It’s 2am: Do You Know What Algorithm Your Load Balancer is Using?

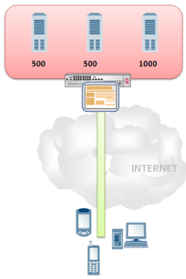
Lori MacVittie, 2010-05-01

The wrong load balancing algorithm can be detrimental to the performance and scalability of your web applications. When you're mixing and matching virtual or physical servers you need to take care with how you configure your Load balancer – and that includes cloud-based load balancing services.

Load balancers do not at this time, unsurprisingly, magically choose the right algorithm for distributing requests for a given environment. One of the nice things about a load balancing solution that comes replete with application-specific templates is that all the work required to determine the optimal configuration for the load balancer and its associated functionality (web application security, acceleration, optimization) has already been done – including the choice of the right algorithm for that application. But for most applications there are no such templates, no guidance, nothing.

Making things more difficult are heterogeneous environments in which the compute resources available vary from instance to instance. These variations make some load balancing algorithms unsuited to such environments. There is some general guidance you can use when trying to determine which algorithm is best suited to meeting the performance and scalability needs of your applications based on an understanding of how the algorithms are designed to make decisions, but if you want optimal performance and scalability you'll ultimately have to do some testing.

Heterogeneous environments can pose a challenge to scale if careful consideration of load balancing algorithms is not taken. Whether the limitations on compute resources are imposed by a virtualization solution or the hardware itself, limitations that vary from application instance to application instance are an important factor to consider when configuring your load balancing solution.

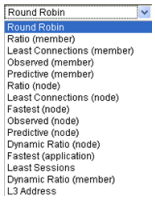


Let's say you've got a pool of application instances and you know the capacity of each in terms of connections (X). Two of the servers can handle 500 concurrent connections, and one can handle 1000 concurrent connections.

Now assume that your load balancer is configured to perform standard **round robin load balancing** between the three instances. Even though the total capacity of these three servers appears to be 2000 concurrent connections, by the time you hit 1501, the first of the three servers will be over capacity because it will have to try to handle 501 connections. If you tweak the configuration just a bit to indicate the maximum connection capacity for each node (instance) you can probably avoid this situation, but there are no guarantees.

Now let's make a small change in the algorithm – instead of standard round robin we'll use *weighted* round robin (often called "**ratio**"), and give the largest capacity server a higher weight based on its capacity ratio to the other servers, say 2. This means the "bigger" server will receive twice as many requests as the other two servers, which brings the total capacity closer to what is expected.

You might be thinking that a **least connection algorithm** would be more appropriate in a heterogeneous environment, but that's not the case. Least connection algorithms base distribution upon the number of connections currently open on any given server instance; it does not necessarily take into consideration the maximum connection capacity for that particular node. Fastest response time combined with per node connection limits would be a better option, but a **fastest response time algorithm** tends to result in a very unequal distribution as load increases in a heterogeneous environment.



This does not, however, say anything about the performance of the application when using any of the aforementioned algorithms. We do know that as application instances near capacity performance tends to degrade. Thus we could extrapolate that the performance for the two "smaller" servers will degrade faster than the performance for the bigger server because they will certainly reach capacity under high load before the larger server instance – when using some algorithms, at least. Algorithms like fastest response time and least connections tend to favor higher performing servers which means in the face of a sudden spike of traffic performance may degrade using *that* algorithm as well.

How about more "dynamic" algorithms that take into consideration multiple factors? Dynamic load balancing methods are designed to work with servers that differ in processing speed and memory. The resulting load balancing decisions may be uneven in terms of distribution but generally provides a more consistent user experience in terms of performance. For example, the **observed dynamic load balancing algorithm** distributes connections across applications based on a ratio calculated every second, and **predictive dynamic load balancing** uses the same ratio but also takes into consideration the change between previous connection counts and current connection counts and adjusts the ratio based on the delta. Predictive mode is more aggressive in adjusting ratio values for individual application instances based on connection changes in real-time and in a heterogeneous environment is likely better able to handle the differences between server capabilities.

What is TCP multiplexing?

TCP multiplexing is a technique used primarily by load balancers and application delivery controllers (but also by some stand-alone web application acceleration solutions) that enables the device to "reuse" existing TCP connections. This is similar to the way in which persistent HTTP 1.1 connections work in that a single HTTP connection can be used to retrieve multiple objects, thus reducing the impact of TCP overhead on application performance.

TCP multiplexing allows the same thing to happen for TCP-based applications (usually HTTP / web) except that instead of the reuse being limited to only 1 client, the connections can be reused over many clients, resulting in much greater efficiency of web servers and faster performing applications.

Interestingly enough, chatting with Dan Bartow (now CloudTest Evangelist and Vice President at SOASTA) about his experiences as Senior Manager of Performance Engineering at Intuit, revealed that testing different algorithms under heavy load generated externally finally led them to the discovery that a simple round robin algorithm combined with the application of TCP multiplexing options yielded a huge boost in both capacity and performance. But that was only after testing under conditions which were similar to those the applications would experience during peaks in usage and normalization of the server environment. This illustrates well that performance and availability isn't simply a matter of dumping a load balancing solution into the mix – it's important to test, to tweak configurations, and test again to find the overall infrastructure configuration that's going to provide the best application performance (and thus end-user experience) while maximizing resource utilization. Theoretical mathematically accurate models of load balancing are all well and good, but in the real world the complexity of the variables and interaction between infrastructure solutions and applications and servers is much higher, rendering the "theory" just that – theory.

Invariably which load balancing algorithm is right for your application is going to depend heavily on what metrics are most important to you. A balance of server efficiency, response time, and availability is likely involved, but which one of these key metrics is most important depends on what business stakeholders have deemed most important to them. The only way to really determine which load balancing algorithm will achieve the results you are looking for is to test them, under load, and observe the distribution and performance of the application.

FIRE and FORGET NOT a GOOD STRATEGY

The worst thing you can do is "fire and forget" about your load balancer. The algorithm that might be right for one application might not be right for another, depending on the style of application, its usage patterns, the servers used to serve it, and even the time of year. Unfortunately we're not quite at the point where the load balancer can automatically determine the right load balancing algorithm for you, but there are ways to adjust – dynamically – the algorithm based on not just the application but also the capabilities of the servers (physical and/or virtual) being load balanced so one day it is quite possible that through the magic of Infrastructure 2.0, load balancing algorithms will be modified on-demand based on the type of servers that make up the pool of resources.

In order for the level of sophistication we'd (all) like to see, however, it's necessary to first understand the impact of the load balancing algorithm on applications and determine which one is best able to meet the service level agreements in various environments based on a variety of parameters. This will become more important as public and private cloud computing environments are leveraged in new ways and introduce more heterogeneous environments. Seasonal demand might, for example, be met by leveraging different "sizes" of unused capacity across multiple servers in the data center. These "servers" would likely be of different CPU and RAM capabilities and thus would certainly be impacted by the choice of load balancing algorithm. Being able to dynamically modify the load balancing algorithm based on the capacities of application instances is an invaluable tool when attempting to maximize the efficiency of resources while minimizing associated costs.

There is, of course, a lack of control over algorithms in cloud computing environments, as well, that make the situation more difficult. With a limited set of choices available from providers the algorithm that's best for your application and server resource composition may not be available. Providers need to make it easier for customers to take advantage of modern, application and resource-aware algorithms that have evolved through trial-and-error over the past decade. Again, Infrastructure 2.0 enables this level of choice but must be leveraged by the provider to extend that choice and control to its customers.

For now, it's going to have to be enough to (1) thoroughly test the application and its supporting infrastructure under load and (2) adjust the load balancing algorithm to meet your specific performance criteria based on what is available. You might be surprised to find how much better your response time and capacity can be when you're using the "right" load balancing algorithm for your application – or at least one that's more right than it is wrong if you're in a cloud computing environment.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com