Apache Performance Tuning
Part Two: Scaling Out

Sander Temme
sander@temme.net

May 30, 2006

Abstract

As your web site grows in popularity, you will get to the point when one server doesn't cut it anymore. You need to add more boxes, and this session discusses several approaches to scaling out. We will cover webserver load balancing, SSL offload and separating application tiers. We will also discuss configuring the Apache HTTP Server to front Apache Tomcat servers and how to load balance between Tomcat servers. Finally, we will cover Java VM and database tuning.

1 Introduction

Building out a web server infrastructure is a large and diverse field. The server infrastructure for any large web site is to a large extent customized for the needs and requirements of that site and it is very hard to make valid general statements about scaling technologies. This paper and its accompanying ApacheCon presentation aim to give a general overview of the field, touching upon approaches and technologies rather than discussing them in depth.

1.1 Why Would You Scale Out?

Scaling Out is a business decision. You may scale out because you can not meet your performance goals with a single web server. Alternatively, you may scale out to meet reliability and uptime goals. There are many approaches to scaling out, with varying price tags. So whatever your motivation, to scale out your web infrastructure you will have to justify added expenses for server hardware, network equipment, possibly software licenses and maintenance contracts, and most certainly system administration time and resources. Very few of us have the above thrown our way without a defined business need.

2 Building Out: Load Balancing

Scaling Out means adding more servers. The question when adding servers to your infrastructure is how to direct client transactions to multiple hosts. The user does not know and should not care that multiple servers are in use. They just want to point their browser to `www.example.com` and spend a lot of money on your products or services. In this section we will review several techniques to distribute client transactions across your hosts.

2.1 Load Balancing with DNS

You have a great deal of control over where your users direct their transactions by using the Domain Name Service (DNS) for your site. This seems obvious, but it has to be mentioned. When your users connect to `www.example.com`, they don't care to which IP address this resolves. If you can manipulate this resolution, you can send the user to whichever physical server you prefer.

2.1.1 Distinct Servers for Distinct Services

One way to distribute transaction load across multiple physical servers is to give each server a separate task. For your `www.example.com` site, use an `images.example.com` server to serve static image content, a `secure.example.com` server to handle SSL transactions, etc. This approach allows you to tune each server for its specialized task. The downside is that this approach does not scale by itself: once, for instance, your secure server runs out of processing headroom, you will have to add more machines using one of the techniques described below.

2.1.2 DNS Round-Robin

If you operate multiple servers that perform identical functions, you can distribute client transactions among them using Domain Name Server Round-Robin. The principle behind this technique is that a single server hostname resolves to a different IP address from your server pool for each DNS resolution request. For instance, if you have three web servers with the IP addresses

```
10.11.0.113
10.11.0.114
10.11.0.115
```

and you have your name server return each of those addresses in turn for queries to your web server name (`www.scalingout.org`), roughly one third of all clients will connect to each of your web servers. Since popular name server implementations like *bind* implement this technique by default, it is very simple to implement without any resource requirements besides control over your DNS zone.

How "roughly" this works depends on many factors, over few of which you have any control. Client-side resolvers cache query reponses, as do intermediate

nameservers at ISPs and corporations. Large ISPs and corporations represent many potential users, all of whom would be directed to the same web server for as long as their nameserver caches the original lookup. However, across your entire user population these discrepancies may even out. You can help this process by reducing the cache timeout for query results in your zone file. An example zone file that uses DNS Round-Robin is shown in Appendix A.

DNS Round-Robin as a load balancing approach is often disparaged because of its simplicity: it does not take into account the load on the servers, and can not compensate for server outage. If a server goes down for whatever reason, one third of all clients will still be directed to the nonfunctional server. If these considerations are important to you, consider one of the more sophisticated load balancing approaches described below. However, do not dismiss DNS Round-Robin out of hand. Depending on your requirements, it may be all you need.

2.2 Peer-based Load Balancing

You can turn a collection of individual servers into a cluster by using load balancing techniques. In this section we will discuss Microsoft's approach.

2.2.1 Windows Network Load Balancing

Windows Load Balancing Service (WLBS) technology has been available since Windows NT Server 4.0, Enterprise Edition and is now included in Windows Server 2003 under the name Network Load Balancing (NLB). Using Network Load Balancing, you can turn up to 32 servers into a cluster. The service work by having every machine assume the same IP address, and the same MAC address, on the clustered interface(s). Incoming connections arrive at all members of the cluster simultaneously from the network switch. The NLB software communicates between cluster members over a unicast or multicast backchannel and is responsible for the load balancing decisions. It sits between the network card driver and the TCP/IP stack, and regulates which cluster member gets to answer each incoming request. Cluster members whose NLB module doesn't communicate with the other members get removed from the pool. This allows NLB to provide High Availability as well as load balancing functionality.

Because it operates below the TCP/IP layer, Network Load Balancing should be compatible with any service that runs on the server machines. Each cluster member has to be configured exactly the same. Please see your Windows Server 2003 documentation for details.

2.3 Load Balancing Appliance

The market for dedicated load balancing appliances is now quite crowded, with offerings from vendors like Cisco, F5, Juniper, Foundry and many others vying for your attention. These products tend to be pricy, but very powerful solutions to load balance your server farm.

2.3.1 How a Load Balancer Works

Load balancing appliances or application switches sit between the web servers and the outbound network connection and intelligently distribute traffic across multiple web servers. They typically keep track of the load and availability of the servers, and adjust their load balancing decisions accordingly. Many of these products can operate on several layers of the network stack and can inspect incoming requests to make load balancing decisions based on source address, requested URI, cookies submitted etc.

2.3.2 Linux Virtual Server

The Linux Virtual Server project is an open source load balancing and high availability implementation. Its core module, IP Virtual Server, is included in the kernel as of version 2.6.10. Auxiliary software like `ipvsadm` is only an install away. If you are considering rolling your own load balancing solution, consider Linux Virtual Server.

The primary disadvantage of Linux Virtual Server is that it does not come as a nice, shiny plug-and-play box with a support contract. Instead, it looks more like an Erector Set[1] of bits and pieces that you get to integrate yourself. However, this disadvantage can also be a strength: it allows you to build a solution that best fits your needs. However, the absence of a 24x7 support plan may upset your decision makers. You can find an example configuration for Linux Virtual Server in Appendix B.

3 Building Out: Separate Tiers

Most web applications can be separated into multiple distinct tiers:

1. Web server tier(Apache, IIS, Sun ONE)
2. Application server tier (Tomcat, PHP, WebLogic, etc.)
3. Database server tier (MySQL, Oracle, Postgres, etc.)

Every tier has distinct and particular performance requirements. Moving each tier to their own hardware allows you to tune and scale them individually. The fact that all of the individual applications communicate with each other over TCP/IP already makes this move even easier.

The **Web Server** tier communicates directly with the users. It is responsible for maintaining connection with a wide variety of client browsers across potentially slow and far-flung connections. This causes a markedly different load on the operating system TCP stack than the long-lived, local, high speed connections between web and application server, and between application server and the database. The web tier can also be configured to serve the application's

[1] Perhaps better known in Europe as *Meccano*

static content: HTML pages, images, JavaScript, etc. It passes only the requests for dynamically generated content (PHP scripts, JavaServer Pages, RSS feeds) on to the application tier. The type of server used for this tier typically has one or two CPUs and enough memory to fit the requisite number of httpd processes. Storage is not a concern.

The **Application Server** tier generates all dynamic content. It receives requests from the web tier and maintains connections to the database tier. The operating system can be tuned specifically to run an application server platform such as a Java virtual machine. The type of server used for this tier may have multiple CPUs as required to run application threads. These servers have more memory than the web servers as required by the application platform, but storage is not important on this tier.

The **Database Server** tier stores all application data. The application server tier connects to the database tier using the JDBC protocol or native database libraries. Database access can be a considerable bottleneck for application performance, so performance is an important consideration for this tier. The type of server used for this tier should have sufficient CPU power and RAM to run the database application, and come with scalable, redundant storage like a RAID-5 array.

4 Designing Your Site for Scaling Out

4.1 Designing for a Load Balancer

A Load Balancer introduces an additional moving part to your web server infrastructure. While most load balancer solutions do their best to appear transparent to the application, you may find some issues that you can solve by properly designing your application.

The main issue arises with session persistence. The HTTP protocol is inherently stateless, which is great for a load balancer: it can consider each incoming request for itself and make a completely independent load balancing decision based on its criteria. Session persistence potentially complicates this issue, especially if a user's session exists only on the server that initially created it. If a subsequent request from that user is directed to a different backend server, the session is lost. Most load balancing solutions solve this problem by consistently directing requests from a particular IP address to the same backend server. Some can inspect incoming HTTP requests and make load balancing decisions based on session cookies.

These load balancer based fixes should be enough under most circumstances, but your requirements may be more stringent: what if the user reconnects after a long time and the load balancer has timed out its IP based persistence? Or the user reconnects from a different IP address (let's say she left one Starbucks and reconnects from the one across the street)? Or the server that holds the user's session goes offline because of a crash or maintenance? If it is important to you to maintain user sessions under circumstances like these, you should build

session persistence into your application. Users sessions are likely to cause more reads than writes. You could write session information to your backend database, or use a special, fast database with a write-through cache just for session maintenance.

5 Conclusion

Scaling out your web site is a mixed blessing. While you get to serve more transactions and, presumably, do more business, the additional hardware, software and network segments will also give you more problems. You get to manage, maintain en secure a farm of servers instead of just one. The configuration of your servers, and application software and content design will be highly influenced by the infrastructure design decisions you make, and they will be heavily intertwined. By the time your are done with this, you will know far more than a general paper or conference session can ever address.

A DNS Round-Robin Zone File

The following is a very basic DNS Zone file that uses Round-Robin DNS to balance three web servers.

```
scalingout.org. 86400 IN SOA ns.scalingout.org. sctemme.scalingout.org. (
           2006051401 ; Serial
           86400      ; refresh (1 day)
           7200       ; retry  (2 hours)
           8640000    ; expire (10 days)
           86400 )    ; minimum (1 day)

scalingout.org.     IN      NS      bagheera.scalingout.org.

gw              IN      A       10.11.0.1
bagheera        IN      A       10.11.0.2

; ...

mail            IN      CNAME   bagheera
ns              IN      CNAME   bagheera

www             IN      A       10.11.0.113
                IN      A       10.11.0.114
                IN      A       10.11.0.115
```

B Linux Virtual Server Configuration

This example uses a Linux Virtual Server director running Ubuntu 5.10 (The Breezy Badger). The outside interface of the Director has IP address 10.0.0.1, its inside interface is on 192.168.1.1. Two backend web servers are connected to an internal interface of the Director. Their Ethernet interfaces are configured for 192.168.1.2 and 192.168.1.3 respectively, and both have 192.168.1.1 for default gateway. On the Director machine, the file `/etc/ipvsadm.rules` has the following information:

```
# ipvsadm.rules
-A -t 10.0.0.1:80 -s rr
-a -t 10.0.0.1:80 -r 192.168.1.2:8080 -m -w 1
-a -t 10.0.0.1:80 -r 192.168.1.3:8080 -m -w 1
```

and the file /etc/defaults/ipvsadm looks as follows:

```
# Do not edit! Use 'dpkg-reconfigure ipvsadm'.
AUTO="true"
DAEMON="none"
```

The tool mentioned in the comment has interactive menus for the two variables. This is all the configuration necessary to run Linux Virtual Server in NAT mode: a reboot or the command `/etc/init.d/ipvsadm start` issued as root starts the load balancer.