologies are custom HTTP headers, SSL session IDs and session-based cookies.

Use Cases

To get this algorithm working, you need agents installed on the servers or other monitoring tools that connect to the servers and send you CPU and memory utilization data frequently. In both cases, there's a trade-off regarding routing overhead and complexity. Despite the trade-off, this algorithm is efficient when it comes to optimizing server resources and works well with applications that require highly dynamic environments and high-performance computing.

Conclusion

In this article, you learned about six widely used load balancing algorithms, both static and dynamic. If you're looking for simplicity, round robin and weighted round robin are ideal. To dynamically adapt to server conditions, consider using the least connections and least response time algorithms. For maximum resource utilization, the resource-based algorithm is recommended. However, none of these algorithms except IP hash are good with session affinity and persistence, so you'll need to integrate these with other algorithms to take care of session affinity. If you need session persistence in your application, consider using the IP hash load balancing algorithm.

If you need a high-performance, fast and secure network to deploy your application globally, dedicated cloud provides full control of your network configuration and server hardware and software. You decide exactly how packets get routed using direct cloud-to-cloud or colo-to-cloud private networking and direct Layer 2 and 3 connections, allowing you to build high-performance Network-as-a-Service solutions globally.

The features include Load Balancer-as-a-Service, a powerful load balancing solution that uses the global network backbone. External traffic directed to your dedicated cloud compute clusters behind a load balancer gets transported through the global backbone, ensuring your clients' requests arrive at the most suitable location on your network, wherever in the world that may be. This ensures optimal performance and efficient use of resources and makes an excellent choice for businesses seeking to optimize their web traffic management.