1   Choosing an Operating System and Hardware

1.1 Description

Before you use the techniques documented on this site to tune servers and write code you need to consider the demands which will be placed on the hardware and the operating system. There is no point in investing a lot of time and money in configuration and coding only to find that your server's performance is poor because you did not choose a suitable platform in the first place.

While the tips below could apply to many web servers, they are aimed primarily at administrators of mod_perl enabled Apache server.

Because hardware platforms and operating systems are developing rapidly (even while you are reading this document), this discussion must be in general terms.

1.2 Choosing an Operating System

First let's talk about Operating Systems (OSs).

Most of the time I prefer to use Linux or something from the *BSD family. Although I am personally a Linux devotee, I do not want to start yet another OS war.

I will try to talk about what characteristics and features you should be looking for to support an Apache/mod_perl server, then when you know what you want from your OS, you can go out and find it. Visit the Web sites of the operating systems you are interested in. You can gauge user's opinions by searching the relevant discussions in newsgroups and mailing list archives. Deja - http://deja.com and eGroups - http://egroups.com are good examples. I will leave this fan research to the reader.

1.2.1 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 9am to 5pm working hours like many conventional businesses you know. You are open 24 hours a day. You cannot afford to be off-line, for 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 which you can find and fix. There are OSs which won't work unless you reboot them twice a day. You don't want to use the OS of this kind, no matter how good the OS' vendor sales department. Do not follow flushy advertisements, follow developers advices instead.

Generally, people who have used the 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. You may need to become familiar with patching and compiling your OS. It's easy.

1.2.2 Memory Management

You want an OS with a good memory management, some OSs 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 10Mb and you have tens of these running, it definitely adds up!

1.2.3 Memory Leaks

Some OSs 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 is not then available for any purpose until the process which requested it dies. We cannot afford such leaks. A single mod_perl process sometimes serves thousands of requests before it terminates. So if a leak occurs on every request, the memory demands could become huge. Of course our code can be the cause of the memory leaks as well (check out the `Apache::Leak` module on CPAN). Certainly, we can reduce the number of requests to be served over the process' life, but that can degrade performance.

1.2.4 Sharing Memory

We want an OS with good memory sharing capabilities. As we have seen, if we preload the modules and scripts at server startup, they are shared between the spawned children (at least for a part of a process' life - memory pages can become "dirty" and cease to be shared). This feature can reduce memory consumption a lot!

1.2.5 Cost and Support

If we are in a big business we probably do not mind paying another $1000 for some fancy OS with bundled support. But if our resources are low, we will look for cheaper and free OSs. Free does not mean bad, it can be quite the opposite. Free OSs can have the best support we can find. Some do. It is very easy to understand - most of the people are not rich and will try to use a cheaper or free OS first if it does the work for them. Since it really fits their needs, many people keep using it and eventually know it well enough to be able to provide support for others in trouble. Why would they do this for free? One reason is for the spirit of the first days of the Internet, when there was no commercial Internet and people helped each other, because someone helped them in first place. I was there, I was touched by that spirit and I am keen to keep that spirit alive.

But, let's get back to our world. We are living in 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, and blame them for any problem. Your boss wants to be able to sue someone if the project has a problem caused by the external product that is being used in the project. If you buy a product and the company selling it claims support, you have someone to sue or at least to put the blame on.

If we go with Open Source and it fails we do not have someone to sue... wrong--in the last years many companies have realized how good the Open Source products are and started to provide an official support for these products. So your boss cannot just dismiss your suggestion of using an Open Source Operating System. You can get a paid support just like with any other commercial OS vendor.

Also remember that the less money you spend on OS and Software, the more you will be able to spend on faster and stronger hardware.

1.2.6 Discontinued Products

The OSs in this hazard group tend to be developed by a single company or organization.

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 which takes advantage of some proprietary features of the OS and which 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 which supplies your beloved OS goes bankrupt (not unlikely nowadays), or they produce a newer incompatible version and they will not support the old one (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...

Everyone can be hit by this mini-disaster so it is better to check the background of the company when making your choice. Even so you never know what will happen tomorrow - in 1980, a company called Tektronix did something similar to one of the Guide reviewers with its microprocessor development system. The guy just had to buy another system. He didn't buy it from Tektronix, of course. The second system never really worked very well and the firm he bought it from went bust before they ever got around to fixing it. So in 1982 he wrote his own microprocessor development system software. It didn't take long, it works fine, and he's still using it 18 years later.

Free and Open Source OSs are probably less susceptible to this kind of problem. Development is usually distributed between many companies and developers, so if a person who developed a really important part of the kernel lost interest in continuing, someone else will pick the falling flag and carry on. Of course if tomorrow some better project shows up, 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 OSs you can have the source! So you can always have a go yourself, but do not under-estimate the amounts of work involved. There are many, many man-years of work in an OS.

1.2.7 OS Releases

Actively developed OSs 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, 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 backwards compatibility with previous products of the same family, you might not reap all the benefits of the new product's features. Perhaps you get almost the same functionality for much less money if you were to buy an older model of the same product.

1.3 Choosing Hardware

Sometimes the most expensive machine is not the one which 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.

In the discussion we use terms that may be unfamiliar to some readers:

- 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 'fail-over' 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 will simply not dispatch the requests to the machine that was taken down.

- Load balancing - users are given the name of one of your machines but perhaps it cannot stand the heavy load. You can use a clustering approach to distribute the load over a number of machines. The central server, which users access initially when they type the name of your service, works as a dispatcher. It just redirects requests to other machines. Sometimes the central server also collects the results and returns them to the users. You can get the advantages of clustering too.

  There are many load balancing techniques. (See High-Availability Linux Project for more info.)

- NIC - Network Interface Card. A hardware component that allows to connect your machine to the network. It performs packets sending and receiving, newer cards can encrypt and decrypt packets and perform digital signing and verifying of the such. These are coming in different speeds categories varying from 10Mbps to 10Gbps and faster. The most used type of the NIC card is the one that implements the Ethernet networking protocol.

- RAM - Random Access Memory. It's the memory that you have in your computer. (Comes in units of 8Mb, 16Mb, 64Mb, 256Mb, etc.)

- RAID - Redundant Array of Inexpensive Disks.

  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 array 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. See for example the Linux 'HOWTO' documents Disk-HOWTO, Module-HOWTO and Parallel-Processing-HOWTO.

1.3.1 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 guest book, any old 486 machine could do it. 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 webserver scalability more precisely, the important thing is to make sure that you can add more power to your webserver(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 OSs that can talk to other machines and become a 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 :)

1.3.1.1 Single Strong Machine vs Many Weaker Machines

Let's start with a claim that a four years old processor is still very powerful and can be put to a 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 ten older but very cheap machines. I claim that with ten old machines connected into a cluster and by deploying load balancing you will be able to serve about five times more requests than with one single new machine.

Why is that? Because 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 you will have it anyway, for in a short time the new machine you have just bought might not stand the load. Then you will have to purchase more equipment and think about how to implement load balancing and web server file system distribution anyway.

Why I'm so convinced? Look at the busiest services on the Internet: search engines, web-email servers and the like -- most of them use a clustering approach. You may not always notice it, because they hide the real implementation behind proxy servers.

1.3.2 Internet Connection

You have the best hardware you can get, but the service is still crawling. Make sure you have a fast Internet connection. Not as fast as your ISP claims it to be, but 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!

The idea of having a connection to **The Internet** is a little misleading. Many Web hosting and co-location 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 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.
```

1.3.3 I/O Performance

If your service is I/O bound (does a lot of read/write operations to disk) you need a very fast disk, especially if the you need a relational database, which are the main I/O stream creators. So you should not spend the money on Video card and monitor! A cheap card and a 14" monochrome monitor are perfectly adequate for a Web server, you will probably access it by `telnet` or `ssh` most of the time. Look for disks with the best price/performance ratio. Of course, ask around and avoid disks that have a reputation for headcrashes and other disasters.

You must think about RAID or similar systems if you have an enormous data set to serve (what is an enormous data set nowadays? Gigabytes, Terabytes?) or you expect a really big web traffic.

Ok, you have a fast disk, what's next? You need a fast disk controller. There may be one 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.

1.3.4 Memory

Memory should be well tested. Many memory test programs are practically useless. Running a busy system for a few weeks without ever shutting it down is a pretty good memory test. If you increase the amount of RAM on a well-tested box, use well-tested RAM.

How much RAM do you need? Nowadays, the 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 which 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 your servers can grow to, 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, and 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 plan not only 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 came in. (This is called the Slashdot effect, which was born at http://slashdot.org). If you are about to announce something cool, be aware of the possible consequences.

1.3.5 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 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 which can be serious in certain circumstances. Try not to get one of them.

1.3.6 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 under-utilized, usually it will 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, 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.

Use a fast NIC that does not create a bottleneck. They are cheap. If the NIC is slow, the whole service is slow. This is a most important component, since webservers are much more often network-bound than they are disk-bound!

1.3.6.1 Solving Hardware Requirement Conflicts

It may happen that the combination of software components which 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 mod_perl server, it can be wise to put the two on different machines, since while RDBMS need a very fast disk, mod_perl processes need lots of memory. So by placing the two on different machines it's easy to optimize each machine at separate and satisfy the each software components requirements in the best way.

1.3.7 Conclusion

To use your money optimally you have to understand the hardware very well, so you will know what to pick. Otherwise, you should hire a knowledgeable hardware consultant and employ them on a regular basis, since your needs will probably change as time goes by and your hardware will likewise be forced to adapt as well.

1.4 Maintainers

Maintainer is the person(s) you should contact with updates, corrections and patches.

- Stas Bekman <stas (at) stason.org>

1.5 Authors

- Stas Bekman <stas (at) stason.org>

Only the major authors are listed above. For contributors see the Changes file.

Table of Contents: