close window

IBM Systems MAGAZINE

Administrator

Print

Linux Performance Tuning

April | May 2007 | by Jaqui Lynch

Note: This is the second article in a two-part series. The first installment was published in the February/March issue.

In last issue's article I introduced basic Linux* performance-tuning concepts. Now I delve into I/O and memory tools and commands that are useful for optimizing Linux performance.

Uptime

Uptime provides the analyst with the system load average (SLA). The SLA can only be used as a rough indicator as it doesn't take scheduling priority into account. It also counts as runnable all jobs waiting for disk I/O, including network file system (NFS) I/O. However, uptime is a good place to start when trying to determine whether a bottleneck is CPU or I/O based.

Running uptime provides three load averages: the last minute, the last five minutes and the last 15 minutes. The load average should be divided by the number of processors. If the calculated value is borderline, but has been falling over the last 15 minutes, then it would be wise to just monitor the situation. However, a value between 4 and 7 is fairly heavy and means that performance is being negatively affected. Above 7 means the system needs serious review and below 3 means the workload is relatively light. If the system is a single-user workstation, then the load average should be less than 2. The ruptime command also allows you to request uptime information remotely.

mpstat

The mpstat command shows a breakdown of utilization by an individual processor that's very useful on a multiprocessor system. The command shows the kernel level, user CPU, system CPU, nice time, idle time, wait time and interrupts per second. Similar data can be obtained with the sar command.

sar

The sar command provides a good alternative to uptime with the -q option or mpstat with the –u option. This command provides statistics on the average length of the run queue, the percentage of time the run queue is occupied, the average length of the swap queue and the percentage of time the swap queue is occupied.

The run queue lists jobs that are in memory and runnable, but doesn't include jobs that are waiting for I/O or sleeping. The run queue size should be less than 2 per processor. If the load is high and the runqocc=0, then the problem is most likely memory or I/O, not CPU.

The swap queue lists jobs that are ready to run but that have been swapped out.

The powerful sar command is run by typing:

sar -options int #samples

Where valid options generally are: -g or -p: paging, -q: average Q length, -u: CPU usage, -w: swapping and paging, -y: terminal activity, -v: state of kernel tables.

One advantage of using sar is you can write the data to a file and then post-process it when the system isn't busy. The file is created using the -a and -o flags. An example of creating a file of 30 snapshots, each two seconds apart, would look like:

sar a o sardata.out 2 30

This file would then be processed using:

sar -u -f sardata.out

This would cause the CPU usage report to display from the recorded file.

iostat

After determining that the problem may well be CPU-based by using uptime and sar, it would then be necessary to move on to iostat to get more detail. Running iostat provides much information, but the values of concern are %user and %sys. If (%user + %sys) > 80 percent over a period of time, then it's likely the bottleneck is CPU. In particular, it's necessary to watch for average CPU being greater than 70 percent with peaks above 90 percent. It's also possible to get similar information by running the ps -au or sar -u commands. Both commands provide information about CPU time. The sar -u command, in particular, breaks the time into user, system, time waiting for blocked I/O (i.e., NFS, disk, etc.) and idle time.

Once it's been determined that the problem is a CPU bottleneck, there are several options to consider. It's possible to limit the CPU time a process can use with the limit command. If the problem relates to one process, then it's also possible to model or profile that process using one of the profiler commands to find out whether optimizing the program code is a possibility.

Running the time command also provides a good indication of whether the process is CPU-intensive. Compiler options have been proven to extensively affect the performance of CPU-intensive programs and turning on optimization, if you're able to do so at compile time, is worthwhile.

If none of the aforementioned options improves performance and the problem still appears to be CPU, three options remain:

- Run the workload on another more powerful system
- Reduce the priority of the process (use nice or renice)
- Upgrade the CPU

I/O Scheduling

The 2.6 kernel introduces a concept called I/O elevators and provides four I/O schedulers

(CFQ, NOOP, anticipatory and deadline). The choice of I/O scheduler can greatly impact I/O performance. An I/O elevator is a queue where I/O requests are ordered based on where their sector is on the disk. I/O elevators are included in the deadline and anticipatory I/O schedulers.

The CFQ (completely fair) scheduler is the default on Red Hat Fedora Core 6. Check what you're using by running dmesg and scanning through the output. Different tunables for schedulers can be found in: /sys/block/device/iosched (i.e., /sys/block/sda/queue is the directory for the CFQ scheduler tunables on disk sda).

Using 64 internal I/O queues and a single I/O dispatch queue, the CFQ scheduler targets the fair allocation of I/O bandwidth among the initiators of I/O requests.

The NOOP scheduler is a minimal-overhead scheduler that has been shown to be useful for large I/O subsystems that use RAID controllers. It's worth trying this out if the server is connected to a large RAID storage-area network (SAN).

The anticipatory scheduler is targeted at reducing the per-thread read response time and tries to assist synchronous sequential reads especially during streamed writes. It's not designed for random read workloads with many seeks all over the disk; it will perform poorly if used for this kind of workload.

The deadline scheduler uses five I/O queues to keep track of I/O. It's designed to emphasize average read-request response time for workloads that seek all over the disk.

I/O Problems

If the problem doesn't appear to be CPU, then it's necessary to investigate memory and I/O as possible causes. Again, it's possible to use iostat or sar to get the information needed. The 2.6 kernel provides three new flags on the iostat command: -p, -k and –x. The –p flag adds a column for %iowait into the processor statistics. It also tracks time spent waiting for the Hypervisor (%steal). The -k flag provides KB/sec written and read instead of blocks. Additionally you can use the tps field as an indicator of logical I/Os per second. Finally the –x flag provides detailed performance statistics such as read and write service times.

iostat provides data on several important values for each physical disk. These values include percent of time the physical disk was busy, kilobytes per second to/from the disk, transfers per second to/from, kilobytes read and kilobytes written. This data will help determine if an imbalance of I/O exists among the physical volumes. If all appears normal, the next step is investigating the filesystems at which the I/O is directed. If the I/O is directed at the page files, then memory must be investigated.

If the bulk of the I/O (more than 30 percent) is going to a logical volume or filesystem not used for paging, then the problem is most likely user I/O. This can be resolved by:

- Checking fragmentation
- Reorganizing the filesystem
- Adding physical volumes
- Splitting the data
- Adding memory

Paging is investigated using vmstat, iostat, sar and swapon. Information on the page spaces available including physical volume and utilization is provided by swapon -s. The tool vmstat provides information on actual memory, free pages, processes on the I/O waitq,

pageins, pageouts, etc. It doesn't provide timestamps. It's also possible to use the sar -r, -b or -w commands.

The free -m command shows memory usage broken down into the following: total memory, used memory, free memory, buffers, file-cache size (cached), total size of swap space, used and free swap space. This information can also be obtained by using: more/proc/meminfo.

Sequential readahead can help sequential I/O performance. Previously we would use hdparm or change the values in /prop/sys/vm. For the 2.6 kernel use blockdev:

1. To get the current readahead value:

blockdev --getra /dev/sda

2. To set the readahead value temporarily:

blockdev --setra 4096 /dev/sda

(Note: 4096 is just an example. Testing determines the optimal value.) The OS will read ahead 4096 pages and throughput will be higher. To make the change available every time you boot, add: blockdev --setra 40964 /dev/sda, blockdev --setra 4096 /dev/sdb, etc., to /etc/rc.d/rc.local. The blockdev command also has other options, so examining the main pages is recommended.

Figure 1 shows the impact of small blocksizes on sequential performance. The move from a 4 KB blocksize to a 16 KB blocksize increased bandwidth from 18.6 MB/sec to 1 GB/sec.

If there's no performance problem with the CPU and the system isn't disk or memory bound, it becomes necessary to investigate the network to check whether it's remote-file I/O bound. This is the last step before running the more resource-heavy trace command to determine what's really happening.

The Diagnosis

A great deal of information can be gleaned from the system with relatively minimal effort. This information can then be used to determine what's happening with the system. Additionally, free tools such as nmon and ganglia allow you to get a full view of the system in one page. Running nmon in logging mode creates a file that can be used by the nmon analyzer (Excel spreadsheet) to produce graphs of what's happening on the system.

To properly measure performance, it's helpful to automate the reporting process using a scripting language (e.g., perl) combined with scheduling commands (e.g., at or cron). These languages can also be used to create graphical views of the output from the tools. By using these tools and this methodology, it's possible to diagnose performance problems on most UNIX* systems, using non-proprietary tools that come standard with the system.

References

- http://linuxperf.nl.linux.org (This is great, but it's in Dutch.)
- www.psc.edu/networking/perf_tune.html
- Enabling high performance data transfers on hosts: www.ibm.com/developerworks/linux/library/l-async/index.html
- www.ibm.com/developerworks/linux/library/l-scheduler/index.html
- DB2* memory and file caching: www.ibm.com/developerworks/

db2/library/techarticle/dm-0509wright/index.html
- Performance Tuning for Linux* Servers – Johnson, et al. ISBN 0-13-144753-x
- www.ibm.com/redbooks, search on Linux performance: Red Hat is Redpiece 3861, Novell SUSE is 3862
- http://people.redhat.com/alikins/system_tuning.html

Additional Tools

1. **Ganglia**: This tool is great for clusters and grids and provides central reporting (http://ganglia.sourceforge.net).

2. **Nmon**: This tool runs on AIX* and Linux* and can be used with the nmon analyzer to produce useful spreadsheets that analyze and summarize performance (www.ibm.com/collaboration/ wiki/display/WikiPtype/nmon).

3. **Sysstat rpm**: This is required to get the iostat, mpstat and sar commands.