PETER BAER GALVIN

Pete's all things Sun: Solaris System Analysis 102

Peter Baer Galvin is the Chief Technologist for Corporate Technologies, a premier systems integrator and VAR (www.cptech.com). Before that, Peter was the systems manager for Brown University's Computer Science Department. He has written articles and columns for many publications and is coauthor of the *Operating Systems Concepts* and *Applied Operating Systems Concepts* textbooks. As a consultant and trainer, Peter teaches tutorials and gives talks on security and system administration worldwide. Peter blogs at http://pbgalvin.wordpress.com and twitters as "PeterGalvin."

pbg@cptech.com

RECENTLY I WAS HELPING OUT A FRIEND, a CEO at a small business, who had her main system running without a backup. As we all know, friends don't let friends compute without backups. Given that the system was an Apple Mac, it was a trivial matter to attach an external drive and push the couple of buttons needed to execute a backup. When I was done she was rather surprised and asked if that was all there was to it. Was computer administration really that easy? After pondering a second I came up with a fundamental statement about system administration: It's easy, except when something goes wrong, and then it can be very, very challenging.

For a sysadmin, a good day can turn into a very bad day with just a few words: "The system has a problem." Such problems, especially ones of performance or reliability, can be difficult to solve. In fact they can be the most difficult task a sysadmin performs.

The goal of the previous "Pete's All Things Sun" column, "Solaris System Analysis 101," was to put a stake in the ground about the first steps that should be taken when a system has "a problem." The hope is that you, the sysadmin reader, will contribute to it, creating a consensus document. Given that we live in the time of Web 2.0, a wiki seemed like the best way to foster contributions, and that wiki is now live [1]. Please have a look and contribute your wisdom and knowledge for the betterment of sysadmin-kind.

That leads us to this column, "Solaris System Analysis 102." Once the 101 steps are taken, what can be done to determine the specific cause of the problem and fix it? The previous column was mostly operating-system-independent. Almost all of the ideas there apply to all operating systems equally. In this column that will not be the case. Here, then, are specific steps I use to analyze a Solaris system and determine the cause of the problem. Most of these commands are Solaris-specific, including DTrace code. This column will also be added to the wiki, allowing you to comment, correct, and expand. Please do so! In the future, watch for BoFs and other activities at USENIX conferences about this topic.

Phase 1: Search for the Smoking Gun

Sometimes the system has a large, easy-to-find problem. In those cases it would be a shame to spend a lot of time chasing down complex paths. Rather, the first step is to check for obvious problems with the "usual suspect" commands. The goal of this phase is to narrow the problem area to a specific aspect of the system.

Solaris System Analysis 101 ended with a list of areas to explore. Here are some more specifics:

- Scan through log files such as /var/adm/messages and via `dmesg`. Don't ignore anything odd: It could be the canary indicating the problem.
- Run `svcs -a` to check for services that have failed or are disabled.
- Check for full disks or changed mount information via `df` and `mount`.
- Run `ifconfig -a` and look for any errors; run `kstat` and read through the section of output of a given network interface (such as e1000g0) to check network parameters such as duplex and speed.
- Read through /etc/system and look for settings copied from other systems or left behind during an upgrade. /etc/system should never be copied or left intact between operating system or application upgrades; such events should cause an audit of the file for entries to remove or update. Check the *Solaris Tunable Parameters Reference Manual* [2]. This document is updated for every Solaris release. Watch out for system setting recommendations from vendor documents.
- Check /etc/projects for any resource management settings that could be affecting system or application performance.
- Check the load average of the system. `uptime` shows the 5-, 10-, and 15-minute average number of threads wanting to run on the system. If those numbers are significantly (two times or more) higher than the number of cores in the system, users will report "slowness."
- Check the stat commands and look for anomalies. Note that the first set of output is averages per second since the system booted. The following sets are averages per second since the previous set of output. As with all of these commands, understanding the output and the underlying system is key.
- Check `iostat -x 10` and check the `svc_t` column for large service times (in milliseconds). Anything above 30 ms can be of concern. Also note that dividing kilobytes written per second by writes per second produces the average write size during that period, which can help when analyzing I/O issues. The same applies to the read values (`r/s` and `kr/s`).
- Check `mpstat 10`: How was processor time spent? Per CPU (each row being a CPU's status), what percentage of time was spent in user-land (running user code) (`usr`), how much in the kernel (`sys`), and how much idle (`idl`)? Most time should be `usr`, and any more than a few percent in the kernel can indicate a problem.
- Check `vmstat 10`: How many threads are running or want to run (`kthr r`), how many are blocked waiting for something (usually I/O) (`kthr b`), and how many processes have been swapped out (`kthr s`)? Swapped out means that the system was desperately short of memory and booted entire processes out to disk. That's bad. Also check `page sr`, the scan rate, to see whether the system is short of memory. The larger this number, the more the system is hunting for memory. Anything above 0 is considered a memory shortage. Memory is orders of magnitude faster than disk, so any use of disk as virtual memory can cause a system slowdown.
- Check `vmstat -p 10`. This shows system-wide memory operations. This is the place to check whether the system is short on memory and to

determine which system aspect is using the memory [executable process pages, anonymous (heap, stack, or malloc) uses, or file system I/O].

- Check `prstat`. If the problem is simply processes using up CPUs, then `prstat` can show which processes those are. What is more difficult is figuring out what the process is doing and whether it should be doing it.
- Check `prstat -Lmp <pid>`. This shows detailed state information about a specific process at the current time. If the process has multiple threads it shows a row per thread. Columns 3 through 9 (`USR` through `SLP`) add up to 100%, showing the percentage of time the thread spent running in user mode (`USR`), in kernel mode (`SYS`), and so on.
- Use `pmap -x <pid>`to explore the memory map of a problem process.
- Use DTraceToolKit and DTrace scripts to look at specific suspect aspects of the system.

Phase 2: Finding the Owner of the Gun

With the Phase 1 rough data in hand, did you find the problem? User-level problems are relatively easy. If a process is using too much CPU or memory and you have the source code, it is now a program development and debugging problem. If the application is well written, then perhaps the only solution is adding resources to the system to allow the application to match your performance needs. For home-grown code, be sure to use the latest version of a given compiler. Also note that Sun's SunStudio development environment is now available [3] for free (without support), generates great code, and has good debugging tools built in, including the DTrace-based D-light tool and "performance analyzer" functionality. Also, at least with Solaris, each release usually brings about performance improvements. If you are running an older version of Solaris, consider the (difficult) step of upgrading. In addition, Java code is a major component in many applications, and Java can be difficult to performance-analyze and tune. Try to use the latest JVM, especially because Java 1.5 adds DTrace support and Java 1.6 automatically optimizes garbage collection.

If the problem is at the system level, then more time (and commands) may be needed to track down the problem. The good news is that Solaris 10 has many more tools than previous Solaris releases (and other operating systems in general) to find and fix these problems.

Some higher-level system areas to consider include:

- Are you running the most appropriate scheduler for each system in your environment? Solaris defaults to time-share scheduling for user processes. If your system is a server that doesn't run general user tasks, then time-sharing is overkill with more overhead. If you want all processes on the system to have the same priority (not changing as time-sharing does based on CPU used and I/O requested), then consider changing to the much lower-overhead fixed priority scheduler "FX." Such a change could buy you 5% or more CPU time. To make FX the default class execute `dispadmin -d FSS`. That change is persistent across reboots. To move current processes from time-sharing to FX, use `priocntl -s -c FX -i class TS`.
- If the problem involves some processes starving others of resources, consider implementing the fair-share scheduler and resource management. Those can be implemented either for the full system or, more easily, per-zone when zones (a.k.a. containers) are installed on a system. There is a lot to resource management and zones, as has been covered previously in *;login:*. The slides from my tutorial on Solaris 10 adminis-

tration have all the gory details and are freely available online [4]. There are links to this and other resources at my blog [5].

- If there are high-priority processes on a system, consider "pinning" them to a set of CPUs. These processes will stay on those CPUs and not be rescheduled or interrupted. A good time to use this technique is for database servers or just for the log-writing process of a database. The Solaris tools to use here are processor sets and process bindings.
- Are you using the best-fit page sizes? Having the same sizes of I/O operations from memory through to the physical disk is one key to good I/O performance. For example, OLTP databases such as Oracle's frequently perform I/O in 8-kB chunks. If you format your disks to use 8-kB sectors, I/O will be streamlined. Be sure to take into account the underlying disk structures (i.e., if you have a SAN, understand the I/O geometry within the LUNs that are provided). Note that terminology of disk structures varies, but ZFS calls its I/O chunk the "recordsize." In this Oracle example, set a ZFS recordsize to 8 kB, and, for good performance, make sure that the underlying storage array has RAID sets that are multiples of 8 kB. Jiri Schindler wrote a very in-depth analysis of matching application and device I/O patterns in his PhD thesis [6].
- Is your I/O well-balanced and spread across enough devices (e.g., disks and network ports)? In general, I/O is the most likely bottleneck, disk I/O the most likely I/O culprit, and individual disks the most limiting I/O device. Any given disk can perform 100 to 200 I/O operations per second (IOPS). If your system needs to do thousands of IOPS, then you need tens of disks, well tuned, to provide that I/O. RAID 0+1 and 1+0 are better-performing than RAID 5, so match the RAID level with the performance needed.
- Are you using the best CPU for the workload? Sun has two product categories: The first includes the "X" and "M" servers, which run a few threads very fast. The "T" servers are chip multi-threading (CMT) systems and run lots of threads, but run them rather slowly. An analogy can help sort out the best uses for these systems. Think of the "X" and "M" servers as race cars and the "T" servers as trucks. Each has its uses, so make sure you use the right system for the needed performance. Also, there are several steps that can be taken to determine whether a "T" server is right for your applications and to tune these servers. Sun's Web site [7] is the best place to start.

As always, benchmarking is the best way to test performance and performance changes, if the benchmarking is accurate and repeatable. Watch out especially for caching effects in benchmark efforts. Caching happens at all levels of computer systems, so, for example, it is safest to reboot the systems involved between each benchmark run. Consider, however, that SAN arrays also have caches, which could invalidate (or at least complicate) benchmark results.

Run Forensics on the Gun

Once the range of the problem has been narrowed, specific analysis can be done on the problem area to ferret out the source of the problem. DTrace is a fabulous tool for this analysis.

The DTraceToolkit provides over 200 prewritten (but unsupported) tools for getting detailed information about the operation of many areas of the system. Get familiar with the tools so they are in your arsenal when needed. The scripts are well documented and demonstrated online [8], so I won't repeat that information here.

Beyond the DtraceToolkit, the sky is the limit for delving into system activity details. For example, here is sample code to graph the time spent in each system call by each process:

```
syscall:::entry
/uid != 0/
{
self->tm = timestamp
}
syscall:::return
/self->tm/
{
@[execname, pid, probefunc] = quantize(timestamp - self->tm);
self->tm = 0
}
```

In another example, processes starting and exiting immediately can be difficult to spot and can greatly decrease system performance. Find them by the command line /usr/sbin/dtrace -n 'proc:::exec{printf("%s execing %s, , uid/zone =%d/%s\n",execname,args[0],uid,zonename)}'.

Another previously hidden performance hit is error management. Detect and fix failing system calls before moving forward, as that will change your performance picture. A DTraceToolkit tool, errinfo, displays all system call errors.

For I/O, to display files and the I/O being done to them execute /usr/sbin/dtrace -n 'io:::start{@[execname, args[2]->fi_pathname] = count()}'. To determine the block size execute /usr/sbin/dtrace -n 'io:::start{@[execname, args[2]->fi_pathname] = quantize(args[0]->b_bufsize)}.

To determine the level of multi-threading of the applications on the system execute /usr/sbin/dtrace -n 'profile:::profile-100hz /pid/{@[pid, execname] = lquantize(cpu, 0, 512, 1);}'.

Networking can also be a bottleneck, as even multiple 1-Gb links can be slower than other system aspects. Even with Solaris 10, network bottlenecks can be difficult to spot owing to the lack of a DTrace networking provider. That provider was included in Solaris Nevada build 93, so it should appear in a future Solaris release. For details see Sun's wiki [9]. In the meantime a good tool is nicstat, also available online [10].

If the information in this column helped determine the problem but didn't provide a solution to the problem, then it is time to drill down further into the specific problem area. The resources listed below should help with that.

Next Time

If the OpenSolaris Distribution (project Indiana) meets its release goals, then the first production release will be done before the next issue of *;login:*, and that should a rich topic for the next PATS column.

Resources

Very good information for drilling down into each Solaris area of performance tuning is available at http://www.solarisinternals.com/wiki/index.php/Solaris_Internals_and_Performance_FAQ.

A good paper about specific detailed aspects of Solaris performance problem resolution is "Performance Analysis Using DTrace," by Benoit Chaffanjon, at

http://opensolaris.org/os/project/sdosug/past_meetings/Performance_Analysis_Using_DTrace.pdf.

REFERENCES

[1] http://wiki.sage.org/bin/view/Main/AllThingsSun.

[2] http://docs.sun.com/app/docs/doc/817-0404.

[3] Sun's SunStudio development environment is available at http://developers.sun.com/sunstudio/.

[4] http://www.galvin.info/2006-11.s10admin.zip.

[5] http://pbgalvin.wordpress.com.

[6] http://www.pdl.cmu.edu/PDL-FTP/Database/CMU-PDL-03-109.pdf.

[7] http://www.sun.com/bigadmin/topics/coolthreads/.

[8] The best starting point for the toolkit is http://www.brendangregg.com/dtrace.html.

[9] http://wikis.sun.com/display/DTrace/ip+Provider.

[10] http://www.brendangregg.com/K9Toolkit/nicstat.