Published on **The O'Reilly Network** (http://www.oreillynet.com/)
http://www.oreillynet.com/pub/a/onjava/2001/09/26/load.html
See this if you're having trouble printing code examples

Load Balancing Web Applications

by Vivek Viswanathan
09/28/2001

This article offers an overview of several approaches to load balancing on Web application server clusters. A cluster is a group of servers running a Web application simultaneously, appearing to the world as if it were a single server. To balance server load, the system distributes requests to different nodes within the server cluster, with the goal of optimizing system performance. This results in higher availability and scalability -- necessities in an enterprise, Web-based application.

High availability can be defined as redundancy. If one server cannot handle a request, can other servers in the cluster handle it? In a highly available system, if a single Web server fails, then another server takes over, as transparently as possible, to process the request.

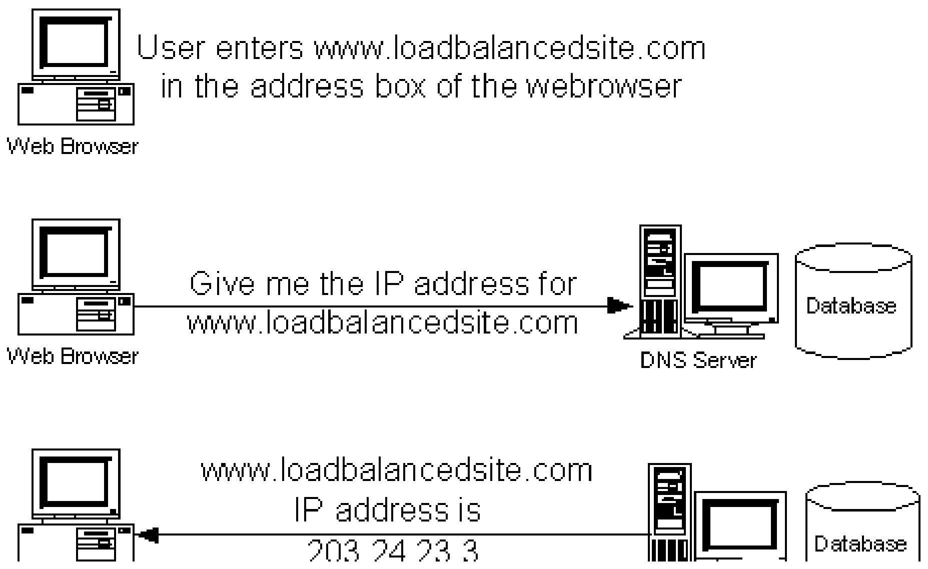
Scalability is an application's ability to support a growing number of users. If it takes an application 10 milliseconds(ms) to respond to one request, how long does it take to respond to 10,000 concurrent requests? Infinite scalability would allow it to respond to those in 10 ms; in the real world, it's somwhere between 10 ms and a logjam. Scalability is a measure of a range of factors, including the number of simultaneous users a cluster can support and the time it takes to process a request.

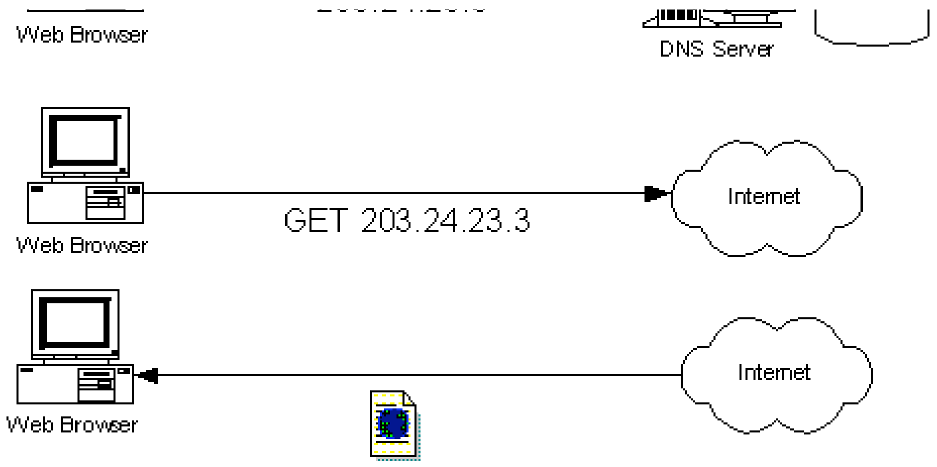
Of the many methods available to balance a server load, the main two are:

- DNS round robin
  and
- Hardware load balancers.

DNS Round Robin

As most readers of ONJava probably know, the Domain Name Server (DNS) database maps host names to their IP addresses.





When you enter a URL in a browser (say, `www.loadbalancedsite.com`), the browser sends a request to the DNS asking it to return the IP address of the site. This is called the DNS lookup. After the Web browser gets the IP address for that site, it contacts the site using the IP address, and displays the page for you.

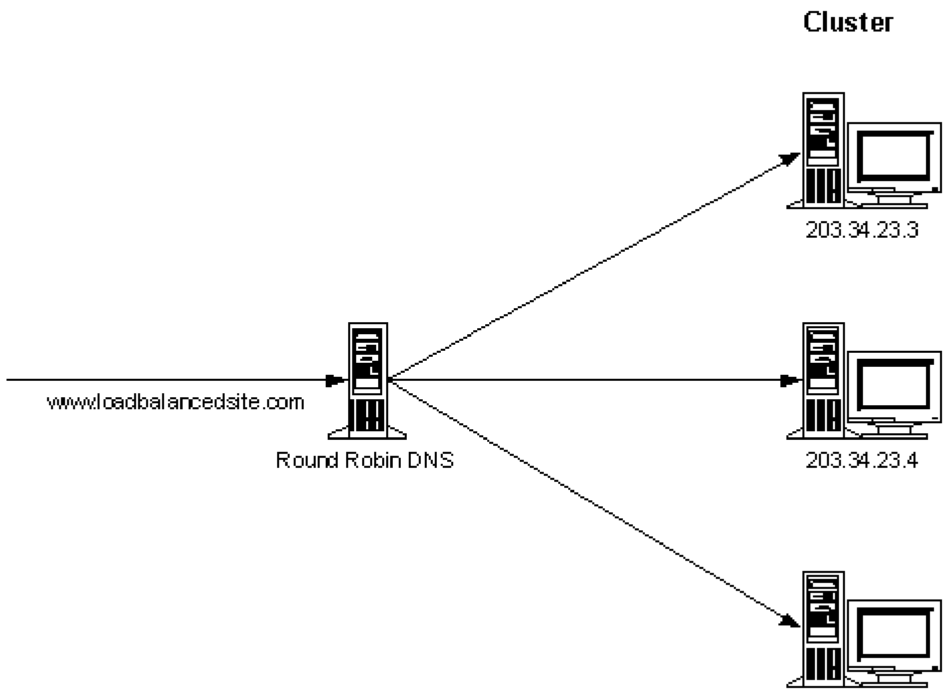
The DNS server generally contains a single IP address mapped to a particular site name. In our fictional example, our site www.loadbalancedsite.com maps to the IP address 203.24.23.3

To balance server loads using DNS, the DNS server maintains several different IP addresses for a site name. The multiple IP addresses represent the machines in the cluster, all of which map to the same single logical site name. Using our example, www.loadbalancedsite.com could be hosted on three machines in a cluster with the following IP addresses:

203.34.23.3
203.34.23.4
203.34.23.5

In this case, the DNS server contains the following mappings:

```
www.loadbalancedsite.com  203.34.23.3
www.loadbalancedsite.com  203.34.23.4
www.loadbalancedsite.com  203.34.23.5
```



```
203.34.23.5
```

When the first request arrives at the DNS server, it returns the IP address 203.34.23.3, the first machine. On the second request, it returns the second IP address: 203.34.23.4. And so on. On the fourth request, the first IP address is returned again.

Using the above DNS round robin, all of the requests to the particular site have been evenly distributed among all of the machines in the cluster. Therefore, with the DNS round robin method of load balancing, all of the nodes in the cluster are exposed to the net.

Advantages of DNS Round Robin

The main advantages of DNS round robin are that it's cheap and easy:

- **Inexpensive and easy to set up.** The system administrator only needs to make a few changes in the DNS server to support round robin, and many of the newer DNS servers already include support. It doesn't require any code change to the Web application; in fact, Web applications aren't aware of the load-balancing scheme in front of it.
- **Simplicity.** It does not require any networking experts to set up or debug the system in case a problem arises.

Disadvantages of DNS Round Robin

Two main disadvantages of this software-based method of load balancing are that it offers no real support for server affinity and doesn't support high availability.

- **No support for server affinity**. Server affinity is a load-balancing system's ability to manage a user's requests, either to a specific server or any server, depending on whether session information is maintained on the server or at an underlying, database level.

  Without server affinity, DNS round robin relies on one of three methods devised to maintain session control or user identity to requests coming in over HTTP, which is a stateless protocol.

  - cookies
  - hidden fields
  - URL rewriting

  When a user makes a first request, the Web server returns a text-based token uniquely identifying that user. Subsequent requests include this token using either cookies, URL rewriting, or hidden fields, allowing the server to appear to maintain a session between client and server. When a user establishes a session with one server, all subsequent requests usually go to the same server.

  The problem is that the browser caches that server's IP address. Once the cache expires, the browser makes another request to the DNS server for the IP address associated with the domain name. If the DNS server returns a differnt IP address, that of another server in the cluster, the session information is lost.

- **No support for high availability.** Consider a cluster of *n* nodes. If a node goes down, then every *n*th request to the DNS server directs you to the dead node. An advanced router solves this problem by checking nodes at regular intervals, detecting failed nodes and removing them from the list, so no requests go to them. However, the problem still exists if the node is up but the Web application running on the node goes down.

  Changes to the cluster take time to propagate through the rest of the Internet. One reason is that many large organizations -- ISPs, corporations, agencies -- cache their DNS requests to reduce network traffic and request time. When a user within these organizations makes a DNS request, it's checked against the cache's list of DNS names mapped to IP addresses. If it finds an entry, it returns the IP address to the user. If an entry is not found in its local cache, the ISP sends this DNS request to the DNS server and caches response.

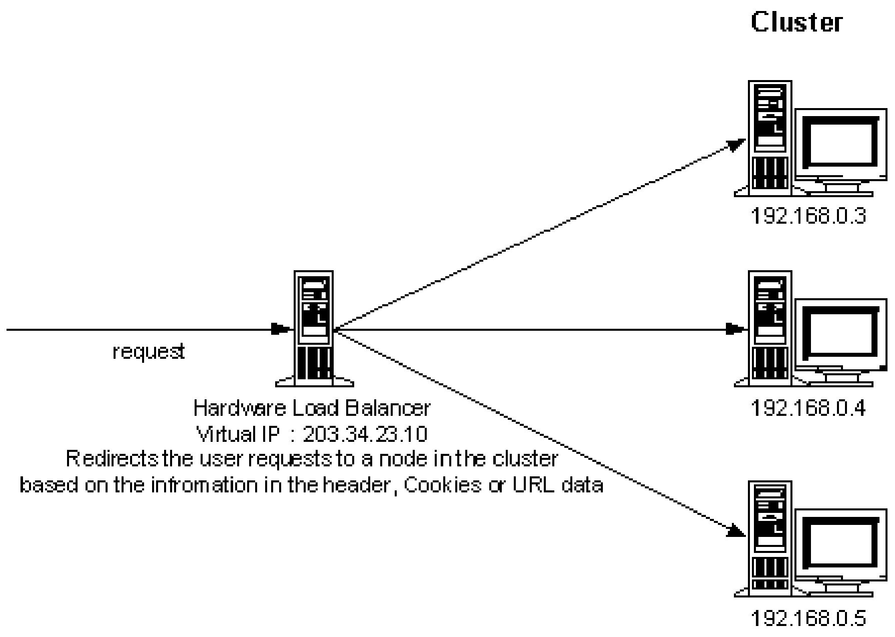
When a cached entry expires, the ISP updates its local database by contacting other DNS servers. When your list of servers changes, it can take a while for the cached entries on other organizations' networks to expire and look for the updated list of servers. During that period, a client can still attempt to hit the downed server node, if that client's ISP still has an entry pointing to it. In such a case, some users of that ISP couldn't access your site on their first attempt, even if your cluster has redundant servers up and running.

This is a bigger problem when removing a node than when adding one. When you drop a node, a user may be trying to hit a non-existing server. When you add one, that server may just be under-utilized until its IP address propogates to all the DNS servers.

Although this method tries to balance the number of users on each server, it doesn't necessarily balance the server load. Some users could demand a higher load of activity during their session than users on another server, and this methodology cannot guard against that inequity.

Hardware Load Balancers

Hardware load balancers solve many of the problems faced by the round robin software solution through virtual IP addresses. The load balancer shows a single (virtual) IP address to the outside world, which maps to the addresses of each machine in the cluster. So, in a way, the load balancer exposes the IP address of the entire cluster to the world.


Cluster
192.168.0.3
request
192.168.0.4
Hardware Load Balancer
Virtual IP : 203.34.23.10
Redirects the user requests to a node in the cluster
based on the infromation in the header, Cookies or URL data
192.168.0.5

When a request comes to the load balancer, it rewrites the request's header to point to other machines in the cluster. If a machine is removed from the cluster, the request doesn't run the risk of hitting a dead server, since all of the machines in the cluster appear to have the same IP address. This address remains the same even if a node in the cluster is down. Moreover, cached DNS entries around the Internet aren't a problem. When a response is returned, the client sees it coming from the hardware load balancer machine. In other words, the client is dealing with a single machine, the hardware load balancer.

Advantages of Hardware Load Balancers

- **Server affinity.** The hardware load balancer reads the cookies or URL readings on each request made by the client. Based on this information, it can rewrite the header information and send the request to the appropriate node in the cluster, where its session is maintained.

  Hardware load balancers can provide server affinity in HTTP communication, but not through a secure channel, such as HTTPS. In a secure channel, the messages are SSL-encrypted, and this prevents the load balancer from reading the session information.

- **High Availability Through Failover.** Failover happens when one node in a cluster cannot process a request and redirects it to another. There are two types of failover:

  - **Request Level Failover.** When one node in a cluster cannot process a request (often because it's down), it passes it along to another node.
  - **Transparent Session Failover.** When an invocation fails, it's transparently routed to another node in the cluster to complete the execution.

  Hardware load balancers provide request-level failover; when the load balancer detects that a particular node has gone down, it redirects all subsequent requests to that dead node to another active node in the cluster. However, any session information on the dead node will be lost when requests are redirected to a new node.

  Transparent session failover requires execution knowledge for a single process in a node, since the hardware load balancer can only detect network-level problems, not errors. In the execution process of a single node, hardware load balancers do not provide transparent session failover. To achieve transparent session failover, the nodes in the cluster must collaborate among each other and have something like a shared memory area or a common database where all the session data is stored. Therefore, if a node in the cluster has a problem, a session can continue in another node.

- **Metrics.** Since all requests to a Web application must pass through the load-balancing system, the system can determine the number of active sessions, the number of active sessions connected in any instance, response times, peak load times, the number of sessions during peak load, the number of sessions during minimum load, and more. All this audit information is used to fine tune the entire system for optimal performance.

Disadvantages of Hardware Load Balancers

The drawbacks to the hardware route are the costs, the complexity of setting up, and the vulnerability to a single point of failure. Since all requests pass through a single hardware load balancer, the failure of that piece of hardware sinks the entire site.

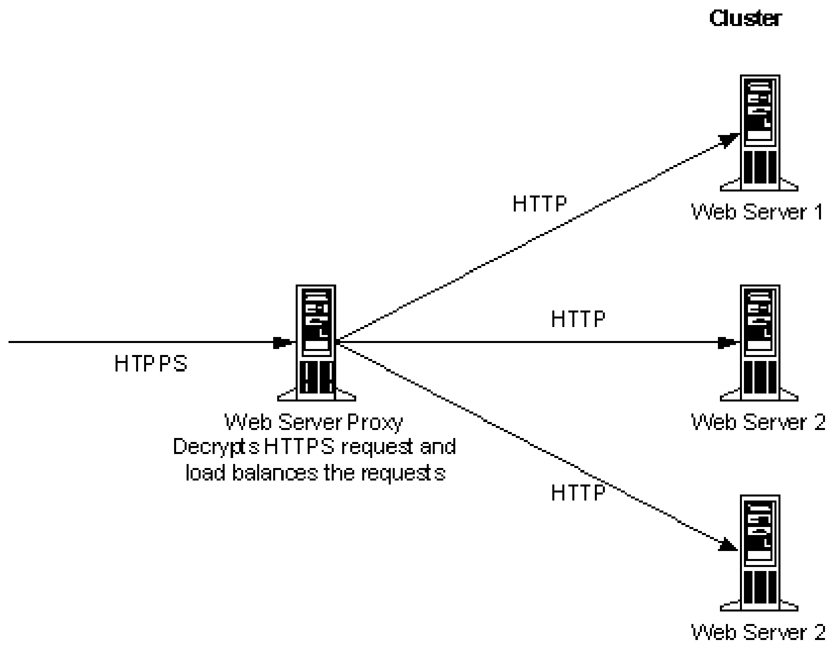
Load Balancing HTTPS Requests

As mentioned above, it's difficult to load balance and maintain session information of requests that come in over HTTPS, as they're encrypted. The hardware load balancer cannot redirect requests based on the information in the header, cookies, or URL readings. There are two options to solve this problem:

- Web server proxies
- Hardware SSL decoders.

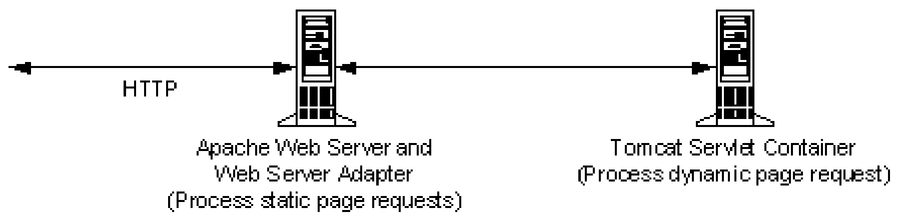
Implementing Web Server Proxies

A Web server proxy that sits in front of a cluster of Web servers takes all requests and decrypts them. Then it redirects them to the appropriate node, based on header information in the header, cookies, and URL readings.

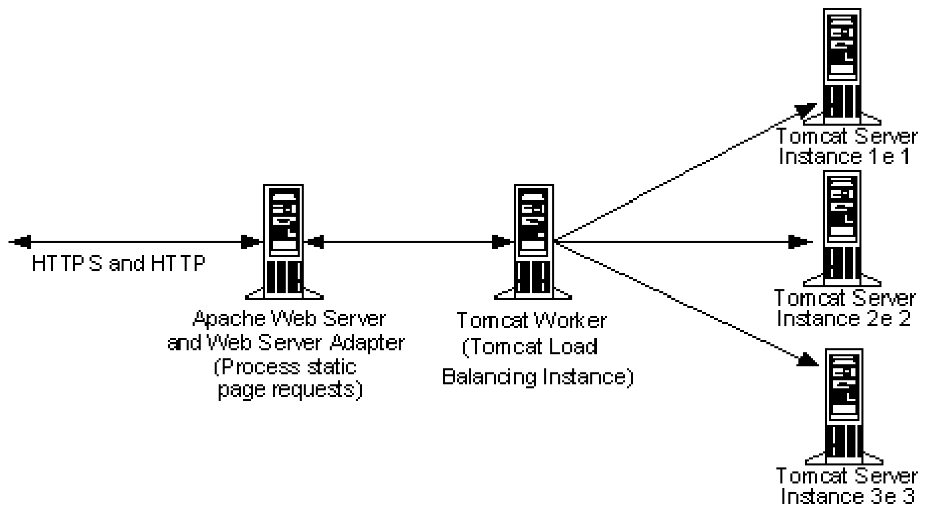


The advantages of Web server proxies are that they offer a way to get server affinity for SSL-encrypted messages, without any extra hardware. But extensive SSL processing puts an extra load on the proxy.

Apache and Tomcat. In many serving systems, Apache and Tomcat servers work together to handle all HTTP requests. Apache handles the request for static pages (including HTML, JPEG, and GIF files), while Tomcat handles requests for dynamic pages (JSPs or servlets). Tomcat servers can also handle static pages, but in combined systems, they're usually set up to handle dynamic requests.



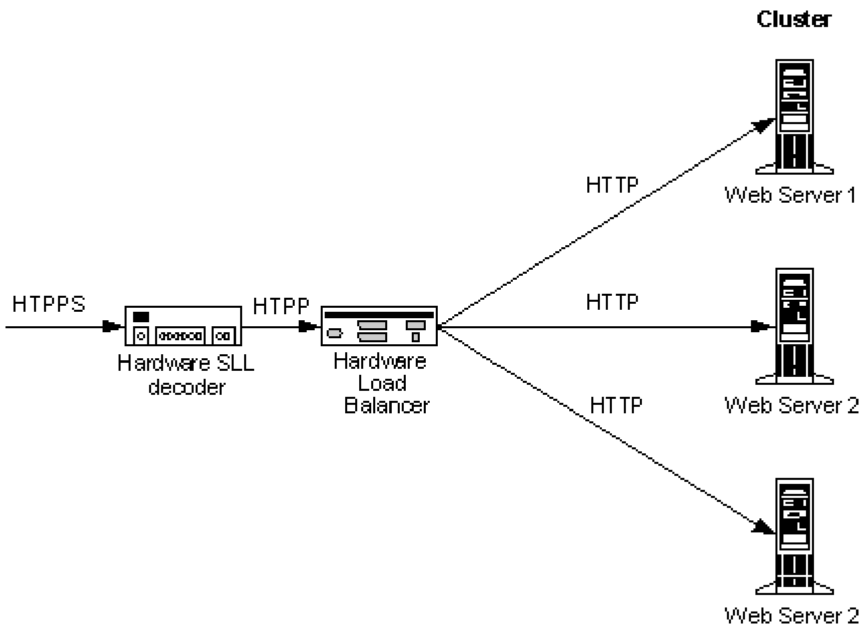
You can also configure Apache and Tomcat to handle HTTPS requests and to balance loads. To achieve this, you run multiple instances of Tomcat servers on one or more machines. If all of the Tomcat servers are running on one machine, they should be configured to listen on different ports. To implement load balancing, you create a special type of Tomcat instance, called a Tomcat Worker.


HTTP S and HTTP
Apache Web Server and Web Server Adapter (Process static page requests)
Tomcat Worker (Tomcat Load Balancing Instance)
Tomcat Server Instance 1e 1
Tomcat Server Instance 2e 2
Tomcat Server Instance 3e 3

As shown in the illustration, the Apache Web server receives HTTP and HTTPS requests from clients. If the request is HTTPS, the Apache Web server decrypts the request and sends it to a Web server adapter, which in turn sends the request to the Tomcat Worker, which contains a load-balancing algorithm. Similar to the Web server proxy, this algorithm balances the load among Tomcat instances.

Hardware SSL Decoder

Finally, we should mention that there are hardware devices capable of decoding SSL requests. A complete description of them is beyond the scope of this article, but briefly, they sit in front of the hardware load balancer, allowing it to decrypt information in cookies, headers and URLs.



These hardware SSL decoders are faster than Web server proxies and are highly scalable. But as with most hardware solutions, they cost more and are complicated to set up and configure.

Vivek Viswanathan is an enterprise web application developer, whose interests include clustering, performance and optimization.

Return to ONJava.com.