CHAPTER 8

Choosing a Platform for the Best Performance

Before you start to optimize your code and server configuration, you need to consider the demands that will be placed on the hardware and the operating system. There is no point in investing a lot of time and money in configuration tuning and code optimizing only to find that your server's performance is poor because you did not choose a suitable platform in the first place.

Because hardware platforms and operating systems are developing rapidly, the following advisory discussion must be in general terms, without mentioning specific vendors' names.

Choosing the Right Operating System

This section discusses the characteristics and features you should be looking for to support a mod_perl-enabled Apache server. When you know what you want from your OS, you can go out and find it. Visit the web sites of the operating systems that interest you. You can gauge users' opinions by searching the relevant discussions in newsgroup and mailing-list archives. Deja (*http://deja.com/*) and eGroups (*http://egroups.com/*) are good examples. However, your best shot is probably to ask other mod_perl users.

mod_perl Support for the Operating System

Clearly, before choosing an OS, you will want to make sure that mod_perl even *runs* on it! As you will have noticed throughout this book, mod_perl 1.x is traditionally a Unix-centric solution. Although it also runs on Windows, there are several limitations related to its implementation.

The problem is that Apache on Windows uses a multithreaded implementation, due to the fact that Windows can't use the multi-process scheme deployed on Unix platforms. However, when mod_perl (and thereby the Perl runtime) is built into the

Apache process, it *cannot* run multithreaded, because before Version 5.8.0 the Perl runtime wasn't thread-safe.

What does this mean for you? Well, essentially it means that your Apache process will be able to serve only *one request at a time*, just like when using *httpd -X*. Of course, this becomes a severe performance hit, making you unable to have more than one user receiving a page at a time. The situation is resolved in mod_perl 2.0, however, thanks to advances in both Apache and Perl, as described in Chapter 24. Furthermore, you can still use mod_perl on Windows for development, although you should follow the considerations below when choosing the production OS.

Stability and Robustness

Probably the most important features in an OS are stability and robustness. You are in an Internet business. You do not keep normal 9 A.M. to 5 P.M. working hours like many conventional businesses you know. You are open 24 hours a day. You cannot afford to be offline, because your customers will go shop at another service like yours (unless you have a monopoly). If the OS of your choice crashes every day, first do a little investigation. There might be a simple reason that you can find and fix. However, there are OSes that won't work unless you reboot them twice a day. You don't want to use an OS of this kind, no matter how good the OS's vendor sales department is. Do not follow flush advertisements—follow developers' advice instead.

Generally, people who have used an OS for some time can tell you a lot about its stability. Ask them. Try to find people who are doing similar things to what you are planning to do; they may even be using the same software. There are often compatibility issues to resolve, and you may need to become familiar with patching and compiling your OS.

Good Memory Management

You want an OS with a good memory-management implementation. Some OSes are well known as memory hogs. The same code can use twice as much memory on one OS compared to another. If the size of the mod_perl process is 10 MB and you have tens of these processes running, it definitely adds up!

Avoiding Memory Leaks

Some OSes and/or their libraries (e.g., C runtime libraries) suffer from memory leaks. A *leak* is when some process requests a chunk of memory for temporary storage but then does not subsequently release it. The chunk of memory then won't be available for any purpose until the process that requested it dies. You cannot afford such leaks. A single mod_perl process sometimes serves thousands of requests before it terminates; if a leak occurs on every request, the memory demands could become huge. Of course, your code can be the cause of the memory leaks as well, but that's easy to detect and solve. Certainly, you can reduce the number of requests to be served over the process's life, but that can degrade performance. When you have so many performance concerns to think about, do you *really* want to be using faulty code that's not under your control?

Memory-Sharing Capabilities

You want an OS with good memory-sharing capabilities. If you preload the Perl modules and scripts at server startup, they are shared between the spawned children (at least for part of a process's life—memory pages can become "dirty" and cease to be shared). This feature can vastly reduce memory consumption. Therefore, you don't want an OS that doesn't have memory-sharing capabilities.

The Real Cost of Support

If you are in a big business, you probably do not mind paying another $1,000 for some fancy OS with bundled support. But if your resources are low, you will look for cheaper or free OSes. Free does not mean bad. In fact, it can be quite the opposite—some of the free OSes have the best support available.

This is easy to understand—most people are not rich and will try to use a cheaper or free OS first if it does the work for them. If it fits their needs, they will keep using it and eventually come to know it well enough to be able to provide support for others in trouble. Why would they do this for free? One reason is the spirit of the first days of the Internet, when there was no commercial Internet and people helped each other because someone else had helped them first. We were there, we were touched by that spirit, and we are keen to keep that spirit alive.

Nevertheless, we are living in a material world, and our bosses pay us to keep the systems running. So if you feel that you cannot provide the support yourself and you do not trust the available free resources, you must pay for an OS backed by a company to which you can turn in case of problems. Insufficient support has often been characterized as an important drawback of open source products, and in the past it may have been the main reason for many companies to choose a commercial product.

Luckily, in recent years many companies have realized how good the open source products are and started to provide official support for these products. So your suggestion of using an open source operating system cannot be dismissed solely on the basis of lacking vendor support; most likely you will be able to find commercial support just like with any other commercial OS vendor!

Also remember that the less money you spend on an OS and software, the more you will be able to spend on faster and stronger hardware. Of course, for some companies money is a non-issue, but there are many companies for which it is a major concern.

Discontinued Products

You might find yourself in a position where you have invested a lot of time and money into developing some proprietary software that is bundled with the OS you chose (say, writing a mod_perl handler that takes advantage of some proprietary features of the OS and that will not run on any other OS). Things are under control, the performance is great, and you sing with happiness on your way to work. Then, one day, the company that supplies your beloved OS goes bankrupt (not unlikely nowadays), or they produce a newer, incompatible version and decide not to support the old one (it happens all the time). You are stuck with their early masterpiece, no support, and no source code! What are you going to do? Invest more money into porting the software to another OS?

The OSes in this hazard group tend to be developed by a single company or organization, so free and open source OSes are probably less susceptible to this kind of problem. Their development is usually distributed between many companies and developers, so if a person who developed a really important part of the kernel loses interest in continuing, someone else usually will pick up the work and carry on. Of course, if some better project shows up tomorrow, developers might migrate there and finally drop the development, but in practice people are often given support on older versions and helped to migrate to current versions. Development tends to be more incremental than revolutionary, so upgrades are less traumatic, and there is usually plenty of notice of the forthcoming changes so that you have time to plan for them.

Of course, with the open source OSes you have the source code, too. You can always have a go at maintaining it yourself, but do not underestimate the amount of work involved.

Keeping Up with OS Releases

Actively developed OSes generally try to keep pace with the latest technology developments and continually optimize the kernel and other parts of the OS to become better and faster. Nowadays, the Internet and networking in general are the hottest topics for system developers. Sometimes a simple OS upgrade to the latest stable version can save you an expensive hardware upgrade. Also, remember that when you buy new hardware, chances are that the latest software will make the most of it.

If a new product supports an old one by virtue of backward compatibility with previous products of the same family, you might not reap all the benefits of the new product's features. You might get almost the same functionality for much less money if you were to buy an older model of the same product.

Choosing the Right Hardware

Sometimes the most expensive machine is not the one that provides the best performance. Your demands on the platform hardware are based on many aspects and affect many components. Let's discuss some of them.

This discussion relies on the specific definitions of various hardware and operating-system terms. Although you may be familiar with the terms below, we have explicitly provided definitions to make sure there is no ambiguity when we discuss the hardware strategies.

Cluster
: A group of machines connected together to perform one big or many small computational tasks in a reasonable time. Clustering can also be used to provide failover, where if one machine fails, its processes are transferred to another without interruption of service. And you may be able to take one of the machines down for maintenance (or an upgrade) and keep your service running—the main server simply will not dispatch the requests to the machine that was taken down.

Load balancing
: Say that users are given the name of one of your machines, but it cannot stand the heavy load. You can use a clustering approach to distribute the load over a number of machines (which gives you the advantages of clustering, too). The central server, which users access initially when they type the name of your service into their browsers, works as a dispatcher. It redirects requests to other machines, and sometimes the central server also collects the results and returns them to the users.

Network Interface Card (NIC)
: A hardware component that allows your machine to connect to the network. It sends and receives packets. NICs come in different speeds, varying from 10 MBps to 10 GBps and faster. The most widely used NIC type is the one that implements the Ethernet networking protocol.

Random Access Memory (RAM)
: The memory that you have in your computer (comes in units of 8 MB, 16 MB, 64 MB, 256 MB, etc.).

Redundant Array of Inexpensive Disks (RAID)
: An array of physical disks, usually treated by the operating system as one single disk, and often forced to appear that way by the hardware. The reason for using RAID is often simply to achieve a high data-transfer rate, but it may also be to get adequate disk capacity or high reliability. *Redundancy* means that the system is capable of continued operation even if a disk fails. There are various types of RAID arrays and several different approaches to implementing them. Some systems provide protection against failure of more than one drive and some ("hot-swappable") systems allow a drive to be replaced without even stopping the OS.

Machine Strength Demands According to Expected Site Traffic

If you are building a fan site and you want to amaze your friends with a mod_perl guestbook, any old 486 machine could do it. But if you are in a serious business, it is very important to build a scalable server. If your service is successful and becomes popular, the traffic could double every few days, and you should be ready to add more resources to keep up with the demand. While we can define the web server scalability more precisely, the important thing is to make sure that you can add more power to your web server(s) without investing much additional money in software development (you will need a little software effort to connect your servers, if you add more of them). This means that you should choose hardware and OSes that can talk to other machines and become part of a cluster.

On the other hand, if you prepare for a lot of traffic and buy a monster to do the work for you, what happens if your service doesn't prove to be as successful as you thought it would be? Then you've spent too much money, and meanwhile faster processors and other hardware components have been released, so you lose.

Wisdom and prophecy, that's all it takes. :)

A Single Strong Machine Versus Many Weaker Machines

Let's start with a claim that a four-year-old processor is still very powerful and can be put to good use. Now let's say that for a given amount of money you can probably buy either one new, very strong machine or about 10 older but very cheap machines. We claim that with 10 old machines connected into a cluster, by deploying load balancing, you will be able to serve about five times more requests than with a single new machine.

Why is that? Generally the performance improvement on a new machine is marginal, while the price is much higher. Ten machines will do faster disk I/O than one single machine, even if the new disk is quite a bit faster. Yes, you have more administration overhead, but there is a chance that you will have it anyway, for in a short time the new machine you have just bought might not be able to handle the load. Then you will have to purchase more equipment and think about how to implement load balancing and web server filesystem distribution anyway.

Why are we so convinced? Look at the busiest services on the Internet: search engines, webmail servers, and the like—most of them use a clustering approach. You may not always notice it, because they hide the real implementation details behind proxy servers, but they do.

Getting a Fast Internet Connection

You have the best hardware you can get, but the service is still crawling. What's wrong? Make sure you have a fast Internet connection—not necessarily as fast as your ISP claims it to be, but as fast as it should be. The ISP might have a very good connection to the Internet but put many clients on the same line. If these are heavy clients, your traffic will have to share the same line and your throughput will suffer. Think about a dedicated connection and make sure it is truly dedicated. Don't trust the ISP, check it!

Another issue is connection latency. Latency defines the number of milliseconds it takes for a packet to travel to its final destination. This issue is really important if you have to do interactive work (via *ssh* or a similar protocol) on some remote machine, since if the latency is big (400+ ms) it's really hard to work. It is less of an issue for web services, since it influences only the first packet. The rest of the packets arrive without any extra delay.

The idea of having a connection to "the Internet" is a little misleading. Many web hosting and colocation companies have large amounts of bandwidth but still have poor connectivity. The public exchanges, such as MAE-East and MAE-West, frequently become overloaded, yet many ISPs depend on these exchanges.

Private peering is a solution used by the larger backbone operators. No longer exchanging traffic among themselves at the public exchanges, each implements private interconnections with each of the others. Private peering means that providers can exchange traffic much quicker.

Also, if your web site is of global interest, check that the ISP has good global connectivity. If the web site is going to be visited mostly by people in a certain country or region, your server should probably be located there.

Bad connectivity can directly influence your machine's performance. Here is a story one of the developers told on the mod_perl mailing list:

```
What relationship has 10% packet loss on one upstream provider got to
do with machine memory ?

Yes.. a lot. For a nightmare week, the box was located downstream of a
provider who was struggling with some serious bandwidth problems of
his own... people were connecting to the site via this link, and
packet loss was such that retransmits and TCP stalls were keeping
httpd heavies around for much longer than normal.. instead of blasting
out the data at high or even modem speeds, they would be stuck at
1k/sec or stalled out...  people would press stop and refresh, httpds
would take 300 seconds to timeout on writes to no-one.. it was a
nightmare.  Those problems didn't go away till I moved the box to a
place closer to some decent backbones.
```

```
Note that with a proxy, this only keeps a lightweight httpd tied up,
assuming the page is small enough to fit in the buffers.  If you are a
busy internet site you always have some slow clients.  This is a
difficult thing to simulate in benchmark testing, though.
```

Tuning I/O Performance

If your service is I/O-bound (i.e., does a lot of read/write operations to disk) you need a very fast disk, especially when using a relational database. Don't spend the money on a fancy video card and monitor! A cheap card and a 14-inch monochrome monitor are perfectly adequate for a web server—you will probably access it by *telnet* or *ssh* most of the time anyway. Look for hard disks with the best price/performance ratio. Of course, ask around and avoid disks that have a reputation for headcrashes and other disasters.

Consider RAID or similar systems when you want to improve I/O's throughput (performance) and the reliability of the stored data, and of course if you have an enormous amount of data to store.

OK, you have a fast disk—so what's next? You need a fast disk controller. There may be a controller embedded on your computer's motherboard. If the controller is not fast enough, you should buy a faster one. Don't forget that it may be necessary to disable the original controller.

How Much Memory Is Enough?

How much RAM do you need? Nowadays, chances are that you will hear: "Memory is cheap, the more you buy the better." But how much is enough? The answer is pretty straightforward: *you do not want your machine to swap*! When the CPU needs to write something into memory, but memory is already full, it takes the least frequently used memory pages and swaps them out to disk. This means you have to bear the time penalty of writing the data to disk. If another process then references some of the data that happens to be on one of the pages that has just been swapped out, the CPU swaps it back in again, probably swapping out some other data that will be needed very shortly by some other process. Carried to the extreme, the CPU and disk start to thrash hopelessly in circles, without getting any real work done. The less RAM there is, the more often this scenario arises. Worse, you can exhaust swap space as well, and then your troubles really start.

How do you make a decision? You know the highest rate at which your server expects to serve pages and how long it takes on average to serve one. Now you can calculate how many server processes you need. If you know the maximum size to which your servers can grow, you know how much memory you need. If your OS supports memory sharing, you can make best use of this feature by preloading the modules and scripts at server startup, so you will need less memory than you have calculated.

Do not forget that other essential system processes need memory as well, so you should not only plan for the web server but also take into account the other players. Remember that requests can be queued, so you can afford to let your client wait for a few moments until a server is available to serve it. Most of the time your server will not have the maximum load, but you should be ready to bear the peaks. You need to reserve at least 20% of free memory for peak situations. Many sites have crashed a few moments after a big scoop about them was posted and an unexpected number of requests suddenly arrived. If you are about to announce something cool, be aware of the possible consequences.

Getting a Fault-Tolerant CPU

Make sure that the CPU is operating within its specifications. Many boxes are shipped with incorrect settings for CPU clock speed, power supply voltage, etc. Sometimes a if cooling fan is not fitted, it may be ineffective because a cable assembly fouls the fan blades. Like faulty RAM, an overheating processor can cause all kinds of strange and unpredictable things to happen. Some CPUs are known to have bugs that can be serious in certain circumstances. Try not to get one of them.

Detecting and Avoiding Bottlenecks

You might use the most expensive components but still get bad performance. Why? Let me introduce an annoying word: bottleneck.

A machine is an aggregate of many components. Almost any one of them may become a bottleneck. If you have a fast processor but a small amount of RAM, the RAM will probably be the bottleneck. The processor will be underutilized, and it will often be waiting for the kernel to swap the memory pages in and out, because memory is too small to hold the busiest pages.

If you have a lot of memory, a fast processor, and a fast disk, but a slow disk controller, the disk controller will be the bottleneck. The performance will still be bad, and you will have wasted money.

A slow NIC can cause a bottleneck as well and make the whole service run slowly. This is a most important component, since web servers are much more often network-bound than they are disk-bound (i.e., they have more network traffic than disk utilization).

Solving Hardware Requirement Conflicts

It may happen that the combination of software components you find yourself using gives rise to conflicting requirements for the optimization of tuning parameters. If you can separate the components onto different machines you may find that this approach (a kind of clustering) solves the problem, at much less cost than buying faster hardware, because you can tune the machines individually to suit the tasks they should perform.

For example, if you need to run a relational database engine and a mod_perl server, it can be wise to put the two on different machines, since an RDBMS needs a very fast disk while mod_perl processes need lots of memory. Placing the two on different machines makes it easy to optimize each machine separately and satisfy each software component's requirements in the best way.

References

- For more information about RAID, see the Disk-HOWTO, Module-HOWTO, and Parallel-Processing-HOWTO, available from the Linux Documentation Project and its mirrors (*http://www.tldp.org/docs.html#howto*).
- For more information about clusters and high-availability setups, see:

  High-Availability Linux Project, the definitive guide to load-balancing techniques: *http://www.linux-ha.org/*

  Linux Virtual Server Project: *http://www.linuxvirtualserver.org/*

  mod_backhand, which provides load balancing for Apache: *http://www.backhand.org/mod_backhand/*

  lbnamed, a load balancing name server written in Perl: *http://www.stanford.edu/~riepel/lbnamed/*, *http://www.stanford.edu/~riepel/lbnamed/bof.talk/*, or *http://www.stanford.edu/~schemers/docs/lbnamed/lbnamed.html*
- Chapters 11 to 18 of *Web Performance Tuning*, by Patrick Killelea (O'Reilly).
- Chapters 2 and 12 in *Optimizing UNIX for Performance*, by Amir H. Majidimehr (Prentice Hall).
- Chapter 9 ("Tuning Apache and mod_perl") in *mod_perl Developer's Cookbook*, by Geoffrey Young, Paul Lindner, and Randy Kobes (Sams Publishing).