Beyond Performance Testing

by:

R. Scott Barber

Part 13: Testing and Tuning Load Balancers and Networks

In this installment of our final theme in this series, which I call "the performance testing and tuning team,"I'll discuss testing and tuning load balancers and networks. These are generally pretty easy to test, but the testing methods are fundamentally different from those discussed in Part 12 for the common tiers. Load balancers and networks shouldn't actually be causing performance problems or bottlenecks, but if they are, some configuration changes will usually remedy the problem. Once again, this article won't turn you into an expert on these topics, but I hope it will help you make a greater contribution to the testing and tuning team when faced with bottlenecks in these areas.

So far, this is what we've covered in this series:

Part 1: Introduction

Part 2: A Performance Engineering Strategy

Part 3: How Fast Is Fast Enough?

Part 4: Accounting for User Abandonment

Part 5: Determining the Root Cause of Script Failures

Part 6: Interpreting Scatter Charts

Part 7: Identifying the Critical Failure or Bottleneck

Part 8: Modifying Tests to Focus on Failure or Bottleneck Resolution

Part 9: Pinpointing the Architectural Tier of the Failure or Bottleneck

Part 10: Creating a Test to Exploit the Failure or Bottleneck

Part 11: Collaborative Tuning

Part 12: Testing and Tuning Common Tiers

This article is intended for mid- to senior-level performance testers and members of the development team who work closely with performance test engineers. You should have read at least Parts 9 and 11 before reading this article; it would be helpful to have read Parts 5, 6, 7, 8 and Part 10 as well.

Load Balancers

Load balancers are conceptually quite simple. They take the incoming load of client requests and distribute that load across multiple server resources. When configured correctly, a load balancer rarely causes a performance problem. But my experience shows that load balancers are often *not*

configured correctly, and this results in very poor performance. The only way to ensure that a load balancer is configured properly is to test it under load before it's put into use in the production system. The bottom line is that if the load balancer isn't speeding up your site or increasing the volume it can handle, it's not doing its job properly and needs to be reconfigured.

Before I describe how to test load balancers, I'll refresh your memory of the basic concepts behind load balancing and load balancers.

A Refresher on Load Balancers

Let's start by reviewing where the load balancer fits in the physical architecture. As you can see in Figure 1, the load balancer is generally the last stop a request makes before reaching a Web server.

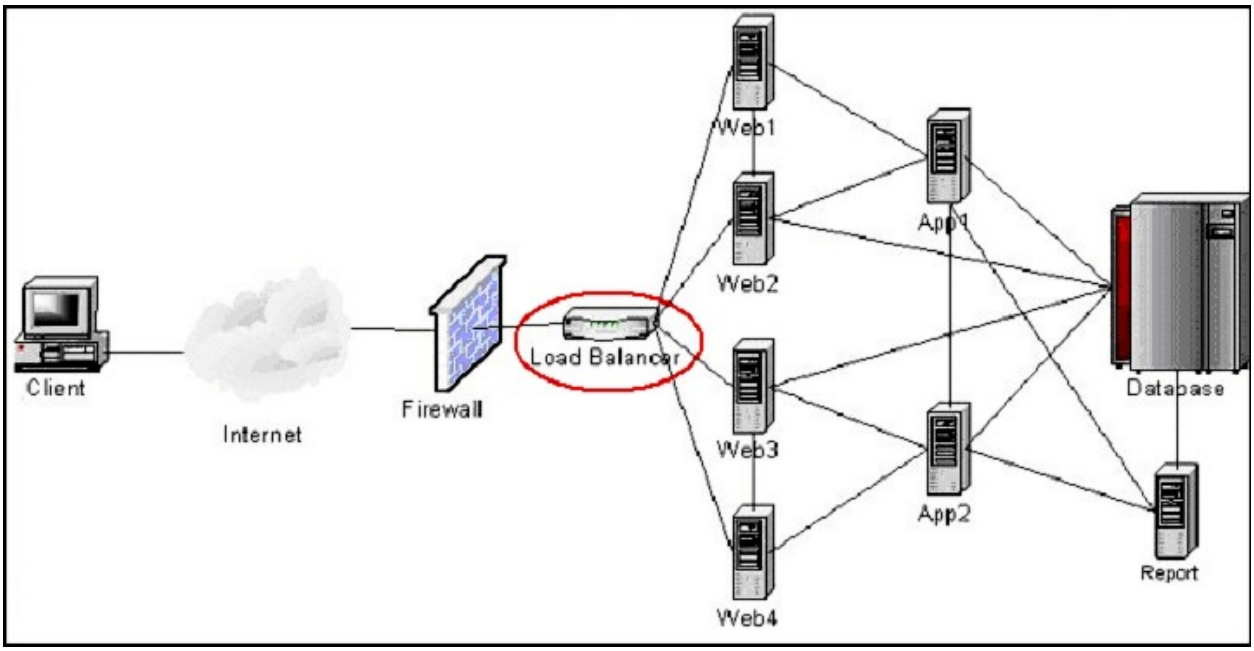
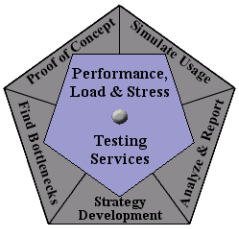


Figure 1: Typical physical architecture with load balancer

This diagram shows the load balancer as being separate from the firewall. Sometimes the same physical device serves as any combination of firewall, proxy server, router, and/or load balancer. Sometimes the load balancer is an actual hardware device with the software embedded (sometimes referred to as a content switch), while other times the load balancer is just software installed on a machine of your choosing.

On top of the different hardware and software configurations, there are many different ways that a load balancer distributes load. The simplest is known as the round-robin method. With this method, the load balancer simply takes each incoming request and sends it to the next Web server. For instance, if Figure 1 represented our actual environment, the first request to reach the load balancer would be directed to Web server 1, the second to server 2, the third to server 3, the fourth to server 4, and the fifth back to server 1.

The round-robin form of load balancing completely ignores the concept of user sessions, so that during a single session one user could be passing requests through several different Web servers. For an e-



commerce application, this won't work. For these types of applications, on sites that use sessions, the load balancer needs to be able to identify a particular user and keep that user pointed to the same Web server throughout the entire session.

Even this doesn't ensure a balanced load on each server. If somehow all of the users assigned to Web server 1 spent hours using the application and all the other users just spent minutes, Web server 1 could get overloaded while the other servers went underutilized. That's why many load balancers have dozens of load-balancing options and algorithms. They may balance by total traffic volume, they may monitor the resource utilization on the Web server to decide which server is least utilized at the moment the next request is received, or they may exercise any number of other possibilities.

For more information about a few currently popular hardware-based load balancers (including the balancing algorithms they support), see the Network Computing site.

Testing

If you've modeled a realistic workload, your performance test is going to be the best indicator of whether the load balancer is configured in the best possible way for your site. In fact, the only other way to see if the load balancer is configured properly is to release it into production and monitor the traffic.

In order to exercise the load balancer in a way that's useful for determining if it's configured properly, you'll need to make your virtual testers appear to be coming from unique IP addresses (since most load balancers identify users by IP address). If the load balancer doesn't recognize your virtual users as being unique users, it's not likely to balance the load properly. There are only two ways to make your virtual testers appear to be coming from unique IP addresses: use a lot of agents or enable IP aliasing through TestManager. Having lots of agents isn't always practical, so that leaves IP aliasing.

In case you're not familiar with it, IP aliasing allows many IP addresses to be assigned to the same physical system. Every virtual tester can be assigned a different IP address to realistically emulate your user community (although you don't necessarily need one IP address per virtual user, as I'll explain in a minute). The requests generated by these virtual testers receive responses back from the Web server with timing characteristics and validation recorded intact.

As explained by the TestManager Users Guide, if IP aliasing is enabled, the TestManager software on each computer (local or agent) queries the system for all available IP addresses at the beginning of a run. Each suite scheduled to run on that computer is assigned an IP address from that list, in round-robin fashion. If a computer has more virtual testers than IP addresses, an IP address is assigned to multiple virtual testers. If a computer has fewer virtual testers than IP addresses, some IP addresses aren't used. This approach optimizes the distribution of IP addresses regardless of the number of virtual testers scheduled on a computer and frees you from having to match IP addresses to specific virtual testers.

IP aliasing takes effect only if you're running HTTP test scripts and your system administrator has configured your system for IP aliasing. While this configuration may not be technically difficult, it requires a knowledge of which IP addresses are both valid and available on the network you're testing from. For Windows NT, the administrator sets up the IP addresses on any particular computer with the Settings > Control Panel > Network > Protocols > TCP/IP Protocol > Properties > Advanced > IP

Addresses > Add button. For UNIX, this can be done with the `ifconfig` command line utility.

But you don't need to worry about any of that. All you need to do is to enable IP aliasing through TestManager. If you've already created your suite, simply choose Suite > Edit Runtime from the menu bar and check the box as shown in Figure 2.

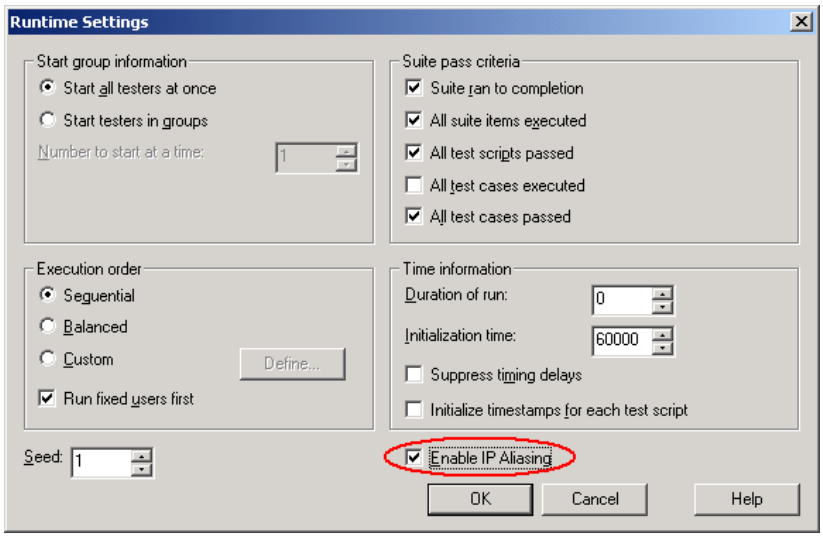


Figure 2: Enabling IP aliasing through TestManager

To test a load balancer, you don't necessarily need one IP address for every virtual tester, as mentioned above, but I recommend having at least four times the number of IP addresses as you have Web servers behind the load balancer. In the case of the environment depicted in Figure 1, I would recommend a minimum of sixteen IP addresses (four IPs times four Web servers) via any combination of agents and/or IP aliasing. Make certain you talk to the administrator in charge of configuring the load balancer. That person will be better able to tell you what a realistic sampling size will be for your environment.

Generally, you'll want to test a load balancer under a fairly high load. It often makes sense to do some performance testing and tuning on a single Web server (and associated back-end components) before adding the load balancer and other "legs" of the load-balanced environment (see Figure 3) to ensure that the rest of the system can handle the load that you'll be applying to the load balancer. If the rest of the application is unable to handle the load, you won't be able to effectively determine whether performance issues observed during the test are related to the load balancer or something else in the system.

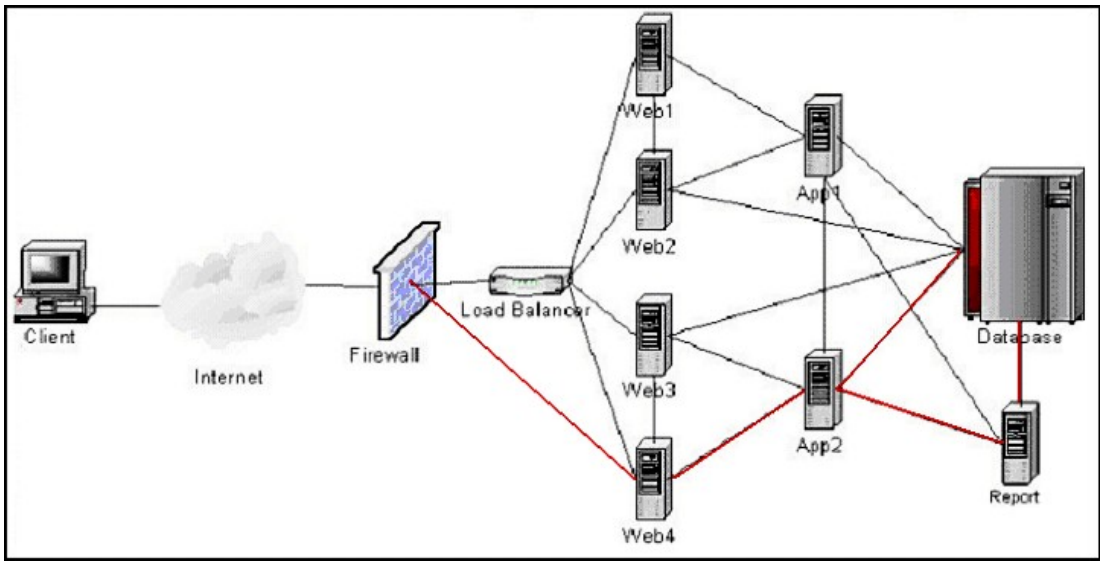


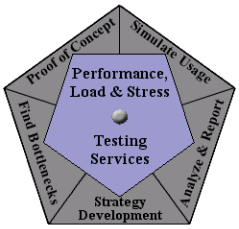
Figure 3: Single "leg" of a load-balanced environment (indicated by red lines)

Most teams believe they can stop testing after they've tested this single leg. They assume that if the application server has more than half its resources still available and if the database and report servers have more than 75% of their resources available when the Web server reaches its target load, adding the rest of the servers in this environment with a load balancer will simply multiply that target load by four. This is absolutely a faulty assumption. Just because one leg can handle a certain load doesn't mean that more legs will be able to handle more load. The actual environment must be tested. Even under ideal conditions, adding legs rarely, if ever, results in simple multipliers of performance volume.

Remember when testing a load-balanced environment to have someone monitoring not only the load balancer but also each Web server to ensure that it's handling its share of the load. Most of this type of testing isn't about response time but rather about ensuring the load balancer is configured correctly and determining the overall load the entire load-balanced system can handle. Ensure your testing and monitoring reflect that focus.

Tuning

This section could probably be titled "Configuring" rather than "Tuning." Since the majority of popular load balancers today come embedded on their own hardware, there's very little hardware tuning to be done. It's often the case, however, that additional configuring is required to ensure that all of the available Web servers are receiving an even portion of the generated load. It may be that the type of load that's being generated requires a different balancing algorithm than the one the load balancer is currently using. This is generally up to the administrator who handles the load balancer to decide. It's our job simply to provide an accurate load and to help monitor the Web servers to provide the administrator the information necessary to make any configuration changes to improve the distribution of the load.



Networks

You may remember that in Part 7 I asserted some rules for bottlenecks. One of those rules was: "The bottleneck is more likely to be found in the hardware than in the network, but the network is easier to check." I felt confident in making this assertion for two reasons. First, you don't need to have accurate performance scripts or a working application to test the network, and second, virtually all networks have an administrator and that administrator has tools to help you test the network.

Testing

The simplest way to start testing a network is to "ping" the remote server from your client. To use the `ping` utility in Windows, simply go to the Start menu, choose Programs > Accessories > Command Prompt, and type "ping [URL or IP to ping]" at the command line. Your results will look something like this:

```
C:\WINDOWS>ping www.testsite.com

Pinging www.testsite.com [17.2.240.92] with 32 bytes of data:

Reply from 17.2.240.92: bytes=32 time<10ms TTl=63
Reply from 17.2.240.92: bytes=32 time=10ms TTl=63
Reply from 17.2.240.92: bytes=32 time<10ms TTl=63
Reply from 17.2.240.92: bytes=32 time<10ms TTl=63

Ping statistics for 17.2.240.92;
  Packets Sent = 4, Received = 4, Lost = 0 <0% loss>,
Approximate round trip times in milli-seconds:
  Minimum = 5ms, Maximum = 10ms, Average = 7ms
```

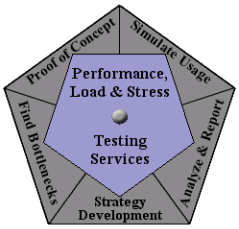
As you can see, the `ping` utility sent 32 bytes of data to www.testsite.com. This was done four times. One time, the round trip took 10 milliseconds (ms); the other times, it took less, for an average round trip time of 7 ms. In this case, we can see that the network between this client and www.testsite.com is very fast. If this ping were to be slow, it would be time to contact your network administrator.

The other DOS utility that can be useful in testing networks is `tracert` (short for *trace route*). Using `tracert` at the command line for our test site yielded the following results:

```
C:\WINDOWS>tracert www.testsite.com

Tracing route to www.testsite.com [17.2.240.92]
over a maximum of 30 hops:

 1    1 ms  <10 ms    1 ms 10.215.3.1
 2    4 ms    2 ms    2 ms 129.71.200.254
 3    3 ms    4 ms    4 ms 129.71.200.1
 4    4 ms    4 ms    3 ms 129.71.254.1
 5    8 ms    8 ms   10 ms 207.68.7.18
 6   32 ms   34 ms   41 ms 209.158.31.249
 7   43 ms   37 ms   41 ms 205.171.24.85
 8   51 ms   58 ms   47 ms 205.171.5.233
 9   59 ms   58 ms   58 ms 205.171.30.10
10   56 ms   45 ms   43 ms 205.171.30.14
11   46 ms   51 ms   41 ms 38.7.135.1
```




```
Trace complete.
```

This trace shows that a request took 11 hops to get from my location to the destination. Adding the duration of all these hops up, I come up with roughly 300 ms, or 0.3 seconds. Based on this information, I can pretty safely assume that about a third of a second of the time required for each request to reach its destination is due to network latency. That could really add up for a complex Web page. Additionally, if this were a test that I did from inside the firewall, 11 would be an excessive number of internal hops and would indicate a problem that should be addressed. You really don't want every HTTP request bouncing around on your internal network for 11 hops before getting to your Web server!

Both of these simple tests were done under no load. It's often useful to do these same tests during your test execution to see what effect executing the test has on the results. If these tests yield odd results, or if you've conducted these tests as well as other tests and can't find the slowdown, it's probably time to get the network administrator involved. The network administrator will normally have tools to show how much network bandwidth is being used, what the collision rate is, and what the utilization and resource utilization rates are on network components (like routers). The admin should even be able to "sniff" the network to get a clearer picture of what's going on. Find out what tools the admin has available and what kinds of load you should generate to best help the admin either track down bottlenecks on the network or eliminate the network from the list of potential bottleneck suspects.

Tuning

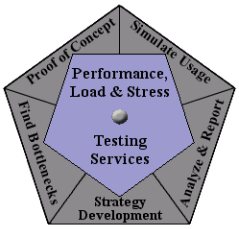
Tuning the network or network components is almost entirely in the hands of the network administrator. The best you can do is be at the admin's disposal to execute tests to verify any changes this person makes, or to develop a specific test to help pinpoint a problem. Even if you're a networking expert, it's unlikely that you have the specific information about the network the system under test is configured on to be able to help. From where we sit, it's nearly impossible to tell if the slow spot we're encountering is a router, a routing table, a proxy server, a firewall, or an overutilized network segment. Occasionally, the network can become congested, and that's a major (and expensive) problem to fix. Luckily, most network administrators are aware when the network is getting full so organizations can plan in advance.

Summing It Up

I've shown you a few simple things you can do to determine whether the load balancer is correctly configured and whether the network is causing any delays in response time. In both cases, making any changes in response to your findings is up to the administrator involved. Your role is to accurately model the load and monitor what goes on during your testing. As I've implied before, sometimes being a helper is as good as being a hero.

Acknowledgments

- The original version of this article was written on commission for IBM Rational and can be found on the IBM DeveloperWorks web site



About the Author

Scott Barber is the CTO of PerfTestPlus (www.PerfTestPlus.com) and Co-Founder of the Workshop on Performance and Reliability (WOPR – www.performance-workshop.org). Scott's particular specialties are testing and analyzing performance for complex systems, developing customized testing methodologies, testing embedded systems, testing biometric identification and security systems, group facilitation and authoring instructional or educational materials. In recognition of his standing as a thought leading performance tester, Scott was invited to be a monthly columnist for Software Test and Performance Magazine in addition to his regular contributions to this and other top software testing print and on-line publications, is regularly invited to participate in industry advancing professional workshops and to present at a wide variety of software development and testing venues. His presentations are well received by industry and academic conferences, college classes, local user groups and individual corporations. Scott is active in his personal mission of improving the state of performance testing across the industry by collaborating with other industry authors, thought leaders and expert practitioners as well as volunteering his time to establish and grow industry organizations. His tireless dedication to the advancement of software testing in general and specifically performance testing is often referred to as a hobby in addition to a job due to the enjoyment he gains from his efforts.

About PerfTestPlus

PerfTestPlus was founded on the concept of making software testing industry expertise and thought-leadership available to organizations, large and small, who want to push their testing beyond "state-of-the-practice" to "state-of-the-art." Our founders are dedicated to delivering expert level software-testing-related services in a manner that is both ethical and cost-effective. PerfTestPlus enables individual experts to deliver expert-level services to clients who value true expertise. Rather than trying to find individuals to fit some pre-determined expertise or service offering, PerfTestPlus builds its services around the expertise of its employees. What this means to you is that when you hire an analyst, trainer, mentor or consultant through PerfTestPlus, what you get is someone who is passionate about what you have hired them to do, someone who considers that task to be their specialty, someone who is willing to stake their personal reputation on the quality of their work - not just the reputation of a distant and "faceless" company.