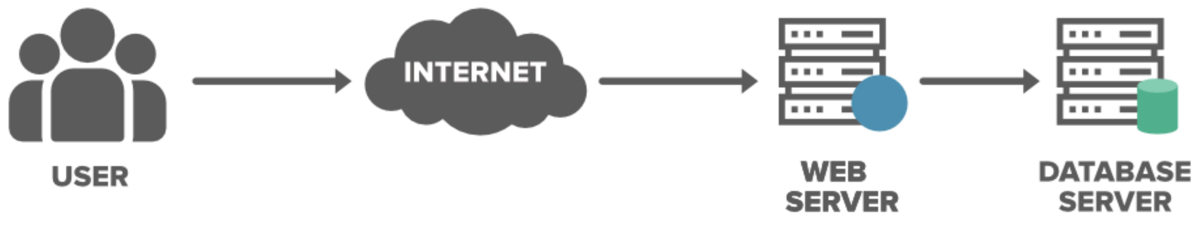
When a website becomes extremely popular, the traffic on that website increases, and the load on a single server also increases. The concurrent traffic overwhelms the single server and the website becomes slower for the users. In order to meet the request of these high volumes of data and to return the correct response in a fast and reliable manner, we need to scale the server. This can be done by adding more servers to the network and distributing all the requests across these servers. But....***who is going to decide which request should be routed to which server...???***
The answer is...**Load Balancer**.

What is a Load Balancer?

A load balancer works as a "traffic cop" sitting in front of your server and routing client requests across all servers. It simply distributes the set of requested operations (database write requests, cache queries) effectively across multiple servers and ensures that no single server bears too many requests that lead to degrading the overall performance of the application. A load balancer can be a physical device or a virtualized instance running on specialized hardware or a software process.
Consider a scenario where an application is running on a single server and the client connects to that server directly without load balancing. It will look something like the one below

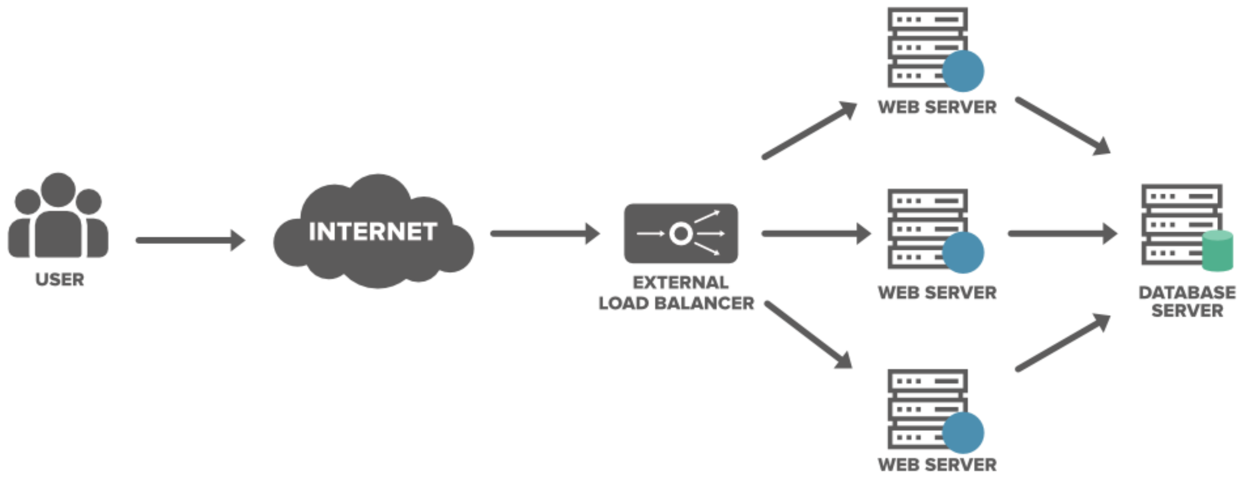


Problem
We need to discuss the two main problems with this model...

- **Single Point of Failure:** If the server goes down or something happens to the server the whole application will be interrupted and it will become unavailable for the users for a certain period. It will create a bad experience for users which is unacceptable for service providers.
- **Overloaded Servers:** There will be a limitation on the number of requests that a web server can handle. If the business grows and the number of requests increases the server will be overloaded. To solve the increasing number of requests we need to add a few more servers and we need to distribute the requests to the cluster of servers.

Solution

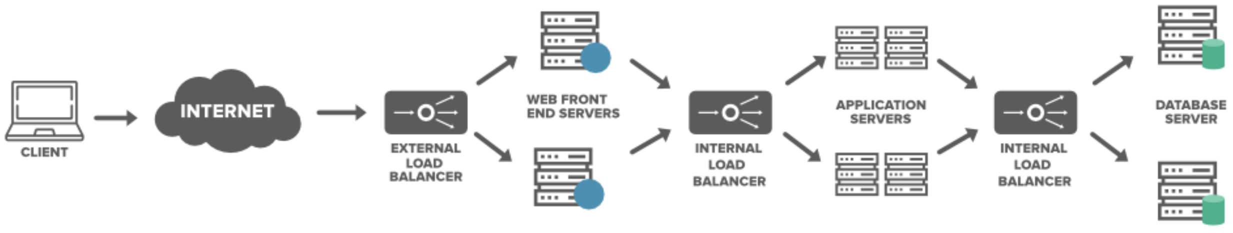
To solve the above issue and to distribute the number of requests we can add a load balancer in front of the web servers and allow our services to handle any number of requests by adding any number of web servers in the network. We can spread the request across multiple servers. For some reason, if one of the servers goes offline the service will be continued. Also, the latency on each request will go down because each server is not bottle-necked on RAM/Disk/CPU anymore.



- Load balancers minimize server response time and maximize throughput.
- Load balancer ensures high availability and reliability by sending requests only to online servers
- Load balancers do continuous health checks to monitor the server's capability of handling the request.
- Depending on the number of requests or demand load balancers add or remove the number of servers.

Where Are Load Balancers Typically Placed?

Below is the image where a load balancer can be placed…



- In between the client application/user and the server
- In between the server and the application/job servers
- In between the application servers and the cache servers
- In between the cache servers the database servers

Types of Load Balancers

We can achieve load balancing in three ways. These are…

1. Software Load Balancers in Clients

As the name suggests all the logic of load balancing resides on the client application (Eg. A mobile phone app). The client application will be provided with a list of web servers/application servers to interact with.

The application chooses the first one in the list and requests data from the server. If any failure occurs persistently (after a configurable number of retries) and the server becomes unavailable, it discards that server and chooses the other one from the list to continue the process. This is one of the cheapest ways to implement load balancing.

2. Software Load Balancers in Services

These load balancers are the pieces of software that receive a set of requests and redirect these requests according to a set of rules. This load balancer provides much more flexibility because it can be installed on any standard device (Ex: Windows or Linux machine). It is also less expensive because there is no need to purchase or maintain the physical device, unlike hardware load balancers. You can have the option to use the off-the-shelf software load balancer or you can write your custom software (Ex: load balance Active Directory Queries of Microsoft Office365) for load balancing.

3. Hardware Load Balancers

As the name suggests we use a physical appliance to distribute the traffic across the cluster of network servers. These load balancers are also known as Layer 4-7 Routers and these are capable of handling all kinds of HTTP, HTTPS, TCP, and UDP traffic. HLDs provide a virtual server address to the outside world. When a request comes from a client application, it forwards the connection to the most appropriate real server doing bi-directional network address translation (NAT). HLDs can handle a large volume of traffic but it comes with a hefty price tag and it also has limited flexibility.

HLDs keep doing the health checks on each server and ensure that each server is responding properly. If any of the servers don’t produce the desired response, it immediately stops sending the traffic to the servers. These load balancers are expensive to acquire and configure, that is the reason a lot of service providers use them only as the first entry point of user requests. Later the internal software load balancers are used to redirect the data behind the infrastructure wall.

Different Categories of Load Balancing

Generally, load balancers are grouped into three categories…

1. Layer 4 (L4) Load Balancer

In the OSI model layer 4 is the transport layer(TCP/SSL) where the routing decisions are made. Layer 4 load balancer is also referred to as **Network Load Balancing** and as the name suggests it leverages network layer information to make the routing decision for the traffic. It can control millions of requests per second and it handles all forms of TCP/UDP traffic. The decision will be based on the TCP or UDP ports that packets use along with their source and destination IP addresses. The L4 load balancer also performs Network Address Translation (NAT) on the request packet but it doesn’t inspect the actual contents of each packet. This category of load balancer maximizes the utilization and availability by distributing the traffic across IP addresses, switches, and routers.

2. Layer 7 (L7) Load Balancer

Layer 7 load balancer is also referred to as **Application Load Balancer** or **HTTP(S) Load Balancer**. It is one of the oldest forms of load balancing. In the OSI model, Layer 7 is the application layer (HTTP/HTTPS) where the routing decisions execute. Layer 7 adds content switching to load balancing and it uses information such as HTTP header, cookies, uniform resource identifier, SSL session ID, and HTML form data to decide the routing request across the servers.

3. Global Server Load Balancing (GSLB)

Today a lot of applications are hosted in cloud data centers in multiple geographic locations. This is the reason a lot of organizations are moving to a different load balancer that can deliver applications with greater reliability and lower latency to any device or location. With the significant change in the capability of the load balancers, GSLB fulfills these expectations of IT organizations. GSLB extends the capability of L4 and L7 servers in different geographic locations and distributes a large amount of traffic across multiple data centers efficiently. It also ensures a consistent experience for end-users when they are navigating multiple applications and services in a digital workspace.

Load Balancing Algorithms

We need a load balancing algorithm to decide which request should be redirected to which backend server. The different system uses different ways to select the servers from the load balancer. Companies use varieties of load balancing algorithm techniques depending on the configuration. Some of the common load balancing algorithms are given below:

1. Round Robin

Requests are distributed across the servers in a sequential or rotational manner. For example, the first request goes to the first server, the second one goes to the second server, the third request goes to the third server and it continues further for all the requests. It is easy to implement but it doesn't consider the load already on a server so there is a risk that one of the servers receives a lot of requests and becomes overloaded.

2. Weighted Round Robin

It is much similar to the round-robin technique. The only difference is, that each of the resources in a list is provided a weighted score. Depending on the weighted score the request is distributed to these servers. So in this method, some of the servers get a bigger share of the overall request.

3. Least Connection Method

In this method, the request will be directed to the server with the fewest number of requests or active connections. To do this load balancer needs to do some additional computing to identify the server with the least number of connections. This may be a little bit costlier compared to the round-robin method but the evaluation is based on the current load on the server. This algorithm is most useful when there is a huge number of persistent connections in the traffic unevenly distributed between the servers.

4. Least Response Time Method

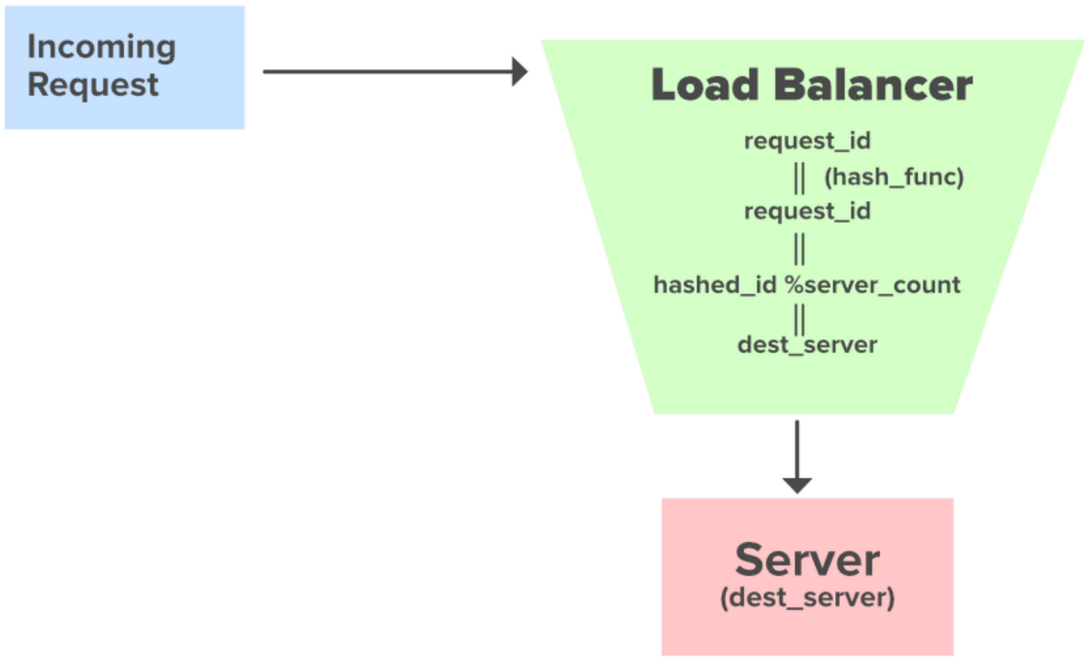
This technique is more sophisticated than the Least connection method. In this method, the request is forwarded to the server with the fewest active connections and the least average response time. The response time taken by the server represents the load on the server and the overall expected user experience.

5. Source IP Hash

In this method, the request is sent to the server based on the client's IP address. The IP address of the client and the receiving compute instance are computed with a cryptographic algorithm.

Routing requests through Load Balancer

A Load Balancer acts as a layer between the incoming requests coming from the user and multiple servers present in the system.We should avoid the scenarios where a single server is getting most of the requests while the rest of them are sitting idle. There are various Load Balancing Algorithms that ensure even distribution of requests across the servers.



Hashing Approach to direct requests from the Load Balancer

Suppose we have **server_count** as the Total number of servers present in the System and a **load_balancer** to distribute the requests among those servers. A request with an id **request_id** enters the system. Before reaching the destination server it is directed to the **load_balancer** from where it is further directed to its destination server.

When the request reaches the load balancer the hashing approach will provide us with the destination server where the request is to be directed.

Discussing the Approach :

- **request_id** : Request ID coming to get served
- **hash_func** : Evenly distributed Hash Function
- **hashed_id** : Hashed Request ID
- **server_count** : Number of Servers

```
class GFG {
        public static int hash_func(int request_id)
        {
                // Computing the hash request id
                int hashed_id = 112;
                return hashed_id;
        }

        public static void route_request_to_server(int dest_server)
        {
                System.out.println("Routing request to the Server ID : " + dest_server);
        }

        public static int request_id = 23; // Incoming Request ID
        public static int server_count = 10; // Total Number of Servers

        public static void main(String args[])
        {
                int hashed_id = hash_func(request_id); // Hashing the incoming request id
                int dest_server = hashed_id % server_count; // Computing the destination
server id

                route_request_to_server(dest_server);
        }
}
```

Computing the Destination Server address :

If the value of **server_count** is 10 i.e we have ten servers with following server ids **server_id_0, server_id_1, ………, server_id_9**.

Suppose the value of **request_id** is 23

When this request reaches the Load Balancer the hash function **hash_func** hashes the value of the incoming request id.

- **hash_func(request_d)** = **hash_func(23)**

Suppose after Hashing the request_id is randomly hashed to a particular value.

- **hashed_id** = 112

In order to bring the hashed id in the range of the number of servers we can perform a modulo of the hashed id with the count of servers.

- **dest_server** = **hashed_id** % **server_count**
- **dest_server** = 112%10
- **dest_server** = 2

So we can route this request to Server **server_id_2**

In this way we can distribute all the requests coming to our Load Balancer evenly to all the servers. But is it an optimal approach? Yes it distributes the requests evenly but what if we need to increase the number of our servers. Increasing the server will change the destination servers of all the incoming requests. What if we were storing the cache related to that request in its destination server? Now as that request is no longer routed to the earlier server, our entire cache can go in trash probably. Think !

`Note for this drawback use gaurav's pie diagram example from video that he has on his youtube`
