Paper 289-28

SAS 9.1 on Solaris 9 Performance and Optimization Tips

Maureen Chew

Abstract: This paper is targeted to administrators of SAS[tm]/Solaris[tm] systems who want to learn about complementary performance improvements in both SAS 9.1 and Solaris 9. Additionally, tips, tricks, and spells to optimize and tune for performance are discussed. The topics are advanced, but you don't need to be a wizard if you think your SAS/Solaris configuration has been hit with the 'Petrificus Totalus' curse (that is, 'molass-ification'). Many topics are relevant to other UNIX/Java platforms.

SAS Version 9.1

With the release of Version 9.1, the SAS software platform unleashes a powerful and flexible new foundation for the next generation of product deliverables. Discussion of these new capabilities is outside the scope of this paper but we will address the running, monitoring and tuning of this platform on Solaris 9.

The advent of SAS Version 9, brought forth a release which implemented a SAS threaded kernel(TK) (not to be confused with the Solaris threaded kernel) which provided an infrastructure for SAS R&D developers to code multi-threaded PROCs. In addition to the V9 PROCs which were re-coded to take advantage of the TK infrastructure, there are several SAS servers such as the SAS Open Metadata Server or the SAS OLAP server which are threaded and run as background, daemon processes.

SAS Version 9.1 unleashes a powerful new software platform that potentially consists of a very different computing model. The following types of programs could be running concurrently:

- Traditional SAS applications – single process , one or more RSUBMIT/MP CONNECT processes
- Standalone Java programs (Ex: SAS Management Console,Enterprise Miner Client)
- SAS programs which invoke Java programs through the in process Java Virtual Machine- JVM) - Ex: SAS/GRAPH components used to render images)
- Java mid-tier programs (Ex: SAS Web Studio Reporting run as JSP/servlets in a Web container)
- SAS Services (Ex: SAS Open Metadata Server, OLAP Server, Infomap Server,Workspace Server, etc)

While these various SAS servers and services can be configured on different platforms, the Sun servers are well suited to handle multiple, multi-threaded applications. Consolidation of multiple SAS servers on a single platform can simply administration iff the HW configuration can support the load.

Thus, a sample full blown Version 9.1 implementation might reasonably have the following running at any given time:

- 30 traditional SAS users run "batch" applications. Any given SAS application may or may not invoke multi-threaded PROCs
- 50-100 users accessing the SAS Web Studio reporting functionality (mid-tier Java based layer sitting in a Web container such as Sun ONE Application Server 7 calling out to traditional/legacy SAS backend processes
- 10-30 users accessing the SAS OLAP server
- 5-10 users logged in over Reflection X running the Java based clients such as SAS Management Console or Enterprise Miner Client.
- 5-7 background SAS processes such as the SAS Open Metadata Server.

Solaris 9

SAS V9.1 is built on, and fully supported on the Solaris 8 Operating Environment (OE). But Solaris 9 is particularly well-suited and preferred if there are no other site specific 3rd party application dependencies.

The "9 Cool Things about Solaris 9" includes:

1.Sun ONE Application Foundation
    Integrated Sun ONE Directory and Application Server
2.Data Management
    Improved file system performance and management
3.Provisioning and Change Management
    Installation (Live Upgrade, Flash/JumpStart, Secure WAN boot)
4.Server Virtualization
    Solaris Containers, Dynamic Reconfiguration, Resource Management
5.Security
    Firewall Everywhere, PAM enhancements, Ipsec/IKE, Kerberos V Server
6.Enhanced Cluster Support / High Avaialbility
  Sun Cluster Software, StorEdge Traffic Manager, Network Multipathin(iPMP)
7.Configuration Management
    Solaris Patch Manager, BigAdmin Portal, RAS Knowledge Database
8.Performance
    Memory, Threading Improvements, Improved Directory Name Lookup Cache(DNLC)
9.Compatibility
    Solaris Compatibility Assurance Toolkit (SolCAT), Application Compatibility Guarantee

A couple of areas are worth mentioning as particularly relevant to the running of SAS applications.

❿Solaris 9, Update 2(12/02) supports Memory Placement Optimization which allows the Solaris Operating Environment to recognize memory locality effects and intelligently place memory pages and processes close to each other. This would be relevant when running SAS on the larger mid-range (ie: Sun Fire 6800) and high-end servers (ie: E12000/E15000). Additionally, other memory management improvements related to advanced page coloring are included.

❿A new and improved 1x1 (as opposed to MxN) threads library is shipped in Solaris 9.

❿Solaris 9 bundles in fine grained Resource Management capability which could be very useful for large, complex SAS installations that support many users and have varying quality of service requirements.

SAS V9.1 on Solaris 9

Basic performance monitoring commands in the context of SAS applications are discussed in the paper: *Pushing the Envelope: SAS System Considerations for Solaris/UNIX in Threaded, 64 bit Environments*

Solaris 9 uses a 1x1 threading model which translates to roughly a 1-1 correspondence between Lightweight Processes(LWPs) and threads.

Prstat(1), introduced in Solaris 8, is a powerful command line tool to give e you a snapshot of the top running processes or detailed information about a single process:

```
# prstat 5
 PID  USER    SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
 28239 sasdhd   41M   33M cpu3     0    0   7:54:04  14% sas/5
  3668  jcliu   46M   40M sleep   59    0   1:36:51 0.3% mozilla-bin/3
  2308  root    86M   75M sleep   59    0   0:14:23 0.1% java/15
   850  root  5072K 3416K sleep   59    0   0:12:52 0.1% automountd/3
    47  root  4704K 4440K cpu2    59    0   0:00:00 0.0% prstat/1
  5510  root   166M  152M sleep   59    0   1:05:09 0.0% java/220
  2714  root    87M   74M sleep   59    0   0:14:25 0.0% java/16
   773  root    28K 1256K sleep   59    0   0:00:00 0.0% keyserv/3
   861  root  4296K 2632K sleep   59    0   0:02:02 0.0% syslogd/15
   875  root  2264K 1304K sleep   59    0   0:00:00 0.0% cron/1
Total: 199 processes, 796 lwps, load averages: 1.20, 1.19, 1.20
```

Let's example the top process:

```
  PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/LWPID
 28239 sasdhd     41M   33M cpu3     0    0   7:54:04  14% sas/5
```

and take a further look at the process on an LWP basis:

```
# prstat -L -p 28239
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU
PROCESS/LWPID
 28239 sasdhd     54M   39M run      0    0   7:51:35  14% sas/3
 28239 sasdhd     54M   39M sleep   59    0   0:00:00 0.0% sas/5
 28239 sasdhd     54M   39M sleep   59    0   0:00:00 0.0% sas/4
 28239 sasdhd     54M   39M sleep   59    0   0:00:00 0.0% sas/2
 28239 sasdhd     54M   39M sleep   59    0   0:00:00 0.0% sas/1
```

Out of the 5 threads, only 1 is active and accumulating time. SAS spawns a number of housekeeping threads which are inconsequential in terms of total CPU cycles consumed. This system has 7 CPUs so the % CPU is an average across all CPUs. Since this LWP has the CPU pegged, the average is 100/7 or 14%.

Let's look at a SAS Open Metadata Server process:

```
 UID   PID  PPID  C    STIME TTY      TIME CMD
olap  3991  3990  0   Jan 24 ?        0:09 /901_unx/master/SAS/sas.s64no -
config /901_unx/master/SAS/sasv9.cfg.s64no -set
```

Using prstat(1), again, display on an LWP basis:

```
#  prstat -L -p 3991
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/LWPID
  3991 olap       98M   69M sleep   47    4   0:00:00 0.0% sas/9
  3991 olap       98M   69M sleep   47    4   0:00:00 0.0% sas/8
  3991 olap       98M   69M sleep   47    4   0:00:00 0.0% sas/7
  3991 olap       98M   69M sleep   47    4   0:00:01 0.0% sas/6
  3991 olap       98M   69M sleep   47    4   0:00:00 0.0% sas/5
  3991 olap       98M   69M sleep   47    4   0:00:00 0.0% sas/4
  3991 olap       98M   69M sleep   47    4   0:00:01 0.0% sas/3
  3991 olap       98M   69M sleep   47    4   0:00:04 0.0% sas/2
  3991 olap       98M   69M sleep   47    4   0:00:00 0.0% sas/1
```

From this snapshot we can observe that the server is sleeping and not accumulating time while in an idle state. We see that while 9 LWPs are spawned, only 3 (LWPs 6,3,2) have racked up any execution time. Thus, when monitoring run times for capacity planning purposes, it's important to get a handle on the number of ***significantly active*** threads or LWPs and not just total number spawned. You may see processes with 10's or 100's of LWPs. Don't get alarmed unless a large number of them are active.

Another useful command is **pargs(1)** which can print out all the arguments specified at command invocation.
Again, let's look at a SAS Open Metadata Server process:

```
  UID   PID  PPID  C    STIME TTY      TIME CMD
olap  3991  3990  0   Jan 24 ?        0:09 /901_unx/master/SAS/sas.s64no -
config /901_unx/master/SAS/sasv9.cfg.s64no -set
```

Because the output of **ps(1)** above truncates the full command line, we do not know the calling sequence. When there are multiple SAS services running, especially when started under the same user id, we could easily have difficulty in locating a specific instance of a service. If you needed to stop a process with **kill(1)**, it would be very unfortunate to inadvertently specify an incorrect process id (PID). **Pargs(1)** can be used to help determine the correct instance of a service.

PID 3991 above is an instance of the SAS Open Metadata Server that we wish to stop/restart after changing a configuration file. We use **pargs(1)** to dump the arguments of the command:

```
# pargs 3991
3991:   /901_unx/master/SAS/sas.s64no -config
/901_unx/master/SAS/sasv9.cfg.s64no -set
argv[0]: /901_unx/master/SAS/sas.s64no
argv[1]: -config
argv[2]: /901_unx/master/SAS/sasv9.cfg.s64no
argv[3]: -set
```

```
argv[4]: SASROOT
argv[5]: /901_unx/master/SAS
argv[6]: -altlog
argv[7]: metalog.txt
argv[8]: -nodms
argv[9]: -memsize
argv[10]: 510M
argv[11]: -sortsize
argv[12]: 510M
argv[13]: -nonews
argv[14]: -noovp
argv[15]: -noterminal
argv[16]: -objectserver
argv[17]: -objectserverparms
argv[18]: protocol=bridge port=7500 classfactory=2887E7D7-4780-11D4-879F-
00C04F38F0DB instantiate nosecurity
```

From the output, we can correlate this instance of the server to the one in question..

Pstack(1) is another useful command to determine a traceback of individual active LWPs. From a **truss(1M)** of a SAS process, a user was seeing a significant number of calls to **poll(2)** and was concerned that it consuming an inappropriate amount of cycles. This is how we proved that it was not an issue.

```
# find the PID of the SAS job (16573 in this case)
base-2.05$ ps
  PID TTY      TIME CMD
  9757 pts/7    0:01 bash
 16572 pts/7    0:00 runit
 16574 pts/7    0:00 ps
 16573 pts/7    0:03 sas
```

```
# dump the thread stack and find the "poll"er
bash-2.05$ pstack -F 16573
16573:  /d0/v91/sasexe/sas -fullstimer -WORK /d2/WORK -memsize 2G -sortsize
1G

bash-2.05$ pstack -F 16573
16573:  /d0/v9/sasexe/sas -fullstimer -WORK /d2/WORK -memsize 2G -sortsize
1G
----------------  lwp# 1 / thread# 1  --------------------
 ffffffff7e9187d8 lwp_park (0, 0, 0)
ffffffff7e915a34 cond_wait_queue (0, 0, ffffffff7ea1b8fc, 0, 0,
ffffffff7e200000) + d4
ffffffff7e9161e4 cond_wait (ffffffff7d605e00, ffffffff7d605de8,
ffffffff7df0bf00, ffffffff7df0c068, 0, cd8) + 10
ffffffff7e916220 pthread_cond_wait (ffffffff7d605e00, ffffffff7d605de8, 0, 1, 1, 2) +
8
ffffffff7df0c068 bktWait (ffffffff7d605de8, 0, 1, 1, 1, 1) + 108
ffffffff7df0b4e8 sktWait (ffffffff7d605cc0, 0, ffffffff7ffffae8,
ffffffff7ffffac8, 1, ffffffff7d605da0) + 168
000000010001d4e8 main (b, ffffffff7ffffbc8, 238, ffffffff7d700000, 0, 0) + a8
000000010001587c _start (0, 0, 0, 0, 0, 0) + 17c
-----------------  lwp# 2 / thread# 2  --------------------
ffffffff7e6a3550 poll     (ffffffff7c2f7b70, 0, 32)
ffffffff7e654fe0 _select (32, ffffffff7e7b83e8, ffffffff7e7b83e8, 0,
ffffffff7e7b83e8, ffffffff7cf0db70) + 298
ffffffff7e91140c select (1, 0, 0, 0, ffffffff7c2f7cf8, ffffffff7df0b280) + 6c
ffffffff7df0c8a4 bktHandleChildProcess (ffffffff7d700000, ffffffff7cf0be80,
ffffffff7df0c3bc, c350, ffffffff7e0398b0, ffffffff7cf0d710) + 1c4
ffffffff7df0ad58 sktMain (ffffffff7cf0d710, d400000, 803fc000, 2800000,
d400020, ffffffff7df0c6e0) + b8
ffffffff7df0bf3c bktMain (ffffffff7cf0d710, 0, 0, 0, 0, 4000) + 3c
ffffffff7e9186c8 _lwp_start (0, 0, 0, 0, 0)
-----------------  lwp# 3 / thread# 3  --------------------
.....
```

```
# use prstat(1) to watch LWP activity for this process
bash-2.05$ prstat -L -p 16573
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/LWPID
 16573 sasmau    146M  139M cpu1     0    0   0:00:21 7.3% sas/3
 16573 sasmau    146M  139M cpu8    50    0   0:00:18 7.1% sas/10
 16573 sasmau    146M  139M cpu0    31    0   0:00:04 2.5% sas/39
 16573 sasmau    146M  139M sleep   59    0   0:00:00 0.0% sas/5
 16573 sasmau    146M  139M sleep   59    0   0:00:00 0.0% sas/4
 16573 sasmau    146M  139M run     59    0   0:00:00 0.0% sas/2
 16573 sasmau    146M  139M sleep   59    0   0:00:00 0.0% sas/1
```

From the above, you can see that the "worker" threads (LWPs 3,10) are accumulating time but LWP 2 (the poller) is not, even though this particular snapshot was caught in the run state.

Though not new to Solaris 9, processor sets are a very simple mechanism to provide course grain CPU resource management. User maureen is contending with SAS wizards on a heavily burdened system. She needs to muster all her muggle know-how to prevent the wizards from denying her her fair share of processing. As an aside, she must get her analysis done over the weekend for a report due Monday and notices that some freeloading wizards have spawned processes for non time critical jobs. Her muggle bag of tricks happens to have the root password and she proceeds to allocate 3 processors for her own use.

```
# Query the number of processors
bash-2.05$ /usr/sbin/psrinfo
0       on-line   since 01/23/2003 15:00:11
1       on-line   since 01/23/2003 15:02:35
2       on-line   since 01/23/2003 15:02:35
3       on-line   since 01/23/2003 15:02:35
8       on-line   since 01/23/2003 15:02:35
9       on-line   since 02/01/2003 22:26:55
11      on-line   since 02/01/2003 22:26:55

# Carve off 3 processors
# psrset -c 8 9 11
created processor set 1
processor 8: was not assigned, now 1
processor 9: was not assigned, now 1
processor 11: was not assigned, now 1

# We can now confirm that all activity has drained from those 3 CPUs
# as their IDLE time increases and decreases for the remaining CPUs
# mpstat 5
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0    0   0  873   313  201  611   27  155   73    0  1218   15   3  58  23
  1   24   0  253    46    2  545   42  134   25    0   795   16  13  16  55
  2    7   0  204    62    1  643   57  144   33    0   926   30   2  17  51
  3   56   0  435    77    0  588   75  172   28    0   755   43   3   2  52
  8    0   0    1    23   22    0    0    0    0    0     0    0   0   0 100
  9    0   0    0     1    0    0    0    0    0    0     0    0   0   0 100
 11    0   0   49   316  315    1    0    0    0    0     0    0   0   0 100

# find the PID of the shell
bash-2.05$ ps
   PID TTY      TIME CMD
   232 pts/7    0:00 ps
  8371 pts/7    0:02 bash

# Bind the shell to that processor set
# psrset -b 1 8371
process id 8371: was not bound, now 1

# Start the SAS processes
bash-2.05$ /d0/v91/sas monthly_report.sas &

# We can start to see that processor 8 IDLE time has now gone to 0
# and processor 11 IDLE is dropping as well.  Maureen's process is
# now exclusively consuming cycles on this dedicated processor set.
bash-2.05$ mpstat 5
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0   20   0  332   310  206  358   24  148   70    0   327   21   0   0  79
  1   87   0 1059    39    5  543   32  149   34    0   679    7   1   0  92
  2   85   0 1241    59    2  489   55  134   26    0   614   36   2   0  62
  3   66   0  399    53    0  454   51  131   24    0   513   36  13   0  51
  8    0   0    1    46   22   39   23    0    1    0    63  100   0   0   0
  9    0   0   54     1    0  122    0    0    0    0    83    0   0   0 100
 11    0   0 24425    6    3   34    2    0    2    0   427    3   7   0  90
```

Solaris 9 bundles in Resource Management features which can be utilized to provide very fine grained and very flexible resource allocation and policy. The concept of projects and tasks are used to label workloads and separate them from one another. The project provides a network-wide administrative identifier for related work. The task collects a group of processes into a manageable entity that represents a workload component.

Java

SAS V9.1 standardizes on version 1.4.1 of the Java Runtime Environment (JRE).

As described earlier, there are 3 ways that the JRE can be invoked in V9.1.

❿ standalone
❿ from SAS
❿ through a web container such as Sun ONE Application Server 7

Accepting the default JVM options is probably acceptable for most cases but there could well be instances where tuning the JVM is desirable. While the majority of Java programs are short lived, it will be well worth some time characterizing the longer running ones to understand how different options might affect the overall runtime performance.

JRE 1.4.1 supports 2 environments; client and server. For graphics based applications or short running applications, the client(default) mode is usually best. However, long running or background service applications might perform better with the -server option. The -server option will tell the JVM to spend more time compiling and optimizing long running methods.

Typically, the parameters that you'll most likely need to change are the settings of the initial and maximum heap allocation. The -X setting applies to the Sun HotSpot JVM:

```
$ /usr/java1.4.1/bin/java -X:
....
   -Xms<size>       set initial Java heap size
   -Xmx<size>       set maximum Java heap size
.....
```

The default initial Java heap (-Xms) is 2M (MB) while the default maximum (-Xmx) heap is 64M.

Its not particularly straightforward to determine a one size fits all default especially when you have to consider the combined system memory utilization.

Below we discuss:

❿how to pass JRE options onto the in-core JVM or web container
❿collect statistics on garbage collections
❿show tradeoffs in performance and increased memory utilization

JRE options for the SAS in-core JVM can be set or changed:

❿at the SAS platform level in **$SASROOT/sasv9.cfg**

```
   jreoptions (-Djava.ext.dirs=!SASROOT/misc/applets -Xusealtsigs  )
```

❿override on command invocation

```
   $ sas -jreoptions "-Xms 128m -Xmx 128m" report.sas
```

For standalone Java applications, the same options can be applied at runtime when invoking the JRE.

When running a SAS Java application deployed as a .war file, the Java options are modified in the web container configuration or startup file For Sun ONE Application Server 7, this would be specifed in the server.xml file located in

```
$APP_SERVER_ROOT/var/opt/SUNWappserver7 domains/domain1/server1/config
```

where **domain1** and **server1** are named domain and server instances of the application server.

For Apache Tomcat 4.06, the JRE options would be specified in the **catalina.sh** startup file located in the **$APACHE_ROOT/bin**

Increasing the initial heap allocation can reduce the cost of more frequent garbage collection costs with the tradeoff of increased memory utilization. In our test below, we realized a gain of ~4 secods at the host of an extra 60 MB of initial memory allocation.

```
# Run a SAS application which calls Java components to render images and
# dump the garbage collection stats.. All in all, we'll see ~30 garbage collection events

bash-2.05$  time /d0/v91/sas -fullstimer -autoexec ../autoexec.sas \
-jreoptions " -verbose:gc " map.sas
[GC 2048K->762K(3520K), 0.0281766 secs]
[Full GC 17692K->14973K(25912K), 0.4007535 secs]
[GC 26877K->17742K(37184K), 0.0241216 secs]
[Full GC 26784K->20254K(38720K), 0.4270222 secs]
...................... ....
[GC 37454K->24701K(50208K), 0.0330959 secs]
[GC 40694K->27429K(50208K), 0.0333282 secs]
[GC 43427K->30153K(50208K), 0.0373865 secs]
[Full GC 46137K->27862K(50208K), 0.5242682 secs]
```

Our SAS log -fullstimer stats show:

```
real   0m32.001s  <=== takes longest, but uses least memory
user   0m42.350s  <==  takes most CPU cycles
sys    0m2.530s
```

```
# Increase the initial heap allocation to 64M, we see ~11 GC events, time drops ~7 sec
bash-2.05$ time /d0/v91/sas -fullstimer -autoexec ../autoexec.sas \
-jreoptions " -verbose:gc  -Xms64m -Xmx256m " map.sas
[Full GC 9072K->2959K(64896K), 0.1718656 secs]
[GC 23567K->7314K(64960K), 0.1069522 secs]
[GC 29122K->12186K(64960K), 0.0930330 secs]
.... (about 11 GC) ......

SAS log shows:
real     0m25.884s
user     0m26.500s
sys      0m2.220s
```

```
# If we further increase the initial heap allocation to 256m, we see only 2 GC events,  no improvement
# in performance and increased memory consumption
bash-2.05$ time /d0/v91/sas -fullstimer -autoexec ../autoexec.sas \
-jreoptions " -verbose:gc  -Xms256m -Xmx256m " map.sas
[Full GC 9018K->2959K(259584K), 0.1620886 secs]
[GC 85199K->16099K(259584K), 0.2766101 secs]
[GC 98335K->24868K(259584K), 0.1624228 secs]
---- (2 GC's, no improvement in time)

SAS log shows:
real   0m26.999s
user   0m36.190s
sys    0m2.780s
```

If problematic garbage collection is suspect, other JRE options to consider might be:
"-verbose:gc -Xloggc:<file> -XX:+PrintGCTimeStamps -XX:+PrintGCDetails"

Additionally, JDK1.4.1 contains 2 new garbage collections (parallel collector, concurrent mark-sweep collector). See the HotSpot reference below for more information on using these collectors.

The Java option, -Xprof, can also give hints as to the breakdown in time spent in various methods categorized by compiled and interpreter sections. This can also help decide whether -server option should be used.

Summary

SAS Version 9.1 brings a potentially very different computing model where multple multi-threaded, mixed environment (C & Java) applications are running simultaneously. We've examined how to monitor processes and examine resource consumption down to the thread/LWP level. Solaris 9 makes and excellent platform for handling a complex workload. Fine grained resource management capabilities are bundled in the base OS as well as an application server. Addtionally, we discussed modifying the SAS in-core JVM options and benefits and tradeoffs.

All tests were run on a Sun Enterpriese Midframe Server, Sun Fire 3800.

References

Solaris 9 Operating Environment
http://wwws.sun.com/software/solaris/index.html#features

Solaris 9 Operating Environment Data Sheet
http://wwws.sun.com/software/solaris/ds/ds-sol9oe/index.html

Solaris 9 12/02 System Administrator Collection ->
System Adminstration Guide: Resource Management and Network Services
http://docs.sun.com/db/doc/816-7125?q=Resource+Manager

Sun BluePrints[tm] OnLine - Resource Managements in the Solaris[tm] 9
Operating Environment - Stuart J. Lawson
http://www.sun.com/solutions/blueprints/browsesubject.html#resource

Sun BluePrints[tm] OnLine – Performance Oriented System Administration – Bob Larson
http://www.sun.com/solutions/blueprints/1202/817-1054.pdf

Solaris[tm] 9 Resource Manager Technical FAQ
http://wwws.sun.com/software/solaris/faqs/resource_manager.html

Performance Documentation for the Java HotSpot VM
http://java.sun.com/docs/hotspot/index.html

Pushing the Envelope: SAS System Considerations for Solaris/UNIX in Threaded, 64 bit Environments
http://www.sas.com/partners/directory/sun/64bit.pdf

Peace between SAS Users & Solaris/Unix System Administrators
http://www.sas.com/partners/directory/sun/performance/index.html

Turbo Charging SAS Applications in Solaris Environments
Managing Highly Performance Applications in Large Multi-User Environments
http://www.sas.com/partners/directory/sun/mgmt/index.html

Turbo-charging the Java HotSpot Virtual Machine, V1.4.x to
Improve the Performance and Scalability of Application Servers
http://developer.java.sun.com/developer/technicalArticles/Programming/turbo

About the Author:

Maureen Chew, Sr. Member of Technical Staff, has been with Sun Microsystems for over 14 years. She is a resident of Chapel Hill, NC and can be reached at maureen.chew@sun.com