Performance Forensics

Bob Sneed—SMI Performance and Availability Engineering (PAE)

Sun BluePrints™ OnLine—December 2003

`http://www.sun.com/blueprints`

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
(650) 960-1300

Part No. 817-4444-10
Revision 06, 12/9/03
Edition: December 2003

Copyright 2003 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, California 95045 U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at http://www.sun.com/patents and one or more additional patents or pending patent applications in the U.S. and in other countries.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Sun BluePrints, SunDocs, Sun Explorer, SunSolve Online, Java, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the US and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

U.S. Government Rights—Commercial use. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the Far and its supplements.

Netscape is a trademark or registered trademark of Netscape Communications Corporation in the United States and other countries. Mozilla is a trademark or registered trademark of Netscape Communications Corporation in the United States and other countries.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2003 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95045 Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuels relatants à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuels peuvent inclure un ou plus des brevets américains énumérés à http://www.sun.com/patents et un ou les brevets plus supplémentaires ou les applications de brevet en attente dans les Etats-Unis et dans les autres pays.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

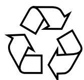
Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque enregistree aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company Ltd.

Sun, Sun Microsystems, le logo Sun, Sun BluePrints, SunDocs, Sun Explorer, SunSolve Online, Java, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

Netscape est une marque de Netscape Communications Corporation aux Etats-Unis et dans d'autres pays. Mozilla est une marque de Netscape Communications Corporation aux Etats-Unis et à d'autres pays.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.



Please
Recycle

Performance Forensics

The health care industry has well-established protocols for the triage, diagnosis, and treatment of patient complaints, while resolution of system performance complaints often seems to take a path that lacks any recognizable process or discipline.

This article draws from lessons and concepts of health care delivery to present ideas for addressing system performance complaints with predictable and accurate results. Specific tools from the Solaris™ Operating System (Solaris OS) are discussed.

Introduction

The treatment of illness and relief of discomfort has a wide and historic body of knowledge and practice. We call those who are skilled in applying such knowledge and practices doctors, and we trust them with our lives. In computer performance analysis, the knowledge base is younger, and the practice less developed. We call performance experts *gurus*, and we trust them with our computer systems.

How do gurus do what they do? It is not just a matter of expertise in using tools or being very smart. It is mostly that they think and work very much like doctors. Doctors share common elements of philosophy, education, experience[1], and established methods of problem solving. Modern medical practices ensure that patients and caregivers know their roles, and it proceeds according to relatively well-known processes and protocols. Analogies relating to medicine can provide a familiar framework and perspective for both *patients* and *caregivers* in resolving computer-performance issues and optimizing business computing objectives.

This article uses liberal doses of medical terminology and analogies as a means of discussing a variety of performance analysis topics. The first sections of this article are philosophical in nature, discussing some analogues between medicine and

1. Experience is the "Knowledge one possesses just *after* it was first needed." (Anon)

computer-performance analysis. Subsequent sections present practical information on applying these analogies with tools and techniques for performance troubleshooting and optimization in the Solaris OS environment.

Performance and Disease

Conspicuously broken bones are among the least challenging of medical problems from a diagnostic perspective. Similarly, component failures in computer systems do not usually present great diagnostic challenges. Clear markers from routine tests can sometimes reveal disease (for example, high blood pressure and high cholesterol), even though the patient has no current complaint. Similarly, good configuration management and capacity-planning practices can prevent complaints from arising with computer system performance complaints.

A bigger challenge for doctors is when the patient complaint is simply "I don't feel well," which is analogous to computer-user complaints of a system being "too slow." In either case, the underlying problem could range from grave to trivial or could even be psychosomatic. A disciplined approach and a wide breadth of knowledge are particularly valuable for getting to the root of the problem.

For medical complaints, the top diagnostic priority is to exclude potentially lethal or dangerous conditions. Events such as crashes, *hangs*, or other disruptive events correlate with the medical notion of *lethality*. Past that, medical diagnostic processes proceed in a predictable way, leading to a diagnosis and a treatment plan. Given that the elements of computer systems are far less complex and diverse than those found in the human body, there is no reason to believe that practices in troubleshooting computer systems cannot match or exceed those found in medicine—without requiring multiple millennia to mature.

Performance Forensics

The term forensics means "The science and practice of collection, analysis, and presentation of information relating to a crime in a manner suitable for use in a court of law." Ultimate resolution of a performance issue[2] can involve tuning, bug fixing, upgrading, product enhancement, or re-architecting the entire solution, but first, there needs to be a working diagnosis. The analytical process of diagnosis can resemble the process by which medical scientists and detectives explore the evidence, looking for clues.

2. A *crime* in this context.

When an issue is diagnosed for which there is no ready workaround or patch, the process of getting it repaired can require nothing less than proving the case, rigorously as one would in a court of law. Even the task of attributing the problem to a specific vendor or component can require rigorous methodology.

In thinking about where to start in the quest for improved process and efficiency in resolving performance issues, the place to start and end is clear: business requirements.

Business Requirements

Complex systems can produce a veritable flood of performance-related data. Systems administrators are naturally compelled to pore over these reams of data looking to find something wrong or to discover opportunities for tuning or for process improvement. All too often, the actual business performance metrics of the system are not formally included in the data, and not included in the process.

While subsystem tuning often leads to improved business throughput, it can occasionally backfire. For example, if improving the efficiency of one part of a system increases the burden on some other part of the system to the point where it becomes unstable, chaotic behavior can occur, possibly cascading through the entire system.

The best measure of any system change is its impact on business requirements; therefore, the principle objective of all tuning should be viewed in terms of business requirements. Business requirements may be expressed in terms of service level agreements (SLAs) or critical-to-quality[3] (CTQ) measures. SLAs and CTQs may comprise the substance of contractual agreements that feature penalties for non-conformance.

For the performance analyst, it is adequate to view business requirements in three principle categories:

- Performance

  Performance involves the primary metrics of how well a system is doing its job (for example, transactions per unit time, time per transaction, or time required to complete a specific set of tasks).

3. The term "critical-to-quality" is part of the Six Sigma techniques, which were developed primarily by General Electric.

- Predictability

  It is not enough for an average response time to be *sub-second* if there are too many outliers or if outliers are long enough to result in complaints of user-perceived outages. Long-running tasks are often required to predictably complete in their prescribed operational "window".

- Price

  Given an unlimited budget for time, equipment, and people, most business goals can be met[4]. Computer systems make good business sense only when they deliver the expected business value within budgets.

These three factors correlate directly with the famous engineering axiom: "Good, Reliable, Cheap—pick any *two.*" Additional business metrics of *headroom* and *availability* are often cited, but these are actually only variants of these three principle business metrics.

Headroom is the concept of having some extra performance capacity to provide confidence that a system's performance will remain adequate throughout its intended design life. Business management must have confidence not only that a system will do what it was designed to do, but also that it will be able to accommodate forecasted business growth. In addition, systems must not grossly malfunction when driven beyond their nominal design point. Encountering the need for unbudgeted upgrades is bad. Headroom is rarely indicated with any accuracy by simple metrics such as "percent idle CPU". The most effective means of assessing headroom is by test-to-scale and test-to-fail techniques.

Availability issues can be viewed as either performance or predictability issues, where performance drops to zero. Availability can also encompass performance issues such as cluster failover or reboot time.

In practice, making the right tradeoffs in these categories is the key to business success. Optimizing non-business metrics is not necessarily folly, but it can be wasteful to optimize factors that produce no gains in business terms[5].

Medical Analogues

It would be easy to dwell too long on colorful analogies between medicine and performance analysis, so we will try to be brief here. For some of the medical terms given here, only the medical image is discussed, leaving the reader to relate the principle to prevailing practices in performance analysis.

4. Business goals are sometimes laid out, but sometimes unattainable due to constraints of physics, such as the speed of light.
5. In the contemporary mythology of "Star Trek," the otherwise loathsome Ferengi aliens regard it as high crime to "engage in unprofitable pursuits."

Philosophy

Medical doctors usually subscribe to some variant of the Hippocratic Oath. It can be a very quick and worthwhile exercise to look up the oath on the Internet and give it a full reading. Upon reading the oath, most will be happy that doctors are guided by it, and many may wish for similar disciplines to spread to other fields.

Process and Protocol

From the writings of Hippocrates (around 400 BCE), it is clear that the distinct topics of *medical practice* and *medical science* have had a very long time to mature.

Modern doctors are subject to an enormous body of laws and ethical constraints, and they have well established paths for escalating patient issues. Much of the medical process is defined and monitored by medical professional organizations and license-granting agencies. Medical process and protocol are aimed at improving the timeliness and consistency of results, as well as towards managing costs.

In the computer industry, structured problem solving methodologies, such as the Kepner-Tregoe and Six Sigma methods, are seeing a great rise in popularity. Widespread adoption of these methods might rapidly mature the practice of computing to rival long-established medical processes in terms of timely positive outcomes and cost containment.

Roles

In Section One of *Aphorisms*, Hippocrates wrote that "*Life is short, and Art long; the crisis fleeting; experience perilous, and decision difficult. The physician must be prepared to do what is right himself, but also to make the patient, the attendants, and the externals cooperate.*" While the roles of the doctor and patient are certainly preeminent, the roles of nurses, emergency medical technicians, and administrative staff are also very well established.

In computing, there are often many diverse players invested in performance issues, ranging from executive management to the application owners and operations staff. Frequently, the scenario is further complicated by the involvement of multiple vendors. The performance analyst should be prepared to act as the physician, but there is rarely any clear protocol for coordinating the activities of all of these parties!

Primary Care

Total health care costs are minimized by a good program of preventative medicine. Thomas Edison is commonly quoted as forecasting that "*The doctor of the future will give no medicine, but instead will interest his patients in the care of the human frame, in diet, and in the causes and prevention of disease.*" Quality of life depends greatly on the skill of the primary health care provider, who should take a holistic approach to patient well-being.

Emergency Care

The first order of business in a medical crisis is to *stabilize* the patient. Then, the doctor must assess the proximate cause and potential severity of the patient's condition. Attention is focused on patients in rough order of the severity of their complaints during the process of *triage*. Priorities are also influenced by the likelihood of a favorable outcome. Detailed diagnosis and treatment is usually left to an appropriate specialist, while lesser complaints might be quickly solved with a bandage or some aspirin. The accuracy of the approximate diagnosis made during triage can have a significant impact on the overall time to resolution. Hospital emergency rooms deliver great consistency in handling crises, but are not geared towards delivering *holistic* or *total patient* care.

Science and Causality

There is some art in the accomplished diagnostician's practice of weighing clinical signs in the context of a patient's history and in the light of the diagnostician's medical training and experience. One might say that neither medical science nor computer science is an exact science. However, no matter if the science is exact or not, what distinguishes credible diagnostic hypotheses from guesswork is the notion of a plausible chain of causality.

In the absence of a clear chain of causality, guesswork may sometimes succeed at alleviating symptoms, but such guesswork does not tend to yield confidence that the correct issues have been identified and repaired. Just as the over-prescription of antibiotics can lead to insidious drug-resistant strains of disease, failure to make accurate diagnoses with computer systems can delay the identification and repair of latent issues. The motivations to make accurate diagnoses are strong, and the dangers of treating symptoms are universal. Of course, notwithstanding clear causality, proper diet and exercise will predictably lead to a reduced frequency of complaints.

Specialization and Collaboration

The complexities of medicine and computer technology each require specialization as a practical matter. The Hippocratic Oath essentially mandates collaboration as a corresponding ethical matter. While a licensed general practitioner of medicine will at least know what all of the specialties are, this is not so consistently the case among the general practitioners of computer technology.

So What?

The analogues between medicine and performance analysis are so sweeping that we may as well just borrow liberally from the medical language. That is just what we will do going forward.

Lab Tests and Record Keeping

Patients typically spend less time with the doctor than the doctor's organization spends on the patient's file. Indeed, patients are not generally allowed to actually see a doctor until their file is complete.

Lab Tests

Because medical diagnostic tests have real costs associated with them, they are ordered sparingly by doctors based on clinical signs. In computing, tests are cheap and easily automated so that vast quantities of data can be easily gathered.

Notwithstanding the low direct cost of collecting performance data, it should still be done only as clinical signs warrant. The main hazard of computer performance monitoring lies in the fact that it is not totally free. The mechanics of measurement can significantly skew the system being measured. This is sometimes referred to as *probe effect* or sometimes as a *Heisenburg effect*. Just ask anyone who has ever been wired up for a sleep study, and they will likely testify that it was among the worst night's sleep they ever had!

The degree to which a workload can be skewed by monitoring depends on the intrusiveness of the tools being used and the degree to which they are used. The impact of monitoring particular components of a very complex system varies depending on how the monitored component relates to other components in the overall work flow. Keeping the effect of measurement overhead aligned with

strategic goals and tradeoffs is part of the art of choosing appropriate instrumentation. There are various strategic motivations for use of performance monitoring tools:

- Health monitoring (establishing norms for a workload and using them to detect when operations become abnormal)
- Capacity planning (gathering data to help forecast when additional capacity will be needed)
- Gaining insight (discovering opportunities for optimization or formulating hypotheses for making high-level architectural decisions)
- Diagnosing (accurately discovering the root cause of a performance problem)

Tools and techniques used for health monitoring and capacity planning purposes should not be expected to provide much value in the pursuit of gaining insight or diagnosing. Conversely, incorrect or excessive use of tools for gaining insight or diagnosis can seriously skew the data used for health monitoring and capacity planning. It is not uncommon to diagnose that excessive monitoring lies at the root of performance complaints. Benchmarks tend to yield their best results with little or no monitoring activities competing for resources.

Record Keeping

Every item in a medical *patient folder* has a clear purpose:

- Basic patient data, including contact and billing information
- Drug allergies and current medications
- History of complaints, diagnoses, treatments, and ongoing conditions
- Results of routine physicals
- Lab reports
- Referrals and reports from other doctors

In contrast, there is a noteworthy lack of standard practices in building a similar folder for computer systems. Perhaps such a file should contain, at a minimum:

- Basic customer data, including contact and billing information
- Configuration information, not only for the system itself, but also for key applications
- History of complaints, diagnoses, treatments, and ongoing conditions
- Routinely monitored data
- Special test reports, including the conditions under which they were obtained
- Reports and findings of consultants and system administrators

A frequently recommended practice in systems management is to record any and all system changes in a central *change log*. This is a good way to capture the system history, but it can be difficult to determine the system's state at any given point in time without keeping periodic snapshots of its configuration state. One principle method of acquiring system configuration data from the Solaris OS is the Sun™ Explorer tool, which is freely downloadable from SunSolve Online℠ program site at `http://sunsolve.sun.com`.

Run with the defaults, the Sun Explorer software can be quite intrusive on a system running a production workload, but its usage can be tailored to be less intrusive to fit the occasion. Among the data it collects are:

- `/etc/system` and other `/etc` files
- Details of the storage stack, down to the disk firmware level, when possible
- `showrev -p` and `pkginfo -l` output, which can be fed into the `patchdiag` tool for patch analysis

  The `patchdiag` tool is available on the SunSolve Online program site.

The Sun Explorer software collects its outputs in a simple file hierarchy, which is easily navigated both by humans and automated system-analysis tools.

One of the biggest challenges in data collection is the observation and the logging of real business metrics and first-order indicators of business performance. The value of archival configuration and operational data is limited if it cannot be correlated with actual business throughput.

Traps and Pitfalls

Many patterns of error in performance work are so commonplace that they warrant a separate article on performance analysis traps and pitfalls. Some of the most common errors are discussed briefly here, with the moral that not everyone should aspire to be a doctor.

Statistics: Cache Hit Rate

Simple statistical findings often contradict instinct and intuition. Consider this simple question: "What is the practical difference between a 95 percent cache hit rate and a 96 percent cache hit rate?" I have asked this question in numerous meetings with customers and in front of multiple user-group audiences. Consistently, the correct answer never comes from the audiences. That is partly because it is a trick question. Intuitive answers like "one percent" and "not much" are wrong. The correct answer, at least in principle[6], is a whopping 20 percent!

Why? Mainly because of what statisticians call ***sample error***. What one should care about most in cache behavior is the ***miss rate***, not the ***hit rate***. For example, in the case of database I/O, each database cache miss results in what the database will view as a physical I/O operation[7]. If all of the I/O is the result of a 5 percent miss rate when the hit rate is 95 percent, then lowering the miss rate from 5 percent to 4 percent lowers the demand for I/O (at least for the reads) by 20 percent.

High cache hit rates might be the result of extensive unnecessary references involving the cache under consideration[8]. The best I/O, as they say, is one you never need to do.

Statistics: Averages and Percentages

> *"Did you hear the one about the statistician who drowned in a pool that was one inch deep, on average?"*

Averages, in general, have limited analytical utility. Effects such as queuing for resources tend to be masked in proportion to the time interval over which the observations are averaged. For example, if ten operations started at the same moment and they all completed within one second, one would report this as "ten per second". However, the same experiment would be reported as "one per second" if measured over a 10 second interval. Also, if the ten operations needed to queue for a resource, the latency of each operation would depend on its time spent waiting in the queue. The same operations would each see a lower latency if they did not need to wait through a queue.

Even "100 percent busy" is not necessarily a useful metric. One can be 100 percent busy doing something useful, or 100 percent busy doing something of low strategic priority. Doctors can be 100 percent busy on the golf course. You can be 100 percent busy being efficient, or 100 percent busy being inefficient. Just as setting priorities and working efficiently are key to maximizing one's personal utility, they are also key concepts in managing computer resource utilization.

Statistics: "Type III" Error

Apart from the common notions of sample error, mathematical error, or logical error, statisticians commonly speak of two types of errors: Type I and Type II. A Type I error involves someone erroneously rejecting the correct answer, and a Type II error involves someone erroneously accepting the incorrect answer. In a 1957 article titled

6. Reported hit rates might not cover all categories of application I/O; thus, they can be misleading.
7. This is true at least as far as the database is concerned. Reads might in fact be found in the filesystem cache rather than causing a real physical I/O.
8. Regarding Oracle, see Cary Millsap in the References section.

"Errors of the Third Kind in Statistical Consulting," Kimball introduced a third type of error (Type III) to describe the more common occurrence of producing the correct answer—to the wrong question!

As discussed previously, the primary metrics of system performance come in terms of business requirements. Solving any perceived "problem" that does not correspond with a business requirement usually results in wasted effort. In the abundance of statistics and raw data that flow from complex systems, there are many opportunities to stray from the path of useful analysis.

For example, some tools report a statistic called *wait I/O* (`%wio`), which is a peculiar attempt to characterize some of a system's *idle* CPU time as being attributable to delays from disk I/O. Apart from the obvious flaw of not including network I/O delays in the calculation, the engineering units of "average percent of a CPU" makes very little sense at all. The method used to calculate this statistic has varied between Solaris OS releases, but none of the methods is backed by any concrete body of science. There is active discussion in the Sun engineering community that contemplates removing the *wait I/O* statistic entirely from the Solaris OS. The utility of this statistic is extremely limited. Mainly though, analysts must know that `%wio` is idle time. Whenever `%wio` is reported, actual *idle* CPU time must be calculated as either (`%wio` + `%idle`) or (100 - `%usr` - `%sys`).

Another not-so-useful statistic is the *load average* reported by the `w`(1) and `uptime`(1) commands. This metric was conceived long ago as a simple "vital sign" to indicate whether or not available CPU was meeting all demands. It is calculated as a moving average of the sum of the run queue depth (number of threads waiting for the CPU) plus the number of currently running threads. Apart from the previously mentioned problems with averages in general (compounded by being *moving*), this statistic has the disturbing characteristic that different favorable system tuning measures can variously drive it either up or down. For example:

- Improving disk I/O should cause more threads to be compute-ready more often, thus increasing this metric and likely advancing business throughput.
- Increasing contention for some resource can cause this metric to rise, but result in a net reduction in business throughput.

This is not to say that *wait I/O* and *load average* are not useful indicators of changes in system load, but they are certainly not metrics that should be specifically targeted as tuning goals!

Public Health

Public health is not so much a matter of medicine as it is of statistics. Doctors tend to know "what's going around," and they combine this knowledge with clinical signs to make accurate diagnoses. A common error of otherwise accomplished analysts lies in attempting to diagnose issues from low-level system statistics before

thoroughly reviewing configuration parameters and researching "what's going around." Often, problems can be easily spotted as deviations from best practices or absence of a program patch. Identification of public health issues requires a view beyond what a typical end-user or consultant can directly attain.

Logic and Causality

It is easy to suspect that a bad outcome might be linked to events immediately preceding it, but to jump to such a conclusion would be to commit the common logical fallacy called "*Post hoc, ergo propter hoc*" (in Latin). This error is perhaps the most frequently committed of all. In English, this translates to "*After that, therefore because of that*," which scientists often express as "*Correlation does not imply causation.*"

Complaints such as "We upgraded to the Solaris 8 OS, and now we have a problem" almost invariably have nothing to do with Solaris 8 OS. System upgrades often involve upgrades to other components, such as third-party storage hardware and software, and sometimes, they involve application software upgrades. In addition, upgrades occasionally involve application-migration process errors.

Accurate and timely diagnoses sometimes require full disclosure of historical factors (the patient file) and are usually accelerated by keeping focused on hypotheses that exhibit plausible chains of causality.

Experiment Design

A common troubleshooting technique is to vary one parameter with all other things held equal, then to reverse changes which produce no gain. Imagine applying this strategy to maximizing flow through a hose with three kinks in it!

Experiment design is a major topic among scientists and statisticians. Good experiments test pertinent hypotheses that are formed from a reasonable understanding of how the system works and an awareness of factors that might not be controllable. This can be as much art as science. When done wrong, huge amounts of time can be wasted by solving the wrong problem and perhaps even exacerbating the real problem.

Where Does the Time Go?

Performance issues are all about time. Successful performance sleuths start from the perspective that "You can always tell where the time is going." Armed with a broad collection of tools, they set out to determine where the time is going. Then, they drill down on why the time is going wherever it is going. Knowing where the time can go is essential to accurately diagnosing where the time is actually going. Here, we break it down, starting with CPU usage.

At a high level of analysis, CPU cycles can be separated into various broad categories. These categories are not mutually exclusive.

- Application code

  The application code is the main logic of a program, which could be expressed in languages as diverse as C, C++, Java™, SQL, Perl, or ABAP.

- Interpreter code

  Interpreter code is the implementation of the engine interpreting the application code, such as a C compiler, a Java runtime environment, or a database engine.

- System calls

  The efficiency of system-call implementations are clearly in the domain of the operating system vendor, but unnecessary calls to the system might originate from a variety of places.

- Library calls

  Whether library calls involve system-supplied libraries or application libraries, the fact that they are contained in libraries offers a chance for adding instrumentation.

At a lower level of analysis, CPU cycles are often attributed as:

- User (`%usr`)

  User cycles represent the amount of CPU cycles used in the user mode. This includes the time spent for some system calls when their logic does not require switching to ***kernel mode.***

- System (`%sys`)

  System cycles represent the amount of CPU cycles used in the ***kernel mode.*** This includes time spent in the OS kernel due to system calls from processes, as well as due to kernel internal operations such as servicing hardware events.

- Interrupts

  Prior to the release of the Solaris 9 OS, time spent in interrupt handlers was not attributed as either system or user time. Whether or not interrupt processing is observable, it will certainly account for some percentage of actual CPU usage. The servicing of some interface cards, such as gigabit ethernet cards, can consume a large proportion of a CPU.

It is noteworthy that the precision of CPU usage attribution in the Solaris OS is not 100 percent accurate. The quality of time accounting in the Solaris OS is under continuous improvement and will vary between software releases.

Some low-level phenomena in a system can help explain CPU consumption in the previously described categories. Analysis of these events is largely in the domain of *gurus*, but tools are evolving that can digest this information into metrics that can be used by a broader range of analysts.

- Traps

  The Solaris 9 OS introduces the `trapstat`(1M) tool for reporting on time spent in low-level service routines including traps[9] and interrupts. Traps occur for low-level events such as delays in remapping memory accesses or handling certain numerical exceptions.

- Esoterica

  Tools like `busstat`(1M) and `cpustat`(1M) can be used to report on low-level counters embedded in the CPU or system architecture. Useful analysis with these tools requires very specific knowledge of low-level architectural factors.

Elapsed time, which is not attributable simply to CPU usage, might be attributed to the following categories:

- I/O delays

  All I/O is very slow compared to CPU speeds.

- Scheduling delays

  Scheduling delays involve the allocation of and contention for CPU resources. This factor is not nearly as well understood as I/O and memory factors, and it can be very difficult to model, forecast, and comprehend.

- Memory delays

  Whether at the CPU chip level or due to page faults at the virtual memory abstraction layer, delays in memory access ensue when a memory reference cannot be immediately satisfied from the nearest cache.

9. See *The SPARC Architecture Manual, Version 9* by Weaver and Germond in the References section and similar publications.

- Protocol delays

  Protocols such as TCP/IP may stall on protocol events such as message acknowledgements or retransmissions.

- Synchronization delays

  Applications use a variety of mechanisms to coordinate their activities. These mechanisms range from inefficient file system-based schemes (for example, based on `lockf`(3C)) to more generally efficient techniques, such as machine-optimized mutual-exclusion locking mechanisms (that is, `mutexes`[10]). Among the methods used to attain low latency in lock acquisition is the ***spin lock*** or ***busy wait***, which is basically a form of polling.

These delays are not necessarily mutually exclusive, and they overlap with the previous categories of time attribution. For example, memory delays might count against `%usr` or `%sys` and might relate to the traps and esoterica mentioned above. I/O latencies can include significant CPU and protocol latencies.

Having a complete overview of where the time *can* go can be most helpful in formulating diagnostic strategies.

Diagnostic Strategies

Rapid and accurate diagnoses can be critical to controlling costs. Given the high stakes common in commercial computing, the importance of diagnostic strategy deserves some special attention. Rapid problem resolution can prevent lost revenues arising from inadequate system performance. Accurate analyses can prevent the waste of time and money that can result from undisciplined approaches. For example, system upgrades undertaken without proper analysis, often called “throwing iron at the problem,” can be very disappointing when the extra iron fails to solve the actual performance bottleneck.

As in medicine, the shortest path to an accurate diagnosis lies in the timely consideration and exclusion of both the grave and the mundane, and in timely referrals to appropriate specialists. Following a systematic high-level diagnostic strategy can help speed this process. The following questions represent one possible process-oriented high-level strategy for performance diagnosis:

- Is the system correctly configured?
- Have appropriate best practices been applied?
- Has appropriate research into candidate bugs been performed, and have the appropriate patches and updates been applied?

10.See *Solaris Internals* by Mauro and McDougall in the References section.

- Is the performance complaint based on reasonable expectations and valid experiment design, backed by good science?
- If a resource is saturated, can more resources be added, or are there options to decrease demand?
- What resources are under contention?
- Where is the time going?
- Why is the time going wherever it is going?

Each of these questions can be augmented by asking if the services of a specialist are needed. A detailed diagnostic outline could fill volumes, but the best diagnosticians do not have, want, or need a highly detailed or rigid process. They merely combine their knowledge, experience, and skills to find problems and fix them. If they determine that the problem is outside their expertise, they make an appropriate referral.

Any strategy that succeeds will be celebrated at the time of victory, whether or not the strategy was philosophically defensible. There are innumerable cases in which trial-and-error strategies will work out well, especially when executed by highly experienced and knowledgeable practitioners. Well-informed hunches and lucky guesses that pan out are undeniably strategically efficient, but as strategies, they are not characteristically predictable.

Much of what is written on system tuning is commonly arranged by subsystem (for example, CPU, disk I/O, and messaging), as in the imminently practical and proven approach described in Chapter 21 of *Configuring and Tuning Databases on the SOLARIS Platform* by Allan Packer. Packer's strategy is exemplary of a classical functionally-oriented approach.

At a functional level, tuning and troubleshooting techniques share a great deal in terms of tools and knowledge base. While system tuning is characteristically a trial-and-error process, it is generally preferred that troubleshooting techniques converge on root causes with maximum determinism and minimal experimentation. The success of system tuning is measured in terms of empirical results. Drill-down troubleshooting might lead to discovery of a previously undiagnosed issue for which no empirical payoff might exist until the issue is repaired.

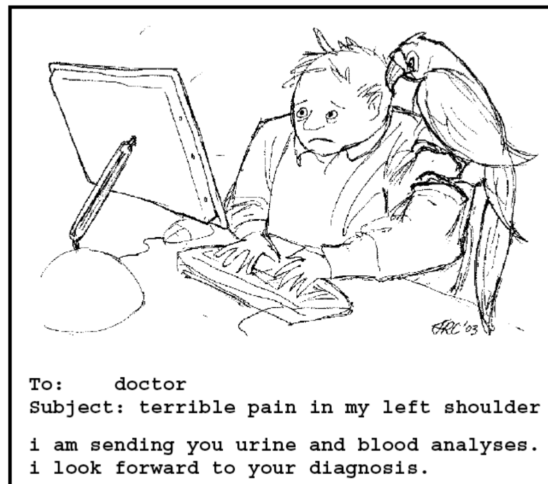
A simple three-step functionally-oriented strategy[11] that is equally applicable to tuning or troubleshooting is to ask:

1. Is the system working or waiting for something?
2. If it is working, is it working intelligently and efficiently?
3. If it is waiting, can the thing it is waiting for be made faster, or can you decrease dependence on that thing?

11. Thanks to Jim Viscusi of Oracle Corporation.

High-level diagnostic strategies are useful as road maps for diagnostic efforts and lay solid foundations for rational problem-solving.

Regardless of the strategy used, there will commonly be challenges to successful execution. For example, a challenge faced by doctors and performance analysts alike is patients who are convinced they have provided all the data needed to make their diagnosis. Diplomacy may be required to extract from the patient or customer whatever data is actually needed.



Root Causes

The root causes of bad performance are just as diverse as the spectrum of illnesses ranging from the common cold to incurable fatal diseases. As in medicine, knowledge of the relative likelihood of various diagnoses has some bearing on the process used for diagnosis.

The process of troubleshooting most frequently reveals root causes that are more closely analogous to the common cold than the rare fatal disease. The common cold is usually diagnosed with no lab tests whatsoever, but rather by the presence or absence of clinical signs, and consideration of the season and the locale. Just as doctors do not invest in culturing the virus that causes the common cold, much of the effort that gets invested in performance analysis could be avoided by beginning with basic symptom-matching diagnostic techniques.

The following categories of root causes should not require intense analysis to diagnose, and should therefore be considered at the earliest phases of diagnostic efforts. There is no simple policy for making these considerations before delving deep into low-level root cause analysis techniques, but much of the time and effort that gets consumed in performance troubleshooting could be saved by doing so.

- Best practice deviations

  Just as 80 percent of disease might be avoided by good habits of diet and exercise, a similar rate of performance complaints can be traced back to deviations from known best practices. Uninformed decisions regarding system feature selections or failure to make appropriate space-versus-speed tradeoffs can often be quickly diagnosed by inspection. It is vastly preferable to diagnose these by examination of configuration parameters than to discover them from low-level technical analysis or sifting through reams of data.

- Known bugs

  By some estimates, 80 percent of all reported bugs have been previously reported. Often, bug reports already on file indicate workarounds or availability of product patches. For known issues with no workaround or patch, the best path to getting them repaired involves building a business case by raising escalations against the appropriate bugs or RFEs.

- Errors in experiment design and data interpretation

  Claims of performance problems sometimes arise from psychological factors doctors routinely encounter, such as hypochondria or anxiety. Sometimes, ostensibly alarming data will turn out to be benign, and some symptoms will turn out to be of no consequence. Much effort can sometimes be saved be conducting a "sanity check" before investing too heavily in additional data collection and analysis. This involves making a preliminary assessment that the problem is real, and establishing realistic expectations regarding performance goals.

- Resource saturation

  This is a major category of initial diagnosis, but more often, it is merely an indicator of a need for tuning, rather than a root cause. A 100-percent-busy issue is often the result of avoidable inefficiencies, so one should not rush to judgment over the inadequacy of the resource itself.

The preceding categories are somewhat of a mixture of science and art, but they should be properly evaluated before investing in detailed low-level root cause analyses.

Whether or not the root causes of a complaint have been previously identified and characterized, they will fall into a finite set of categories. Knowledge of the number and nature of the categories can help guide diagnostic reasoning processes. Medical texts often arrange diseases into familiar categories, such as viral, bacterial, genetic, and psychological. Here is an attempt to enumerate the general categories of root causes for computer performance issues.

- Bad algorithms

  Intuition leads many people to think that bad or inappropriate algorithms are at the heart of performance issues, more often than they actually are. Still, this is a major category of diagnosis. Some algorithms have issues scaling with data volumes, while others have issues scaling to meet ever-increasing demands for scaling through parallelism. Diagnosis of the former might require a detailed analysis of execution profiles using programming tools, and the latter might be diagnosed from looking for contention symptoms. Of course, to say an algorithm is "bad" is rather broad-ranging, and the subsequent categories listed here enumerate some particular types of badness.

- Resource contention

  Given that resource-sharing issues can occur in various kernel subsystems, as well as in application software and layered, third-party software, you might need to look in many places to locate resource contention. At high-levels of resource utilization, the relative priority of resource consumers becomes an interesting topic. Failure to prioritize and failure to tune are common diagnostic findings. In this context, a resource is not simply a CPU, memory, or an I/O device. It also extends to mechanisms such as locks or latches used to coordinate access to shared resources. Queuing and polling for scarce resources can account for considerable clock time, and they can also account for a considerable percentage of overall CPU utilization.

- Serialization

  When it is possible for pieces of a computing workload to be processed *concurrently*, but concurrency does not occur, the work is said to proceed *serially*. Missed opportunities for concurrent processing might arise from laziness or ignorance on behalf of a programmer, constraints on program development schedules, or non-optimal configuration and tuning choices. In some cases, unintended serialization will be diagnosed as a bug.

- Latency effects

  When an operation is highly iterated and not decomposed into parallel operations, the time per iteration is of particular interest. While this notion is obvious in cases of computational loops, latency effects in the areas of memory references, disk I/O, and network I/O are often found at the root of many performance complaints.

- Hardware failures

  Failed or failing system components can lead to poor overall performance. Certain modes of failure might not be immediately apparent, though most failures will result in an error message being logged somewhere.

- Common inefficiencies

  Any book on programming optimization techniques will likely feature a long list of common programming inefficiencies. Among these are major categories like memory leaks, along with common inefficiencies such as repeatedly evaluating the length of a string, poorly chosen buffer sizes, or too much time spent managing memory. Doing unnecessary work counts as a major category, especially when it involves inefficiencies in the underlying system. For example, some versions of the Solaris OS have slow `getcwd`(3C) and `fsync`(3C) performance (see BugIDs 4707713 and 4841161 for `getcwd` and BugID 4336082 for `fsync`).

- Esoteric inefficiencies

  Sometimes, the root cause of performance issues lies at very low levels in the system, involving factors with which most analysts have no concrete grasp. For instance, you might observe a low-level instrumentation issue in the system architecture or the CPU itself. As systems evolve toward chip multiprocessing, you can expect the subject of low-level efficiency to become an increasingly hot topic.

What really defines a performance topic as *esoteric* is the talent, experience, and tools required to troubleshoot it. Many simple problems will appear difficult to the practitioner who lacks the right experience. Some difficult problems will require the services of authentic gurus to diagnose. The steady advance of performance analysis tools will inevitably reduce the mystique of these factors.

Selected Tools and Techniques

This section contains a survey of helpful tools for performance diagnosis, along with a description of features for each tool. Each tool has some value for gaining insight into how things work and where the time is going. Many of the tools are useful for routine monitoring and tuning activities. We start with two important yet often overlooked categories, then discuss tools specific to the Solaris OS.

Application-Specific Instrumentation

Most major application components (for example, database engines, transaction monitors, and web servers) feature facilities for performance monitoring and logging. These facilities are an obvious place to look for clues, and they often include metrics that can be tightly correlated with actual business goals. One should also consider these metrics in the overall scheme of health monitoring and capacity planning activities.

Log Files

It can be very embarrassing to find a log file with reams of errors and warnings indicating an obvious problem just after great effort has been expended discovering the same problem by indirect means. It is good practice to be fully familiar with the logging capabilities of various components, and fully aware of how they are configured.

On many occasions, the rate of diagnostic logging, the size of a log file, or contention for log file writing has been the root cause of a broader performance issue. There are often cases where turning up the diagnostic level only exacerbates an existing performance complaint.

ps(1)

In the Solaris OS, the `ps`(1) command includes features that are frequently overlooked by many users. In particular, the `-o` option allows selection of a specific set of column headings. For example, you can use the following command to gain insight into the thread-level priorities of all of the threads on a system:

```
$ ps -e -o pid,ppid,lwp,nlwp,class,pri,args
```

The Solaris OS also offers a Berkeley UNIX version of the `ps` command, invoked as `/usr/ucb/ps`. The man page is available by typing `man -s 1b ps`. One trick unique to the Berkeley version of `ps` is the ability to show the command arguments of each process by using multiple `-w` options, as in the following example:

```
$ /usr/ucb/ps -www
```

This command can sometimes help in shedding light on why a process is using excessive resources.

vmstat(1), mpstat(1), and netstat(1)

The *stat* commands comprise the largest category of performance monitoring tools, and they continue to evolve. For example, in the Solaris 8 OS, the addition of the `-p` option to the venerable `vmstat`(1) command greatly simplifies the observation of paging-related phenomena. Among the most useful top-level monitoring tools is `mpstat`(1). Some information reported by `netstat`(1) with the `-s` option is not readily available elsewhere. These tools are basic to spotting CPU, network, and memory utilization issues. They are the workhorses of system management and are covered well by most books pertaining to UNIX systems management.

prstat(1)

At first glance, `prstat`(1) appears to be a poor substitute for the `top`[12] command. However, `prstat` has considerable diagnostic capabilities, while `top` has next to none. Two of the most interesting capabilities of `prstat` are its ability to do thread-level monitoring with the `-L` option and its ability to display microstate accounting data with the `-m` option. These features establish `prstat` as an important tool for forming hypotheses regarding root causes of performance issues.

Microstate accounting provides detailed information (optionally, at the thread level) on the breakdown of wall-clock time. As reported by `prstat`, the microstates are:

- `USR` (percentage of time in non-trap user mode CPU mode)
- `SYS` (percentage of time in kernel mode CPU mode)
- `TRP` (percentage of time in user-mode trap handlers)
- `TFL` (percentage of time waiting for text-page faults)
- `DFL` (percentage of time waiting for data-page faults)

12.The `top` command is a shareware program that is not included in the Solaris OS. The Solaris OS includes `sdtprocess`, which is an X Windows-based alternative to the top command.

- `LCK` (percentage of time waiting on user-mode locks)
- `SLP` (percentage of time sleeping for other causes)
- `LAT` (percentage of time waiting for CPU)

In addition, microstate data includes some rate data of great interest for performance analysis:

- `VCX` (Voluntary ConteXt switches, rate at which a process or thread surrenders its CPU prior to consuming its time quantum)
- `ICX` (Involuntary ConteXt switches, rate at which a process or thread has its CPU stolen by expiration of its time quantum or due to preemption by a higher-priority thread)
- `SCL` (rate of system calls)
- `SIG` (rate of signal events)

Indeed, `prstat` is one of the most useful tools for gaining insight into where the time is going at both the process and thread levels.

`iostat`(1M)

The `iostat` command is one of the most commonly used, and perhaps least well-understood, of the ***stat*** commands. Much of the data reported by `iostat` is also available from `sar`(1M) or `kstat`(1M), but the format of `iostat` output is the most convenient and widely used for monitoring or investigating disk I/O.

My own preferred `iostat` usage is usually `iostat -xnzTd`, where:

- `x` (which returns extended device statistics)
- `n` (which returns more useful device names)
- `z` (which eliminates rows with all zeroes)
- `-Td` (which timestamps each report interval)

Digesting reams of `iostat` data can be a wearisome task, but tools have been developed that post-process data in this format. One extremely useful tool is an `awk`(1) script called `iobal` (see Smith in the References section). This handy tool summarizes `iostat` data in terms of bandwidth (megabytes per second) or I/O per second, and optionally, it lists the top ten most active targets. Recent versions also do controller aggregation (as `iostat` does with the `-C` option); thus, you can easily assess controller utilization from `iostat` data that was not collected with the `-C` option.

truss(1)

The primary capability of truss(1) is observing all of the system calls emanating from a process. Over time, truss(1) has been enhanced to provide a great deal of other functionality. In some cases, the functionality overlaps the capabilities of sotruss(1) and apptrace(1), but truss(1) has the ability to attach to an already running process.

Caution – truss(1) is intrusive on the monitored process. Depending on the options, the intrusiveness can range from slight to severe.

Running truss -c -p *pid* against a process for a brief interval, then interrupting it (for example, with Ctrl+C), gives a concise summary of the frequency of systems calls and their average execution times. This can be quite handy for finding system-call hot spots, or on some occasions, bringing system-call implementation efficiency into question.

The -c, -d, and -D options are among the handiest features of truss(1). The ability of truss(1) to calculate timestamps with the -d option and time deltas with the -D option can be quite useful for finding out where the time is going. However, be aware that the time deltas are easily misinterpreted. It is easy to think that the time delta between two reported events represents the time required by the former event, but in fact, the time delta includes any and all compute time between reported events. In addition, because truss(1) has a significant probe effect inherent in the timing data, taking the timing data too literally would not be appropriate.

lockstat(1M)

lockstat(1M) is a very versatile tool for gaining significant insights into where the hots spots are in the system with respect to locking activity. Hot locks are a basic indicator of contention in a system. As a “starter incantation,” one might try the following command:

```
$ lockstat -H -D64 sleep 10 > lockstat.out
```

This command tabulates the top 64 lock events in several categories over a period of 10 seconds. When using this command, you can safely ignore any error messages about “too many records.” The right-hand column in the output consists of cryptic programming symbols. Although a programmer can extract more information from this data than a non-programmer, the symbols are usually named so that they give good clues as to where the system is busy.

The following list gives some examples:

- ufs_ (the UFS file system)
- va_ (virtual address primitives)
- vx_ (something in the VERITAS software)
- lwp_ (light-weight process [that is, thread] management)
- cv_ (condition variables, synchronization primitives)

High lock dwell times or high lock iteration counts can have significant diagnostic power by indicating what parts of the system are most active.

The nanosecond values reported by lockstat(1M) can take some getting used to. A nanosecond is 10^{-9} seconds, so the first digit of a seven-digit nanosecond value is in milliseconds (10^{-3} seconds).

kstat(1M)

The kstat(1M) command reports on statistics that are maintained by the lightweight kstat facility within the Solaris OS kernel. The command interface is a bit awkward, and kstat would be much improved by the addition of error messages when incorrect arguments are used. Nevertheless, its ability to pull statistics by class or module is very versatile and efficient, and kstat) is now preferred over the deprecated netstat -k command.

Because the kstat facility keeps only counts, it is useful to employ tools on top of this interface to compute, log, and display rates. The most famously useful set of tools for this purpose is the SE Toolkit by Adrian Cockcroft and Richard Pettit, which is freely available from http://www.setoolkit.com.

trapstat(1M)

Introduced in the Solaris 9 8/03 OS release[13], the trapstat(1M) command tabulates and formats information about time spent handling certain low-level events. For example, trapstat -t reports on the percentage of time spent in handling memory management tasks.

High-percentage times spent in memory management can indicate memory stride problems or opportunities for code optimization. Here again, the tool is largely in the domain of gurus, but casual users can extract some useful information about where the time is going.

13.An unsupported version exists for the Solaris 8 OS, but it has limited functionality.

busstat(1M), cpustat(1M), and cputrack(1M)

These tools enable specialists to investigate issues previously described as being *esoteric*. Productive use of these tools requires very specific low-level knowledge of system components. Over time, other tools should evolve to automate some of the analyses that are possible based on data from these tools.

Of particular interest are measurements of efficiency such as "cycles per instruction" (CPI), which characterizes low-level overall CPU efficiency. A free tool to derive CPI measures, called har, is available online from Sun's Market Development Engineering web site at http://www.sunmde.com/perf_tools/har.

Sun™ ONE Studio Developer Tools

Sun's programming development suite includes tools for performance analysis. Of particular interest is the ability to gather performance data from noninstrumented binaries (collect) and analyze the collected data either by command line (er_print) or GUI (analyzer). Much of the information on the use of these tools is available on the Internet. Search on the keywords "er_print collect analyzer" to find links to official sources, as well as a wide variety of how-to material.

The Sun ONE Studio developer tools are the tools of choice for identifying code optimization targets and for measuring the impact of incremental code refinements. It can be a very useful exercise, even for non-programmers, to get some firsthand experience on the capabilities of these tools.

References

The following references were used in the development of this article:

- Kimball, A.W. "Errors of the Third Kind in Statistical Consulting." *Journal of the American Statistical Association,* 52, pp 133-142, 1957.
- Mauro, Jim and Richard McDougall. *Solaris Internals.* Prentice Hall, 2001, ISBN 0-13-022496-0.
- Millsap, Cary. "Why You Should Focus on LIOs Instead of PIOs." 2001, downloadable from http://www.hotsos.com (after free registration). Last seen under a link called "Are you still using cache hit ratios?"
- Packer, Allan Packer. *Configuring and Tuning Databases on the SOLARIS Platform.* Sun Microsystems Press, 2002, ISBN 0-13-083417-3.

- Smith, Robert. iobal, a handy awk(1) script for digesting iostat data, hosted on the downloads page of http://solarisinternals.com/si/downloads/
- Weaver, David L. and Tom Germond, Ed. *The SPARC Architecture Manual, Version 9.* Prentice Hall, 1994. ISBN 0-13-099227-5.

Third-Party URLs

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party Web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Acknowledgments

I have many to thank for their varied inputs to the SuperG paper, on which this article is based. Some have played multiple roles. Some *guru* colleagues whose thought processes and techniques I aspire to document and emulate include Pallab Bhattacharya, Phil Harman, and Paul Riethmuller. Some gurus who created useful tools that make performance analysis easier for mere mortals and *guru wannabes* include Adrian Cockcroft, Richard McDougall, Denis Sheahan, Rich Pettit (Adrian's conspirator on the SE Toolkit), Bryan Cantrill (from Solaris OS kernel engineering), and Bob Smith (a Sun colleague). Thanks to a gauntlet of reviewers, including Cecil Jacobs, David Miller, Don Desrosiers, Jim Mauro, Jim Viscusi (from Oracle), Dave Adams and Bob Kenny (from the CMG community), and Dan Barnett (from the Sun BluePrints team). Thanks to my wife Janet for her artwork and editorial assistance—and for always calling me to dinner.

About the Author

Bob Sneed works in the Enterprise Applications Group (EAG) of the Commercial Applications Engineering (CAE) team within Sun's Performance and Availability Engineering (PAE) organization. His group's charter is to continuously improve the competitive standing of key ISV applications in the Sun and Solaris OS environment. Bob stays obsessed with system performance topics, often involving Oracle and data-storage technologies. His recent focus has been on service delivery and knowledge management pertinent to performance issues. Prior to joining Sun in 1997, he worked as a Sun ISV, a Sun reseller, and a Sun customer. His early career experiences in real-time industrial controls and software portability engineering helped shape his unique perspectives on database and system performance issues.

Ordering Sun Documents

The SunDocs[SM] program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is: `http://docs.sun.com/`

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine web site at: `http://www.sun.com/blueprints/online.html`