close window

IBM Systems MAGAZINE

Cover Story

Print

Linux Performance Tuning

Getting the most from your Linux investment

February | March 2007 | by Jaqui Lynch

This is the first article in a two-part series. The second installment will run in the April issue of IBM Systems Magazine, Open Systems edition.

As time goes by, it's becoming apparent Linux* isn't going away. As companies move more than just mail servers and file servers onto their Linux systems, it becomes crucial to be able to tune these systems. Although third-party tools are available to help monitor and tune, it's important for the systems administrator to understand the free tools that come with Linux, as many of the other tools depend on these. It's vital to note performance and capacity must be managed just as they must on other platforms. Linux is running mission-critical workloads throughout the world and needs to be properly set up and managed to be reliable and perform well. Although the commands and discussion in this article are based on Red Hat Linux, they also apply to other distributions. This article will outline only CPU; the second installment will focus on I/O and memory. Future articles will cover network and the use of logical volume manager.

Performance 101

To analyze and improve performance, administrators must take several steps:

1. Determine the problem. First we need to understand what problem we're trying to solve. If there's no problem, why are we dedicating time and resources to this issue?
2. Take a baseline measurement. With any system, it's helpful to have a baseline measurement showing what the system looks like when it's behaving normally. This gives us something to measure against when the system is having problems.
3. Plan changes and predict results. Before making any changes, it's important to map them and note what results to expect.
4. Make changes. These can be made temporarily by issuing a command or, in some cases, by using the cat command to overwrite the current kernel value. Permanent changes normally are made in /etc/sysctl.conf or in /etc/rc.d/rc.local in Red Hat Linux.
5. Rerun the measurement tools to get a new set of data after the changes. 6. Compare the measurement with the previous results and with expected results.
6. Understand what happened and determine if there were any differences.

It's important to have this kind of structure for a successful measurement-and-tuning process. Too often, administrators miss or skip some of the steps. For the purposes of this article, it's assumed they're being followed. One starting tip is to turn off all services not being used - chkconfig -list shows services - and change the default boot mode from 5 to 3 (in /etc/inittab). Changing the default boot mode will remove the xWindows console, which uses memory and processor for no good reason. Any time you want to use xWindows, you can still bring up an X session.

Now let's review memory, the processor scheduler, the I/O scheduler, some issues around memory and some useful commands. For words to know, see "Terminology".

Memory in Linux

Linux uses an always-full model for memory, which is divided into two parts: Kernel space is memory reserved for the kernel and is used for device drivers, scheduling and to keep the kernel in memory; user space is memory reserved for application code. This includes the buffer space used to buffer I/O for disks and network, and memory used for direct memory access (DMA) transfers between devices.

Linux uses demand paging, which means the virtual address space is created empty and all pages in it are marked not present. When we access a page to work with it, it will have been marked not present, so a page fault occurs. The kernel intercepts the page fault, loads the page into the virtual address space and updates the accessed flag. Page faults can be soft, where the page is in real memory but not in the virtual address space; or hard, where the page has to be read in from disk. The sar -B command can be used to monitor these.

Paging happens when a process needs to put something into page space and no pages are available, so a page is borrowed from another process. If the borrowed page is dirty or is from working storage rather than persistent storage, the page must be written out to disk in the swap space before the new page can be brought in. This is an expensive page fault. The alternative occurs when the old page hasn't been updated and is backed by a file system. When this happens, the stealer simply takes that page and overwrites it.

Swapping is when an entire page set is paged out for a process. In Linux, we use paging rather than swapping, and the decisions are based on a least recently used method. Two bits are important in paging: the access bit and the dirty bit. The access bit indicates whether this page was accessed since the page scanner last checked it. The dirty bit indicates whether this page has been updated since it was last paged in. Use the kswapd command to inspect these two bits and clear the access bit or take action based on the settings of these two bits. The swapon -s command is used to list swap spaces and their use. Several sar flags (-g, -p, -w) also provide more information.

x86 Memory

For a 32-bit kernel, Linux has a maximum of 4 GB per process virtual address space. This increases to a theoretical limit of 2**64 with a 64-bit kernel. Memory use is somewhat limited in the 32-bit systems, as IA32 only has 32 bits for virtual addresses: 10 bits are for the page directory, 10 bits are for the page table entry and 12 bits are for offsets. Memory is broken down into three zones for the 32-bit kernel: DMA, normal and high memory (greater than 896 MB). The first 896 MB of real memory is for the kernel's address space. The kernel uses the next 128 MB for operations such as vmalloc and kmap. Basically, even on large-memory 32-bit x86 machines, the kernel needs to copy pages from high memory into that 128 MB window to work on them, then copy them back into other memory. This problem goes away in 64-bit and is a strong argument for migrating to 64-bit if possible.

Large pages are one new option in the 2.6 kernel and some of the 2.4 kernels. The translation lookaside buffer (TLB) is the processor's cache of virtual-to-physical memory translations. This speeds memory access. A TLB miss can be very expensive in terms of processor cycles, so it's important to avoid these. One method is to turn on huge TLB support by building it into the kernel. This allows a TLB entry to map 2 MB and 4 MB pages, reducing TLB misses. Administrators can monitor the current setting by looking at /proc/sys/vm/nr_hugepages.

Processor Scheduling

The dispatcher schedules processes and threads identically. The 2.6 kernel scheduler is a multiqueue scheduler where each processor has its own run queue. Priorities range from 0 to 140 as follows:

```
Pri 0-100     = realtime tasks
Pri 101-140   = normal jobs (nice -20..19)
```

CPU Basics

In Linux, as with most *NIX, there are seven major resource types administrators must monitor and tune:

- CPU
- Memory
- Disk space and arms
- Communications lines
- I/O time
- Network time
- Application programs

Users only see total execution time, so we'll start by looking at that. Total execution time, from a user's perspective, equals wall-clock time, or the time from when they hit "Enter" until they get a reply. At a process level, this is measured by running the time command. This provides a user with real time (wall clock), user-code CPU and system-code CPU. If user + sys > 80 percent, there's a good chance the system is CPU constrained. The components of total running time include:

- User-state CPU - The amount of time the CPU spends running the user's program in the user state. It includes time spent executing library calls but not time spent in the kernel on its behalf. This value can vary greatly depending on optimization use at compile time and on efficient code.
- System-state CPU - The amount of time the CPU spends in the system state on behalf of this program. All I/O routines require kernel services. The programmer can affect system time by changing the blocksize for I/O transfers.
- I/O time and network time - These are measures of time spent servicing I/O requests and moving data.
- Virtual memory performance - This includes context switching and paging.
- Time spent running other programs - This is when the system isn't servicing this application because another application has the CPU.

Information on the processor is included in /proc/cpuinfo or can be obtained from the output of the dmesg command. For example, my Pentium3 450mhz Dell Dimension machine shows the following in /proc/cpuinfo:

```
Processor     : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 7
model name    : Pentium III (Katmai)
cpu MHz       : 448.972
```

```
cache size      : 512 KB
```

Information on processor use can come from several sources, including vmstat, iostat, sar -u and mpstat. Because some of these commands aren't automatically included, you'll need to download the sysstat rpm from the Linux vendor so you can use those commands. Commands in sysstat will include sar, mpstat, iostat and several others. I always install this rpm on Linux systems.

For More Information

There's a great deal of information to glean from the system with relatively minimal effort. This information can then be used to determine what's happening with the system.

Additionally, there are free tools available, including:

- Ganglia is great for clusters and grids and provides central reporting (http://ganglia.sourceforge.net).
- Nmon runs on AIX* and Linux and can be used with the nmon analyzer to produce useful spreadsheets - such as Excel spreadsheets - that analyze and summarize performance (www.ibm.com/collaboration/wiki/display/WikiPtype/nmon).
- Sysstat rpm is required to get the iostat, mpstat and sar commands.

To properly measure performance, it's helpful to automate the reporting process using a scripting language, such as perl, combined with scheduling commands like at or cron. Programmers also can use these languages to create graphic views of the output from the tools. By using these tools and this method, it's possible to diagnose performance problems on most UNIX* systems using nonproprietary tools that come standard with the system. The follow-up to this article in the April/May issue will focus on I/O and memory tools useful for optimizing Linux performance.

Terminology

There are some differences and many similarities between Linux* and *NIX terms. A few important terms include:

Task - A descriptive term for a generic piece of work, regardless of whether it's a process or thread.

Process - A full data structure that may contain several threads.

Thread - A lightweight task. It can share everything with its parent or be more independent.

Kernel thread - A thread that only runs in kernel mode and performs kernel functions.

Context switch - Takes place when a process or thread has to be switched out so another can be dispatched.

References

- www.ibm.com/developerworks/linux/library/l-async/index.html
- www.ibm.com/developerworks/linux/library/l-scheduler/index.html
- www.ibm.com/developerworks/db2/library/techarticle/dm-0509wright/index.html
- http://people.redhat.com/alikins/system_tuning.html