liquidweb.com

Load Balancing Techniques and Optimizations

14-17 minutes

The hosting world's bread & butter solution for providing high availability and redundancy is load balancing. There are many different use cases for a Load Balancer (LB). It is important to know how to effectively manage your LB configuration so that it performs optimally in your environment. The proceeding article will review some of the common practices that, when adhered to, provide a smooth and seamless high availability website/application through the use of load balancing.

Prerequisites

This article will focus on load balancing concepts and uses typical web services (HTTP, HTTPS). These services are independent of the LB device itself so they can run on any combination of operating system and server software, e.g., Linux, Windows, Apache, Nginx, IIS, etc. Load balancing is not limited to just web services. Any type of traffic with a client/server relationship can take advantage of load balancing.

What is Load Balancing?

Load balancing is the practice of using a network device, called a Load Balancer (LB), to distribute traffic between a back-end cluster of servers, called nodes. These nodes are virtually identical, each running the same software, services, and configurations.

Broken nodes can be easily replaced by additional nodes which are also added to handle traffic growth over time. This prevents a single node from becoming overwhelmed as the LB uses specific load balancing algorithmic methods to determine which node handles subsequent requests.

Load balancing is virtually invisible to the end-user, operating behind the scenes allowing a farm of servers to function as a single service or application. Load balancing comprises the backbone of most high availability solutions due to its flexibility, redundancy, and extensibility.

OSI Layer Optimization

All load balancing occurs on one of two layers of The OSI Model. These layers allow for traffic balancing configurations based on different information that is contained in the network packet for that specific layer. The two layers involved in this process are the ***L4 Transport*** and ***L7 Application*** layer.

- **L4 - Transport Layer:** The fourth layer allows balancing rules based on transport protocols. Balancing traffic based on details like IP Address or TCP Port are provided within the L4 layer.
- **L7 - Application Layer:** The Application Layer provides many additional details that can be inspected from the packet for balancing rules. The L7 layer is where rules can be constructed based on information from HTTP Headers, SSL Session ID, HTML Form Data, Cookies, etc.

Rule of Thumb:

For efficiency use the L4 layer. L4 load balancing happens earlier in the transaction, so that traffic can be routed more quickly than L7. L7 load balancing has to inspect data from the application layer.

But, it is not always possible to rely on the L4 layer, and it will depend heavily on the services and back-end node configuration.

Load Balancing Algorithmic Methods

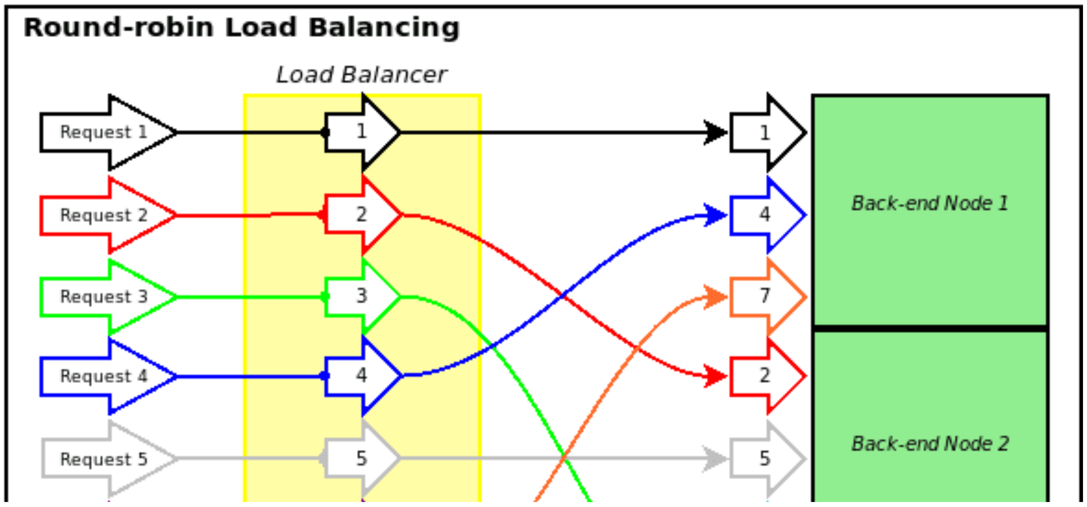
There are several common balancing algorithmic methods which can be used in a load-balanced configuration. Selecting the right algorithm for your infrastructure is critical to load balancing optimization. There is no general, one-size-fits-all, method for every situation. Choosing the correct approach will depend heavily upon the services, traffic, and software used in the load-balanced cluster. Below are some of the common methods used and their strengths/weaknesses from an optimization perspective.

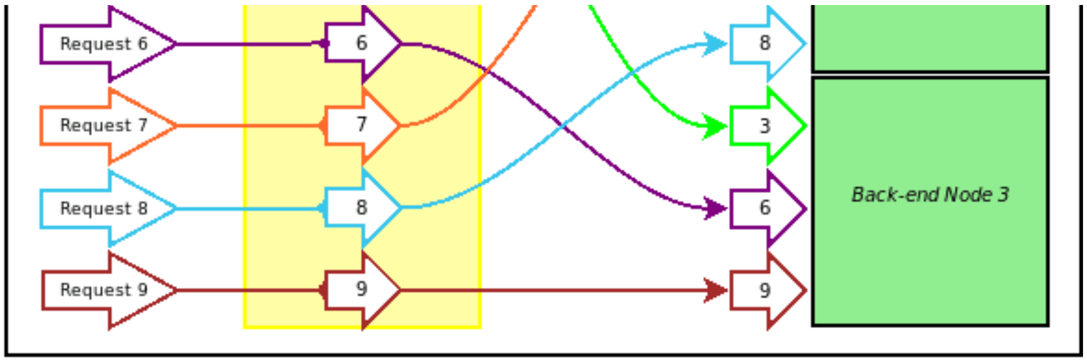
Round Robin

A simplistic load balancing approach. Traffic is sent to each node in series, one after the other, jumping back to the beginning of the list once the end is reached. (e.g., **Node 1** → **Node 2** → **Node 3** → **Repeat**)

Round Robin Pros & Cons

+ **Pros**: Lower memory and CPU footprint from the Load Balancer.
- **Cons**: Not adaptive, sends traffic to nodes without regard for distribution.



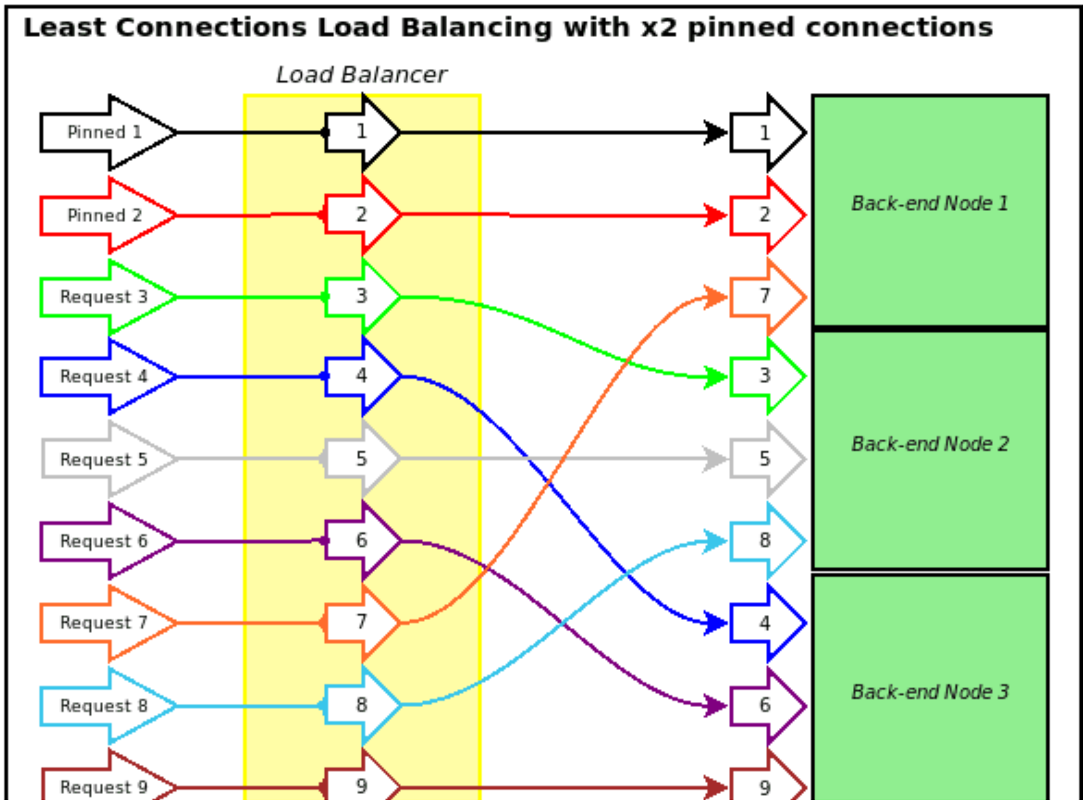


Least Connections

A smart balancing method which tracks each nodes network response time from health checks and routes new connections to the server with the quickest response time, regardless of any other factors.

Least Connections Pros & Cons

+ Pros: Adaptive, provided even distribution of workload among nodes.
- Cons: Requires larger memory and CPU footprint for connection tracking.

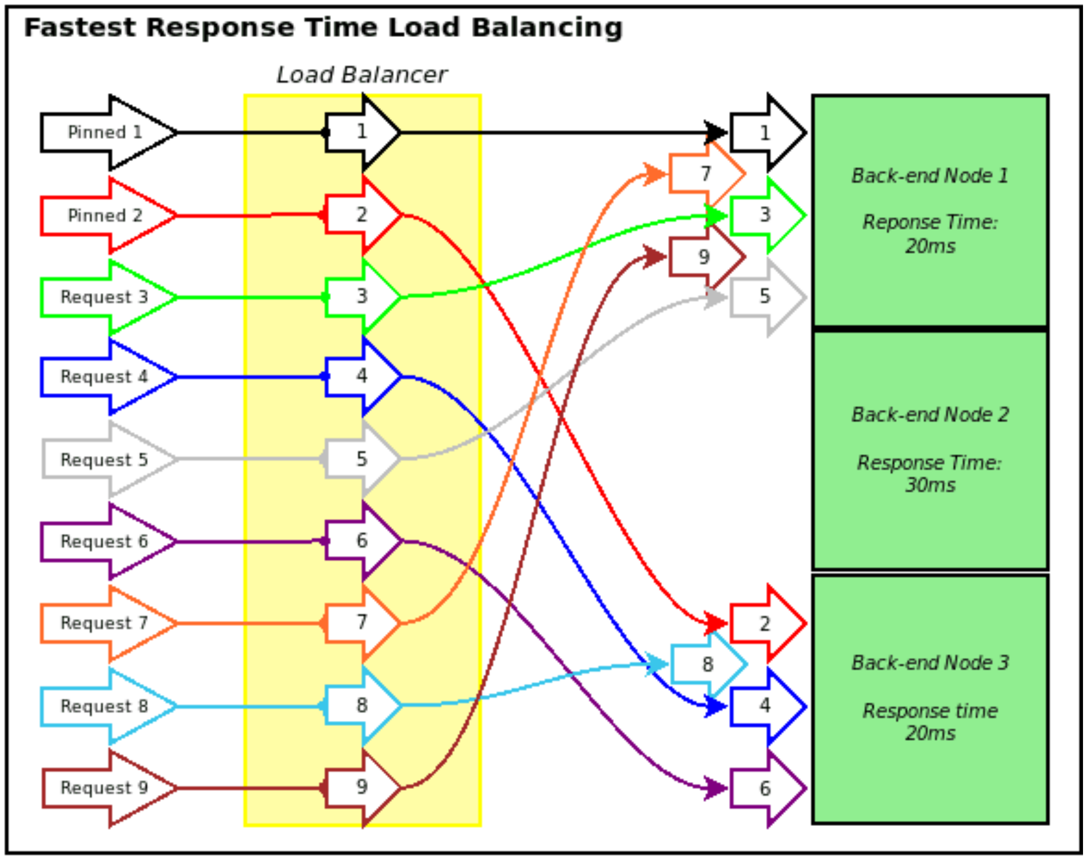


Fastest Response Time

A smart balancing method which tracks each nodes network response time from health checks and routes new connections to the server with the quickest response time, regardless of any other factors.

Fastest Response Time Pros & Cons

+ Pros: Adaptive, low memory, and CPU requirement.
- Cons: Limited workload distribution, connections and workload often fills up one node at a time.



Random Node

A niche method which is only used in very specific scenarios and is almost never appropriate from an optimization standpoint.

Random Node Pros & Cons

+ Pros: Requires a low memory and CPU footprint from load balancer device.
- Cons: Niche, only useful in specific scenarios. Unreliable distribution of workload.

Weighted Load Balancing

Another useful load balancing feature to consider is weighted nodes along with the balancing method. Weighted load balancing assigns additional connections to specific nodes over others dependent on each nodes weight value in the configuration. This feature is generally used when back-end nodes are not identical hardware or certain nodes receive special traffic beyond regular load-balanced traffic. Assigning a lower weight value in the form of a ratio to the weaker nodes allows them to participate in the workload at their individual capacity limits without getting overwhelmed. For example, a weight ratio of 2:2:1, would assign two connections to node 1 & node 2 for every 1 connection assigned to Node 3.

Choosing an Optimal Balancing Algorithm

When considering load balancing optimization, stick with the smarter balancing methods like **Least Connections**, or in some cases, **Fastest Response Time**.

- **Least Connections**, in particular, is very good at distributing workload between all the available hardware without focusing too much on a single node. However, this balancing method will require a sturdier load balancing device as the amount of nodes and traffic ramp up.
- **Fastest Response Time** can be a great alternative when the

hardware available for the load balancer is more limited. Each has its pros and cons as listed above, but both work well as optimized balancing methods for busy server farms.

Traffic Pinning & Session Persistence

Some configurations take advantage of pinning certain traffic to specific back-end nodes depending on the traffic. **Session Persistence** is a widely used form of **Traffic Pinning** as it results in routing requests to specific servers outside the load balancing method used.

Another common practice is assigning administration or upload traffic to a specific node that propagates the changes to the rest of the server farm. It is important to consider these connection types when designing or optimizing your Liquid Web server cluster.

Pro Tip:

Least Connections works well with configurations that rely on Traffic Pinning and/or Session Persistence. Since these features group connections to specific nodes independent of the load balancing algorithms. These connections are included in the tracking of the Least Connections balancing method which allows the load balancer to proportionately assign traffic to the nodes that are less busy while accounting for the existing pinned traffic.

Server Farm/Cluster Scope

Some important considerations should be accounted for when designing or upgrading your server cluster. A common mistake that gets overlooked is the size of the cluster. The number of nodes available in the cluster should not only be enough to handle regular workloads, It should account for both surges in workload as well as failures in the cluster. Ideally, a cluster should be large enough to remain fully operational, even when a surge in workload occurs **and** a critical event throws a node offline.

Rule of Thumb:

The cluster should be large enough to handle traffic surges plus at-least one extra node for redundancy.

Redundancy: Spare vs. Fail-over Node

Quite simply, redundancy is not effective without an extra node to handle the workload in the event that a critical issue occurs with another node. There are essentially two methods of handling redundancy in a load-balanced cluster.

Spare Nodes

An extra node in the cluster that is not needed for the cluster to function. It resided active in the cluster, handling workload just like any other node. However, it's there in case another node falls. The cluster can carry on without interruption, while the broken node is addressed as needed.

Spare Node Pros & Cons

+ Pros: Seamless redundancy, there is no downtime between a critical event and the promotion of the spare.
- **Cons:** Continuous use of the spare can lead to it failing along with another node when it is needed most.

Failover Nodes

An extra node, that has been configured and tested to work normally within the cluster. Once tested, the extra node is assigned as a fail-over node and taken out of the active configuration. The fail-over node, then sits on standby, waiting for a critical event to occur within the cluster. The fail-over node is then automatically activated in the cluster and start handling traffic.

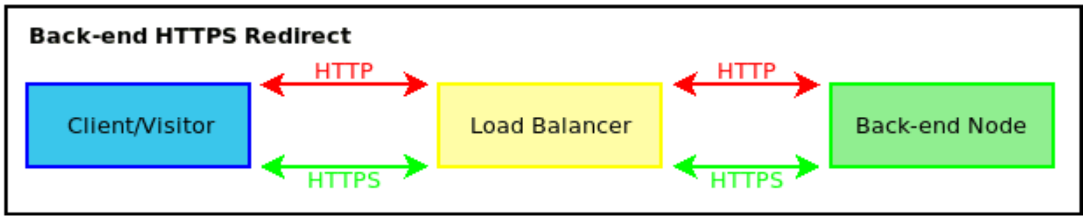
Failover Node Pros & Cons

+ **Pros:** Almost no risk of the fail-over node having hardware problems when it is needed.
- **Cons:** Delay between node failure and activation of the fail-over node.

Either method provides redundancy and keeps the application/site alive during critical times. A combination of both methods can be used as well to have the benefit of both options. It boils down to preference and cost. However, adhering to at least one of these methods should keep you running at an optimal state even during hard times.

Front-loaded Permanent Redirects

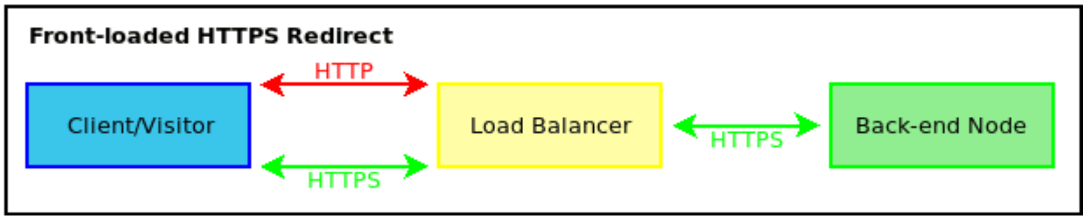
One way to improve response times of a load-balanced setup is to mitigate permanent redirects. This is done by moving them onto the load balancer device directly, instead of relying on the back-end nodes to issue a redirect to the client. This can reduce connection counts on both the load balancer and the back-end nodes themselves by eliminating one segment of the redirect process. The following illustrates this further when using a common practice of forcing HTTPS via redirects.



When a back-end node is configured to force HTTPS connections, the HTTP request comes into the load balancer, then is processed by the balancing algorithm and finally sent onto the necessary back-end node as normal. The back-end node then issues the redirect, which instructs the client to reconnect over the HTTPS protocol. The new HTTPS request is also balanced by the algorithm as needed and sent to a back-end node. The net processing result of this type of redirect is:

- x2 front-end requests (HTTP & HTTPS)
- x2 back-end requests
- x2 load balancing algorithmic checks

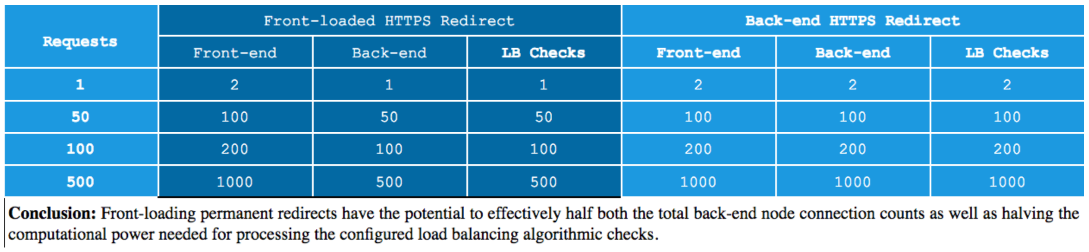
Now compare this to a front-loaded HTTPS redirect.



In the front-load configuration, the HTTP request hits the load balancer but is not processed by the balancing algorithm. Instead, it immediately issues the redirect back to the client. Then the client reconnects over HTTPS protocol, which then invokes the algorithm as needed and is sent onto the back-end node as required. The net processing result of this configuration is:

- x2 front-end requests (HTTP & HTTPS)
- x1 back-end request (HTTPS only)
- x1 load balancing algorithmic checks

This example only illustrates a single request. However, load balancers typically handle hundreds or even thousands of requests concurrently. It is easier to see the overall benefit of a change like this when scaling up the example:



Requests	Front-loaded HTTPS Redirect			Back-end HTTPS Redirect		
	Front-end	Back-end	LB Checks	Front-end	Back-end	LB Checks
1	2	1	1	2	2	2
50	100	50	50	100	100	100
100	200	100	100	200	200	200
500	1000	500	500	1000	1000	1000

Conclusion: Front-loading permanent redirects have the potential to effectively half both the total back-end node connection counts as well as halving the computational power needed for processing the configured load balancing algorithmic checks.

SSL Decryption

There are a handful of ways that load balancers are configured to handle SSL encrypted connections like HTTPS. Encrypted connections are more cumbersome on the load balancer device than non-encrypted connects. The process of validating the certificate chain and then decryption the content adds additional workload to every request handled by the load balancer. This can be mitigated depending on the configuration of the back-end server clusters and the applications/website's encryption requirements. The following are the common configuration scenarios and their pros/cons.

SSL Passthrough

The LB is configured to pass any encrypted connections through to the necessary back-end nodes. Decryption is handled by the back-end nodes only and not the LB device.

SSL Passthrough Pros & Cons

+ Pros: Simple configuration, high security, low CPU/memory footprint. Offers end-end-encryption.
- **Cons:** Limited to L4 Layer load balancing. Cannot perform packet inspection.

Decryption Only

The LB device itself will perform the necessary decryption. Traffic is then balanced to the back-end nodes after decryption. Due to security concerns, this configuration is only recommended when the back-end nodes and load balancer device have an isolated LAN to communicate through.

Decryption Only Pros & Cons

+Pros: Allows packet inspection for L7 Layer load balancing rules. Back-end nodes do not handle decryption.

- **Cons**: Any machines on the network can read decrypted back-end traffic.

Decryption + Re-encryption–

Decryption is handled by both the LB device and then re-encrypted and sent to the back-end node as needed. The back-end node also performs decryption providing full end-to-end encryption. This method is for high-security setups that also require L7 load balancing rules.

Decryption + Re-encryption Pros & Cons

+ Pros: Maximum security with end-to-end encryption. L7 load balancing rules & full packet inspection.
- Cons: Higher workload demand on the LB device. Encryption must be set up on both the front and back-end.

Conclusion

The load balancing optimization stance is the less workload the LB performs, the faster it responds to requests. This makes SSL Passthrough the ideal choice as it requires the least amount of work from the LB device. However, this is not always the practical solution for some applications/sites, particularly those that require L7 layer load balancing or other packet inspection needs. Decryption Only requires less workload when compared to Decryption + Re-encryption, but should only be leveraged when running a dedicated LB on a private LAN with the back-end nodes preventing outside entities from abusing the non-encrypted back-end traffic.

Why High Availability Matters -
And How You Can Achieve It Reliably and Affordably

And How You Can Achieve It Reliably and Affordably

Download Now

Give us a call at 800.580.4985, or open a chat or ticket with us to speak with one of our knowledgeable Solutions Team or an experienced Hosting Advisors today!