SAS Version 9.1 on Solaris 9
Performance, Monitoring 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.

Contents

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 simplify administration if and only if 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 running "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 which would prevent an OS upgrade.
9 Solaris 9 features 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 Availability
   Sun Cluster Software, StorEdge Traffic Manager, Network Multipathing(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 particularly relevant to the running of SAS applications are:

- Solaris 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.

- A new and improved 1x1 (as opposed to MxN) threads library is shipped in Solaris 9. While SAS already takes advantage of this new library in Solaris 8, there are additional optimizations in the Solaris 9 version.

- Solaris 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

Useful Utilities – prstat, pargs, pfiles, pstack, truss

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.

In this section, we'll look at some tools and utilities that should be in every SAS user's bag of tricks where performance or performance monitoring is of concern.

Prstat(1), introduced in Solaris 8, is a powerful command line tool to give 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
```

where the top process shows:

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

This process has accumulated almost 8 hours of CPU time. The percentage in the CPU column is an average across all processors in the system. Additionally, 33Mb of active memory is used by this process.

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)**, -L will report statistics for each LWP:

```
#  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, but 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; Solaris is particularly efficient in handling idle LWPs.

pargs(1) is a utility new to Solaris 9 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 complete 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.

Another useful utility is **pfiles(1)** which can be used to find out what files a given process has open. If you've ever tried to unmount a filesystem only to have it fail because some process has it open, these commands can be used to find the offending process(es):

- Determine the major/minor number of the file system device in question. For example, we wish to **umount(1M)** the filesystem asscociated with /dev/vx/dsk/dg1/vol0.

```
# ls -l /dev/vx/dsk/dg1/vol03
brw-------   1 root     root     218,48002 Feb 26 09:53 /dev/vx/dsk/dg1/vol03
```

We determined that the major/minor device number for this filesystem is 218,48002. We can now run **prfiles(1**) for every process entry in /proc to look for this filesystem.

```
# cd /proc
# ls
0      1119   1188   1305   1406   15677  18638  3      857    927    969
1      1123   1190   1331   1408   15681  18648  4288   878    9289   973
10275  1147   1199   1332   1409   16608  18674  736    88     931    974
10277  1148   12     1375   1410   16610  18676  8169   898    942    978
1041   1149   1216   1382   14229  16856  18683  8171   899    943
1046   1152   1234   1383   14232  18543  18703  825    902    948
1110   11719  1235   1396   14387  18545  18706  8444   91     960
1112   11720  1236   1403   14397  18553  2      854    913    964
# for i in *
> do
> echo $i
> pfiles $i | grep 48002
> done
.........
1406
1408
14229
   9: S_IFREG mode:0644 dev:218,48002 ino:38401 uid:4000 gid:10
size:0
14232
.........
974
978
```

From the above output, we see that process 14229 has this device open. We are then able to kill off that process and the **umount(1M)** of the filesystem succeeds. Note: often a filesystem cannot be unmounted because the **automounter(1M)** has it exported. In this case, you will have to **unshare(1M)** the filesystem.

Truss(1M), used to trace either system calls or library calls is extremely useful and powerful tool. It can be used to narrow down application failures as you can trace system calls to determine what files were opened, error condidtions, arguments passed to system calls, location of configuration,

sequencing order for configuration file searches, etc. Additionally, the calls can be prefaced with time stamps or time delta stamps.

Pstack(1) is a command to determine a traceback of individual active LWPs. Here is an actual case of how this command was used to solve a user conern.. From a **truss(1M**) of a SAS process, a user was seeing a significant number of calls to **poll(2)** fill their screen and was concerned that it consuming an inappropriate amount of CPU resources.

Find the PID of the SAS job in question (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/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
0000000010001d4e8 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, 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.

Familiarize yourself with these commands and you'll be on your way to removing any anti-performance spells that have been cast on your applications or system.

Resource Management

This section will discuss 2 methods of CPU resource management; Solaris processor sets and Solaris 9 Resource Manager.
Solaris processor sets are not new to Solaris 9; they provide course grain CPU resource management. The Solaris 9 resource manager uses processor sets as the underlying foundation but provides a much more flexible and fine grained approach.

Solaris Processor Sets

Imagine the scenario where 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 the necessary CPU cycles to complete the year end reporting. The reports are due Monday so she must work over the weekend. She 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.

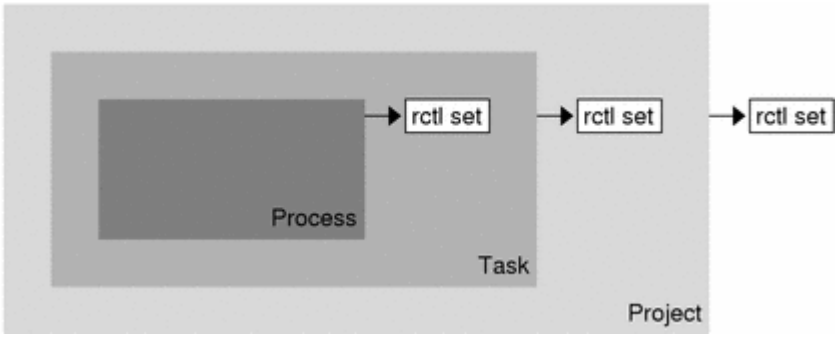
```
Query the number of processors
bash-2.05$ /usr/sbin/psrinfo
0       on-line   since 01/23/2003 15:00:11
1       on-line   since 01/23/2003 15:02:35
2       on-line   since 01/23/2003 15:02:35
3       on-line   since 01/23/2003 15:02:35
8       on-line   since 01/23/2003 15:02:35
9       on-line   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 and will have no problem completing the reports on time.

```
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 Resource Management

Solaris 9 bundles in Resource Management features which can be utilized to provide very fine grained and very flexible resource allocation. 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. The resource manager uses the concept of processor sets above as the underlying foundation.



What is not shown in the above image is that an additional superset layer is availabe, that of resource pools. These pools can be comprised of different projects each of which is running different tasks where each task can consist of 1 or more processes. In the next section, we'll go through a cookbook example of configuring and using this in the context of an enterprise SAS deployment.

Sample Scenario

The system, ctcsun8, is an 8 way, Sun Fire 3800. We would like to have 4 processors dedicated to SAS processing while the remainder would be used to service other system tasks or applications. There are 3 groups of users which have different priorities based on their budgetary contribution and level of SAS wizardry. The 3 groups or projects are:

- **dminers** – data mining analysts
- **olap** – OLAP users
- **queryrpt** – simple query and reporting usage

Since the **dminers** department contributed an amount equal to the combination of the **olap** and **queryrpt** department, they are entitled to 50% of the allocated resources while users in the **olap** and **queryrpt** project are entitled to half of the remaining resources or 25% each. The steps below show you that no wizardry is necessary to utilize the Solaris 9 Resource Manager.

Solaris 9 Resource Manager cookbook setup

Create resource pools, processing sets, define projects, determine allocation policies

- Create a config file, arbitrarily named */pool/poolcfg.ctcsun8* in our example. This specifies that the resource pool, **Sas_Pool**, will be associated with the processor set , **Sas_Pset,** that initially consists of 3 processors but can grow to 4 processors.

```
# cat /pool/poolcfg.ctcsun8
create system ctcsun8
create pset Default_Pset (uint pset.min = 3)
modify pset Default_Pset ( string pset.comment="DEFAULT PSET";
boolean pset.default = true)
create pool Default_Pool (string pool.scheduler="TS"; boolean
pool.default = true)
associate pool Default_Pool (pset Default_Pset)

create pset Sas_Pset (uint pset.min = 3; uint pset.max = 4)
modify pset Sas_Pset (string pset.comment="SAS USAGE")

create pool Sas_Pool (string pool.scheduler="FSS")
associate pool Sas_Pool (pset Sas_Pset)
```

Note: you always need to create a default resource pool in order to service system and other application needs.

- Convert the human readable config file into the required XML format. The command below writes the results to the default XML configuration file, */etc/pooladm.conf.*

```
# poolcfg -f /pool/poolcfg
```

- Verify the creation of the XML config file. Note: the config is **not** active yet.

```
# poolcfg -c info
system ctcsun8
        string system.comment
        int system.version 1
        boolean system.bind-default true
```

```
        pool Default_Pool
                string pool.comment
                boolean pool.default true
                boolean pool.active true
                int pool.importance 1
                string pool.scheduler TS
                pset Default_Pset

        pool Sas_Pool
                string pool.comment
                boolean pool.default false
                boolean pool.active true
                int pool.importance 1
                string pool.scheduler FSS
                pset Sas_Pset

        pset Default_Pset
                string pset.comment DEFAULT PSET
                int pset.sys_id -2
                string pset.units population
                boolean pset.default true
                uint pset.max 4294967295
                uint pset.min 3
                boolean pset.escapable false
                uint pset.load 0
                uint pset.size 0

        pset Sas_Pset
                string pset.comment SAS USAGE
                int pset.sys_id -2
                string pset.units population
                boolean pset.default false
                uint pset.max 4
                uint pset.min 3
                boolean pset.escapable false
                uint pset.load 0
                uint pset.size 0
```

- ***Activate*** the new configuration by using **pooladm(1M)**. The default XML configuration file, */etc/pooladm.conf*, is assumed if an input file is not specified. Note the usage of **psrset(1M)** both before and after **pooladm(1M)** to show that the **Sas_Pset** processor set was created and it consisted of 3 processors as expected.

```
# /usr/sbin/psrset -i
# pooladm -c
# /usr/sbin/psrset -i
user processor set 1: processors 0 1 2
```

- Verify the ***activated*** configuration

```
# pooladm

system ctcsun8
        string system.comment
        int system.version 1
        boolean system.bind-default true

        pool Default_Pool
                string pool.comment
                boolean pool.default true
                boolean pool.active true
                int pool.importance 1
                string pool.scheduler TS
                pset Default_Pset

        pool Sas_Pool
                string pool.comment
                boolean pool.default false
                boolean pool.active true
                int pool.importance 1
                string pool.scheduler FSS
                pset Sas_Pset

        pset Default_Pset
                string pset.comment DEFAULT PSET
                int pset.sys_id -1
                string pset.units population
                boolean pset.default true
                uint pset.max 4294967295
                uint pset.min 3
                boolean pset.escapable true
                uint pset.load 2125
                uint pset.size 4

                cpu
                        string cpu.comment
                        int cpu.sys_id 3

                cpu
```

```
                        string cpu.comment
                        int cpu.sys_id 8

                cpu
                        string cpu.comment
                        int cpu.sys_id 9

                cpu
                        string cpu.comment
                        int cpu.sys_id 11

        pset Sas_Pset
                string pset.comment SAS USAGE
                int pset.sys_id 1
                string pset.units population
                boolean pset.default false
                uint pset.max 4
                uint pset.min 3
                boolean pset.escapable false
                uint pset.load 0
                uint pset.size 3

                cpu
                        string cpu.comment
                        int cpu.sys_id 0

                cpu
                        string cpu.comment
                        int cpu.sys_id 1

                cpu
                        string cpu.comment
                        int cpu.sys_id 2
```

- Create the **project(4)** definitions and fair share scheduling allocations in */etc/project.* Lines are broken for readability but need to be continuous to be syntactically correct.

```
dminers:1001:Data Mining Analysts:::project.pool=Sas_Pool;
                        project.cpu-shares=(privileged,50,deny)
olap:1003:OLAP Users:::project.pool=Sas_Pool;
                        project.cpu-shares=(privileged,25,deny)
queryrpt:1004:Query and Reporting:::project.pool=Sas_Pool;
                        project.cpu-shares=(privileged,25,deny)
```

The number of shares allocated to the Fair Share Scheduler (FSS) represents a relative number. It's not the number that counts but the number of shares relative to the total allocation.
In this example, 50% of the CPU resources (3 CPUs in our case) is allocated to users of the **dminers** progject while 25% is allocated to users in both the **olap** and **queryrpt** projects.

- Set the default project for users ***dm*** and ***maureen*** in */etc/user_attr*.

```
# grep project /etc/user_attr
dm::::project=dminers
maureen::::project=olap
```

- Log in as user ***dm*** and verify configuration settings

```
Show identity
bash-2.05$ id -p
uid=5008(dm) gid=2001(others) projid=1001(dminers)

Print default project
bash-2.05$ projects -d
dminers

Determine process id(PID) of shell (24547)
bash-2.05$ ps
   PID TTY      TIME CMD
 24557 pts/3    0:00 ps
 24547 pts/3    0:00 bash

Show active processor sets(pset)
bash-2.05$ /usr/sbin/psrset -i
user processor set 1: processors 0 1 2

Verify that shell is bound to this pset
bash-2.05$ /usr/sbin/psrset -q 24547
process id 24547: 1
```

```
Show resource pool and processor set binding. $$ is a shortcut
reference to shell PID
bash-2.05$ /usr/sbin/poolbind -q $$
24547   Sas_Pool
bash-2.05$ /usr/sbin/poolbind -Q $$
24547   pset    Sas_Pset

Show CPU allocation to be 50 shares for dminers project
bash-2.05$ prctl -n project.cpu-shares -i project  dminers
24585:  prctl -n project.cpu-shares -i project dminers
project.cpu-shares                      [ no-basic no-local-action
]
                        50 privileged deny
                     65535 system     deny           [ max ]
```

Demonstrating Fair Share Scheduling& CPU Contention

Let's look at a more interesting scenario. Given a 2 processor set, let's examine the effects of running 4 ***identical*** jobs in parallel. We will then repeat the exact same test but this time add a 5th identical job but will run at a higher priority.

We drop the SAS resource pool down to 2 processors in order to make the demonstrations of Fair Share Scheduling and CPU contention more straightforward.. Additionally, user **maureen** can switch to to the **dminers** project which will be the mechanism by which the 5th job will run at a higher priority. To enable **maureen** to switch to the **dminers** project, modify **/etc/project** with the change:

```
dminers:1001:Data Mining Analysts:maureen::project.pool=Sas_Pool;
          project.cpu-shares=(privileged,50,deny)
```

where **maureen** is now in the list of allowed users in this group.

Recall that user **maureen** is in the **olap** project which gets 25% of the CPU resources for the SAS resource pool. It was users in the **dminers** project that were allowed 50% of the CPU resources. After modifying our pool config file and re-activating the configuration with **poolconfig(1M) & pooladm(1M)** as above**,** we see that processors 0 & 1 are completely idle while the other processors have no idle capacity. Note: the system has a CPU out on loan so we are only seeing 7 CPUs instead of the normal 8.

```
# mpstat 5
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  2  809   1 2784   223  143  228   89   63   44    1  2129   63  37   0   0
  3 1628   0 3055   163   38  378  139   82   44    1  3255   85  15   0   0
  8 1078   3 1737   143   24  227  117   37   23    1  2591   90  10   0   0
  9 1239   1 6076    90    7  160   84   27   22    2  2882   87  13   0   0
 11 1542   2 3238   125    0  243  124   38   32    1  2877   90  10   0   0
  0    0   0  278   307  203    0    0    0    0    0     0    0   0   0 100
  1    0   0    0     2    1    0    0    0    0    0     0    0   0   0 100
```

```
Verify configuration for user maureen is as expected (default
project is olap). Check identity and project setting
bash-2.05$ id -p
uid=4239(maureen) gid=10(staff) projid=1003(olap)

Verify processor set configuration
bash-2.05$ /usr/sbin/psrset -i
user processor set 1: processors 0 1

Verify pool and pset.  $$ is a shortcut reference to the active
shell process
bash-2.05$ /usr/sbin/poolbind -q $$
28708   Sas_Pool
bash-2.05$ /usr/sbin/poolbind -Q $$
28708   pset    Sas_Pset
```

User **maureen** has a SAS a CPU intensive SAS application which does a single proc LOESS and normally shows a -fullstimer listing of:

```
      real time           58.30 seconds
      user cpu time       55.16 seconds
      system cpu time     0.73 seconds
```

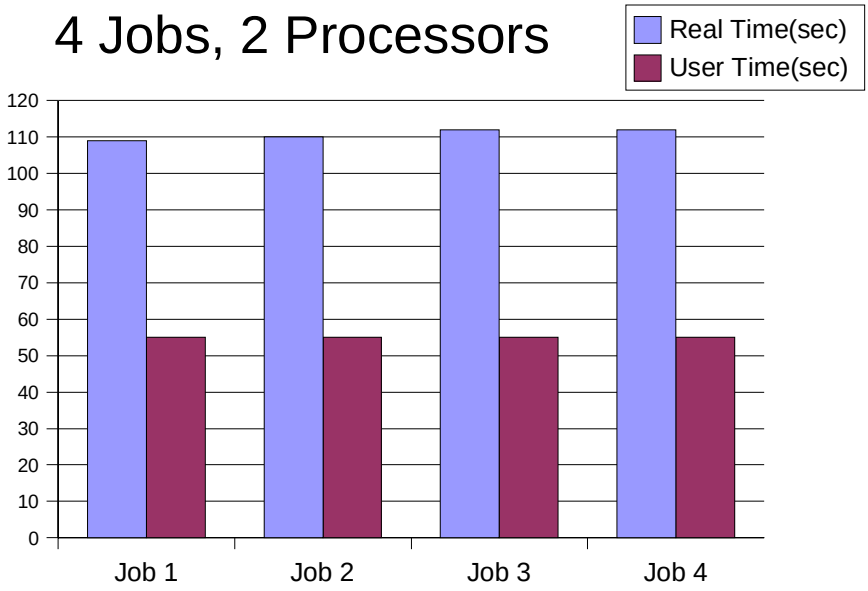
Here we see that wall clock time (real time) is very close to the user time – 58 sec vs. 55 sec. What happens when we start 4 of these processes in parallel? Our expectation is that user time will stay about the same while the real time should double because twice as much work is being requested than

there are processors available(4 jobs, 2 processors). After starting all 4 processors, we can use **prstat(1M)** to view jobs by project ID (default project for user **maureen** is **olap**):

```
bash-2.05$ prstat -j olap
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
  2807 maureen   150M   16M run      1    0   0:00:12 5.1% sas/5
  2809 maureen   150M   16M run      1    0   0:00:11 4.9% sas/5
  2813 maureen   150M   16M run      1    0   0:00:11 4.7% sas/5
  2811 maureen   150M   16M cpu0     2    0   0:00:10 4.6% sas/5
  3099 maureen  5704K 4768K cpu1    59    0   0:00:00 0.0% prstat/1
 28708 maureen  2520K 1856K sleep   59    0   0:00:00 0.0% bash/1
```

Note, that the jobs are executing on processors 0 & 1 as expected.

	Job 1	***Job 2***	***Job 3***	***Job 4***
Real Time	1:49.67	1:51.93	1:52.53	1:50.40
User Time	55.24 seconds	55.36 seconds	55.46 seconds	55.27 seconds



The results are as we expected in that the real time (wall clock) time is about 2X the user times.

Now to make things even more interesting. Using the same 2 processors in the **Sas_Pset** and running the same 4 jobs as before, let's add a 5th job (also identical to the other 4 so that we can understand the effects of the priority scheduling). However, before user **maureen** submits the 5th job, she will switch projects from her default of **olap** to **dminers**. Recall, that users in the **dminers** project will get **50%** of the processor set or 1 whole CPU in this case. So, we expect the **dminers** project job to finish in about the same time as the single test results at the start of this exercise where the wall clock or real time was 58 seconds. Additionallly, we expect that the other 4 jobs would take twice as long as because those 4 jobs are now only running on 1 CPU instead of the 2 before.

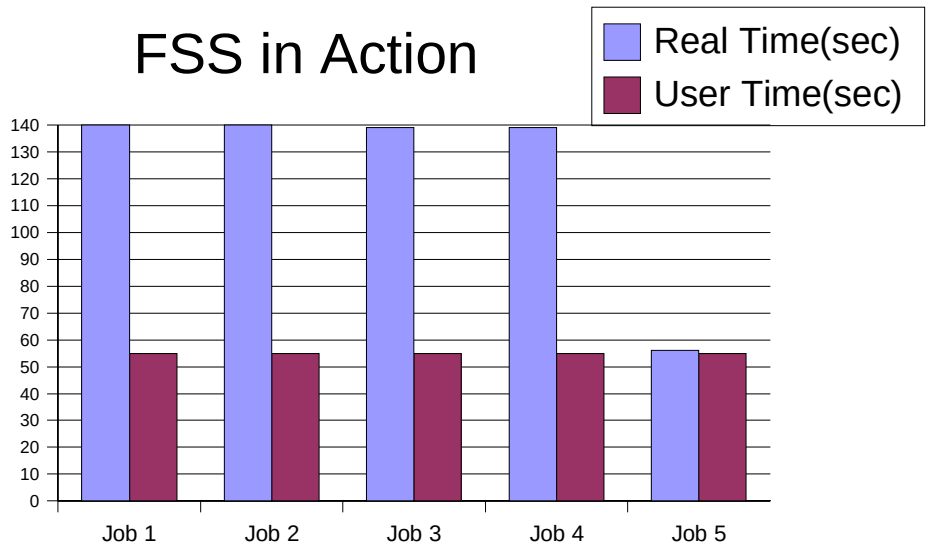
The execution mode for this test was a shell script which started the 4 jobs in the background. At that point, the command **newtask(1)** was used to switch from the default **olap** project to the **dminers** project. Then the 5th job was started.

```
After starting the 4 jobs,  switch to the dminers project with
newtask(1).  All tasks from this shell will then be associated with
the dminers project.
bash-2.05$ newtask -p dminers
```

```
bash-2.05$ id -p
uid=4239(maureen) gid=10(staff) projid=1001(dminers)

After all 5 jobs have started, prstat(1M) can be used to monitor
activity on a project or processor set basis:
Use -j <project> to watch all processes in a project. There are 4
SAS processes running in the olap project
# prstat -j olap
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU
PROCESS/NLWP
 25072 maureen   150M   16M run      1    0   0:00:22 4.6% sas/5
 25084 maureen   150M   16M run      2    0   0:00:23 4.5% sas/5
 25070 maureen   150M   16M cpu1     6    0   0:00:22 4.4% sas/5
 25077 maureen   150M   16M run      1    0   0:00:22 4.3% sas/5
 28708 maureen  2536K 1896K sleep   59    0   0:00:00 0.0% bash/1


There is 1 SAS process running in dminers project
# prstat -j dminers
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU
PROCESS/NLWP
 26222 maureen   150M   16M cpu0    45    0   0:00:26  10% sas/5
 25639 maureen  2528K 1816K sleep   59    0   0:00:00 0.0% bash/1
 22545 dm       2544K 1832K sleep   59    0   0:00:00 0.0% bash/

Use -C to for processor set (Note: 5 jobs running)
# prstat -C 1
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU
PROCESS/NLWP
 26222 maureen   150M   16M cpu0    51    0   0:00:09 5.4% sas/5
 25072 maureen   150M   16M run      1    0   0:00:20 5.2% sas/5
 25084 maureen   150M   16M cpu1     1    0   0:00:20 5.0% sas/5
 25070 maureen   150M   16M run      1    0   0:00:19 4.8% sas/5
 25077 maureen   150M   16M run      1    0   0:00:20 4.8% sas/5
 25639 maureen  2528K 1816K sleep   59    0   0:00:00 0.0% bash/1
 28708 maureen  2536K 1896K sleep   59    0   0:00:00 0.0% bash/1
```

Results for 5 jobs in parallel, where job 5 running in the dminer project

	Job 1	***Job 2***	***Job 3***	***Job 4***	***Job 5*****
Real Time	2:20.58	2:20.06	2:19.07	2:19.87	56.18 sec
User Time	55.22 sec	55.19 sec	55.21 sec	55.39 sec	55.38 sec



So, we indeed saw that the 5th job did finish in the same time as our single job test. We might have expected that jobs 1-4 would have taken 4X as long or 4 minutes since there were 4 jobs competing for a single CPU resource. However, as job 5 completed after 1 minute, the processor was available for use for the other 4 jobs and thus brought the time down. Thus, within a processor set, no CPU resources are wasted or kept idle if there is other work to do.

The Solaris 9 Resource Manager provides the ability for flexible and dynamic soft partitioning within a single instance of Solaris. The examples above have shown to create a simple instance of this partitioning as well as demonstrating the use of utilizing priority in the fair share scheduler.

Filesystem and Filesystem Cache Considerations

Filesystem configuration can make a huge difference in the performance of I/O intensive applications. Many system configurations are typically under-configured in terms of supplying enough I/O channels to support the required I/O bandwidth. The results of a simple I/O study demonstrate this concept. We examine an I/O workload where we had to complete 36 months of work where a *month* of work was a data step to read a unique 2G data set and re-format it to an output data set.. On a 24 way Sun Fire 6800 we tried 2 tests:

- 6 parallel queues of 6 jobs each
- 12 parallel queues of 3 jobs each

Initial results to complete all 36 months of work:

- 6 queues: 30 minutes
- 12 queues: 48 minutes

These tests were done on a single filesystem where the input and output data sets were on the same filesystem. Nonetheless, the results were counter to our expectations. We would expect 12 queues in parallel to run faster. After separating the input data sets and the output data sets to separate file systems and spreading the WRITE output workload to 6 different file systems(which were built on 6 different controllers), the 12 queue results improved dramatically:

- 12 queues: 9 minutes

When we took the 6 devices (driven by 6 controllers) and striped them together into 1 filesystem, the results for 12 queues increased slightly to 11 minutes. However it is probably worth the slight performance tradeoff as the complexity of distributing the I/O workloads and partitions is reduced fairly significantly.

The use of direct I/O can sometimes help I/O performance but we have found that in most cases, you have to work very hard to demonstrate benefit. And when it doesn't benefit, it usually hurts a great deal as sequential write I/O throughput tends to degrade by at least half and sequential read throughput to an even greater extent. Typically, the benefit is seen where there is a high I/O load *and* many CPU bound processes. In this case, while the I/O throughput is reduced, there are additional CPU cycles to service the CPU bound processes. It does take CPU cycles to manage the file system cache. In general, the Solaris file system cache is very efficient which is why it is difficult to find a generalized case where bypassing the cache is beneficial.

Prior to Solaris 9, the only tools available to look at the various kernel memory allocations was Richard McDougall's Memtool package (available at ftp://playground.sun.com). The Solaris 9 **mdb(1M)** utility now provides the same functionality as the Memtool prtmem command.

```
# mdb -k
Loading modules: [ unix krtld genunix ip isp sgsbbc usba s1394 wrsm
ufs_log logindmux ptm cpc random nca sppp ipc wrsmd nfs lofs ]
> ::memstat
Page Summary                Pages                MB  %Tot
------------     ----------------  ----------------  ----
Kernel                     274181              2142    4%
Anon                       313943              2452    4%
Exec and libs               10122                79    0%
Page cache                 984416              7690   14%
Free (cachelist)           857596              6699   12%
Free (freelist)           4623968             36124   65%

Total                     7064226             55189
> <CNTRL-D to exit>

And for those familiar with the Memtools prtmem command:
# prtmem
```

```
Total memory:           55189 Megabytes
Kernel Memory:           2141 Megabytes
Application:             2458 Megabytes
Executable & libs:         78 Megabytes
File Cache:              7691 Megabytes
Free, file cache:        6701 Megabytes
Free, free:             36118 Megabytes
```

In this example, we see that the system has 55 GB of RAM where ~7.6GB is dedicated to the file system page cache. We can see that in these snapshots that there is about 36GB freely available.

Java Usage

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.

- Standalone Java program
- SAS in-core JVM
- through a web container such as the 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 in this context are short lived, it could well be worth 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 complete system wide memory utilization requirements.

Below we discuss:

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

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

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

  `jreoptions (-Djava.ext.dirs=!SASROOT/misc/applets -Xusealtsigs )`
- overridden 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 the Sun ONE Application Server 7 web container, this would be specified in the server.xml file located in:
`$APP_SERVER_ROOT/var/opt/SUNWappserver7 domains/domain1/server1/config`
where **domain1** and **server1** are the 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 **$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 seconds at the cost of an extra 60 MB of initial memory allocation.

```
Dump the garbage collection status for a SAS application which calls
Java components to render images.. 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 suspected , other JRE logging options to consider might be:
"-verbose:gc -Xloggc:<file> -XX:+PrintGCTimeStamps -XX:+PrintGCDetails"

Additionally, JDK1.4.1 contains 2 new garbage collectors(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 multiple 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. The ability to partition and prioritize SAS workloads within a single system is particularly relevant in large enterprise SAS deployments. Addtionally, we discussed modifying options to the SAS in-core JVM and its benefits and tradeoffs. These tools and tips should be a good start in turning SAS user muggles into Solaris adminstration wizards!

All tests were run on a Sun Fire 3800 with Sun StorEdge[tm] A5200s and Sun StorEdge T3 Arrays unless otherwise specified.

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 Administration Guide: Resource Management and Network Services
http://docs.sun.com/db/doc/816-7125?q=Resource+Manager

Sun BluePrints[tm] OnLine - Resource Management 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