Solaris™ Operating System and ORACLE Relational Database Management System Performance Tuning

Ramesh Radhakrishnan, Sun Professional Services

Sun BluePrints™ OnLine—October 2003

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 817-3835-10
Revision A
Edition: October 2003

Please
Recycle

Solaris™ Operating System and ORACLE Relational Database Management System Performance Tuning

When a user experiences a performance problem, any one of the tiers used by the application could be causing the problem. For example, in a Web-based application, the problem could be in any one of the following tiers:

- End User Desktop (Client Tier)
- Web Server (Presentation Tier)
- Application Server (Business Tier)
- SQL Net/JDBC (Integration Tier)
- ORACLE database (Resource Tier)

The problem could also be in any of the networks that connect these tiers together.

This article focuses on the performance problems at the Resource Tier (database server). The assumption is that the database server is a Sun server running an ORACLE Relational Database Management System (RDBMS). The article requires a general knowledge of Solaris™ Operating System (Solaris OS) and Oracle RDBMS system administration and is written for beginner- and intermediate-level system administrators responsible for managing Sun systems, Sun's customer engineers, and database administrators responsible for tuning Oracle databases.

The article covers the following topics:

Solaris OS Performance Tuning

This section covers the following topics:

High-Level Diagnosis

You should first try to determine whether or not there is a really a performance problem. A few things to consider are:

1. If a system is overloaded (an unusually large number of users or transactions in an On-Line Transaction Processing (OLTP) environment or a larger data set than normal for batch processes), it needs more resources. These conditions do not necessarily mean that there is a performance bottleneck.

2. Always collect and keep baseline numbers for user-perceived performance such as:

- Time taken for key batch jobs to complete

- Response times for commonly used queries

- Overall system utilization levels

This data will help you verify if there is really a performance problem and, if so, how bad the problem is.

3. If the users’ response time has increased, but the number of users and number of transactions have not changed, there may be a bottleneck in the system or in the application being used.

4. Someone seeing higher numbers in the output of a UNIX command does not necessarily mean that there is a performance problem. Also, someone saying that the system is “sluggish” also does not necessarily mean that there is a performance problem.

Before you start the tuning process, make sure you get a baseline of the current performance of the database server. Documenting some hard numbers such as current cache hit rate, current scan rates, and so on will help you understand how much the tuning process improves performance.

The first step is gathering statistical data about the system behavior. Data gathering can be done either with bundled utilities (`mpstat`, `iostat`, and so on) or the SE Toolkit.

Download and install the SE Toolkit in `/opt/RICHPse` if it is not already installed. The SE Toolkit is located at:

`http://www.setoolkit.com/`.

After you download and install the SE Toolkit, run the GUI-based executable as follows:

```
# /opt/RICHPse/bin/se   /opt/RICHPse/examples/zoom.se
```

Caution – Do not run SE in a production environment. It is ok to run it in a development or test environment. If you have to run SE in a production environment, run it as a non-root user.

This command brings up a very nice GUI interface to SE and immediately helps identify resource shortages in the areas of memory, I/O, network, CPU utilization, or other kernel resource contention such as “spin on mutexes” or other lock mechanisms. Now you have a general idea of where in the system there are resource shortages, if any. The SE GUI also allows you to drill down a little further and gives you more diagnostic information. FIGURE 1 is a screenshot of the SE GUI. When you select any item, you can find more details. For example, if Disk is red, selecting Disk shows you the specific disks that are very busy.

Identifying a memory shortfall without the SE Toolkit is done by monitoring the scan rate (sr) column of `vmstat output`. If the page scanner never runs (sr is zero), memory is fine. If the page scanner is running, memory is running tight (this rule applies only to Solaris 8 and Solaris 9 OSs).

Diagnosing and Tuning Application and Memory Related Performance Problems

This section explains how to drill down further and investigate memory usage on the system as it relates to applications’ performance. This is what you should do if the SE Toolkit points to RAM shortages.

▼ To Diagnose Memory-Related Problems

1. **Try to identify the top two or three processes that are using a large portion of the system resources such as CPU and memory. This identification can be done using the** `toptool` **as follows:**

```
# /opt/RICHPse/bin/se        /opt/RICHPse/examples/toptool.se
```

FIGURE 1 SE Toolkit GUI

FIGURE 2 shows the `toptool` output.

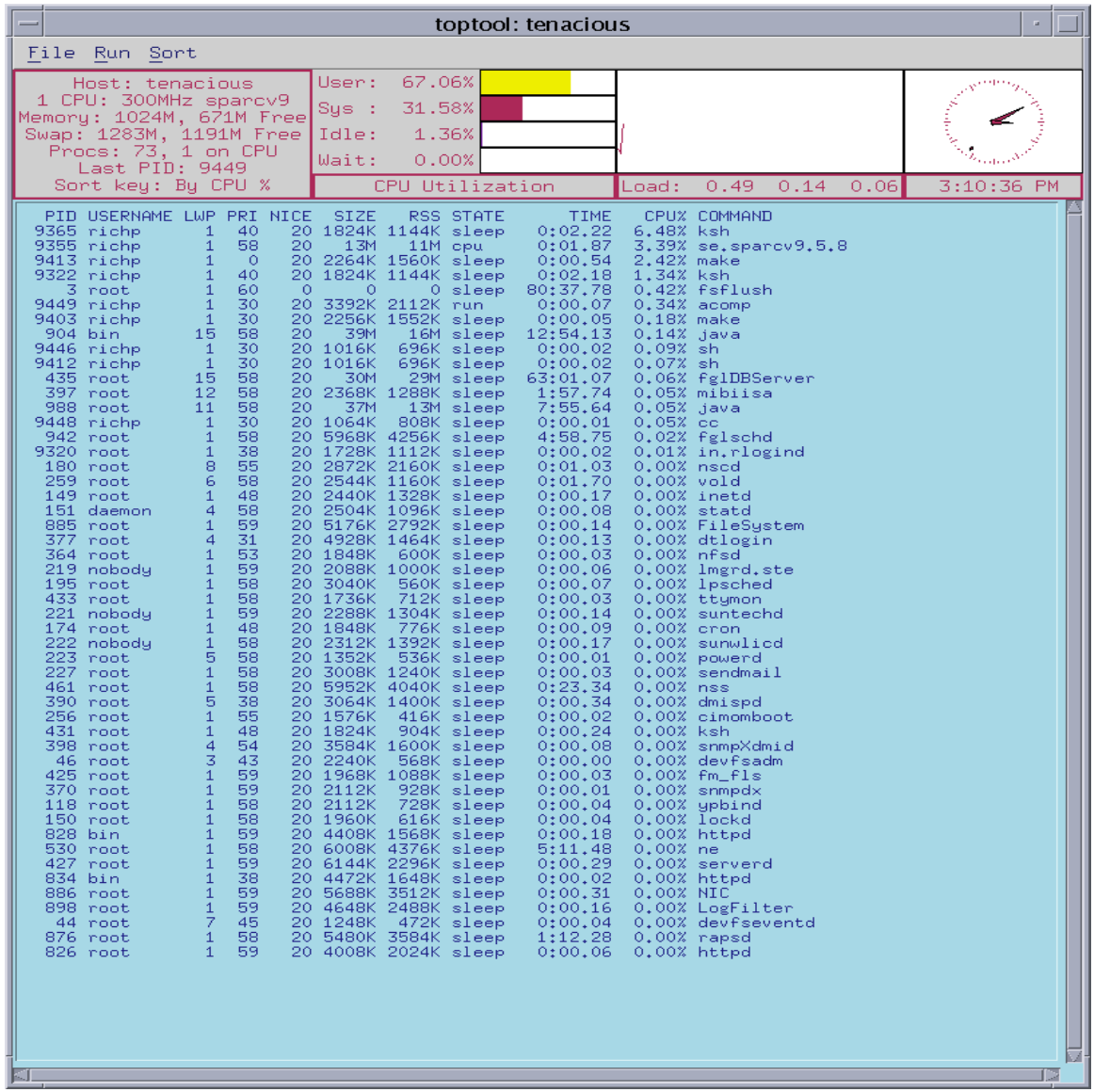


FIGURE 2 `Toptool` Output

At the top of this output you can see some pertinent information related to memory. You can see the total physical memory on the system and the amount of available memory. The list of processes is sorted by CPU usage. Observe the top few processes. The SIZE column tells you the amount of memory that this process uses, and the RSS column tells you the amount of resident memory (amount of physical memory actually in use). The total process address space includes physical memory allocated, Virtual Memory (VM) pages on disk and pages that have been allocated by the application (for example, by `malloc`()), but have never been touched.

The CPU % column tells you the percentage of CPU that this process utilizes. Note that the `toptool` output is sorted by CPU usage and not memory usage. Use this information as an initial diagnosis of which application processes are major consumers of the system resources.

A new top-like tool called `prstat` is bundled into Solaris 8 and Solaris 9 OS. This tool is much better than `toptool` and is more useful for diagnosis as discussed in the following paragraphs. CODE EXAMPLE 1 shows the output of `prstat`.

CODE EXAMPLE 1 `prstat` Output

```
 PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
102602 root     5240K 4328K cpu0    60  -20  29:02:17 9.2% nfsd/44
100438 root     3304K 2448K sleep   59    0   1:23:22 0.7% in.routed/1
102682 root     9384K 9088K sleep   59    0   1:14:07 0.6% mibiisa/7
100420 root     2496K 1368K sleep   58    0   0:28:51 0.3% in.mpathd/1
102650 root       47M   46M sleep   59    0   5:24:37 0.3% iCald.pl6+RPATH/2
100502 root     2912K 2320K sleep   54    0   0:49:59 0.2% rpc.rstatd/1
472925 jackaa   5824K 4984K sleep   59    0   0:00:02 0.2% imapd-daemon/1
468294 michely  3552K 2792K sleep   59    0   0:00:05 0.1% imapd-daemon/1
100542 root     8880K 6656K sleep   59    0   6:46:42 0.1% automountd/3
100520 root     2464K 1880K sleep   59    0   0:16:10 0.1% lockd/11
482764 rmc      1848K 1376K sleep   40    0   0:00:00 0.1% ksh/1
100497 root     4888K 3512K sleep   59    0   0:09:27 0.1% in.named/1
100468 root     4912K 3408K sleep   58    0   0:14:14 0.1% rpcbind/1
442441 chrish   6544K 5728K sleep   55    0   0:00:26 0.1% imapd-daemon/1
326502 joemil     20M   19M sleep   59    0   0:03:01 0.1% imapd-daemon/1
359054 ellisonl 9640K 8808K sleep   59    0   0:03:44 0.0% imapd-daemon/1
221207 cpullela   13M   12M sleep   59    0   0:01:04 0.0% imapd-daemon/1
482780 rmc      4728K 4352K cpu8    30    0   0:00:00 0.0% prstat/1
101278 root     4472K 2112K sleep   59    0   0:06:35 0.0% sendmail-8.12.8/1
436716 root     4480K 3096K sleep   59    0   0:00:54 0.0% in.ftpd/1
481974 macaccia 7032K 6216K sleep   59    0   0:00:36 0.0% imapd-daemon/1
467156 root     7312K 5976K sleep   59    0   0:00:05 0.0% nwadmin/1
482761 ravindra 3296K 2456K sleep   59    0   0:00:00 0.0% imapd-daemon/1
482664 root     4488K 3144K sleep   59    0   0:00:00 0.0% in.ftpd/1
473114 morteza  3800K 2992K sleep   49    0   0:00:03 0.0% imapd-daemon/1
477846 saraseth 5696K 4864K sleep   59    0   0:00:05 0.0% imapd-daemon/1
100573 root     7720K 6456K sleep   59    0   0:06:02 0.0% nscd/43
```

CODE EXAMPLE 1 prstat Output *(Continued)*

```
 PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
482557 paterson 3448K 2616K sleep   59    0   0:00:00 0.0% imapd-daemon/1
227647 jparcel  6272K 5504K sleep   59    0   0:00:36 0.0% imapd-daemon/1
199783 janani     13M   12M sleep   59    0   0:00:49 0.0% imapd-daemon/1
482628 root     5144K 2888K sleep   59    0   0:00:00 0.0% sendmail-8.12.8/1
464130 lucysa     18M   17M sleep   59    0   0:01:23 0.0% imapd-daemon/1
225182 rascal     17M   16M sleep   59    0   0:02:10 0.0% imapd-daemon/1
482284 root     5136K 4168K sleep   29   10   0:00:03 0.0% nsrmmdbd/1
459115 bkerr      18M   17M sleep   59    0   0:00:57 0.0% imapd-daemon/1
482631 root     2912K 1632K sleep   59    0   0:00:00 0.0% mail.local/1
```

By default, `prstat` lists the processes running on a system, sorted by CPU utilization. Note the numbers in the TIME column to see if a process has been running longer than it should. In this output, each process is not broken down into light weight processes (LWPs). Only the number of LWPs per process appears following a "/" after the process name in the last column. LWPs and their corresponding kernel threads allow multiple streams of execution within a single VM environment. The advantage of using LWPs is that they do not require VM context switches.

Note – An LWP is a virtual execution environment for each kernel thread within a process. It allows each kernel thread within a process to make system calls independent of other kernel threads within the same process.

The `prststat -L` output (CODE EXAMPLE 2) shows each LWP for a multi-threaded process. This will allow you to find out for sure if an application in question is multithreaded and is taking advantage of multiple CPUs on the system.

CODE EXAMPLE 2 `prstat -L` Output

```
 PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/LWPID
481965 clamb      14M   11M cpu8     0    0   0:00:29 2.4% imapd-daemon/1
100542 root     8880K 6656K sleep   59    0   3:09:25 2.3% automountd/1
483323 asd      7720K 6376K cpu9    60    0   0:00:02 0.9% imapd-daemon/1
102650 root       47M   46M sleep   59    0   2:07:15 0.5% iCald.pl6+RPATH/2
100438 root     3304K 2448K sleep   59    0   1:23:26 0.3% in.routed/1
458482 ktheisen   15M   14M sleep   59    0   0:01:14 0.2% imapd-daemon/1
102602 root     5240K 4328K sleep   60  -20   0:01:14 0.2% nfsd/657
102602 root     5240K 4328K sleep   60  -20   0:01:19 0.2% nfsd/650
102602 root     5240K 4328K sleep   60  -20   0:03:20 0.2% nfsd/616
102602 root     5240K 4328K sleep   60  -20   0:00:54 0.2% nfsd/662
479728 bubbva   3904K 3096K sleep   59    0   0:00:05 0.2% imapd-daemon/1
102602 root     5240K 4328K sleep   60  -20   0:01:20 0.2% nfsd/656
102602 root     5240K 4328K sleep   60  -20   0:03:22 0.2% nfsd/610
102602 root     5240K 4328K sleep   60  -20   0:06:55 0.2% nfsd/582
```

CODE EXAMPLE 2 `prstat -L` Output *(Continued)*

```
 PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/LWPID
102602 root     5240K 4328K sleep   60  -20   0:06:27 0.2% nfsd/593
102602 root     5240K 4328K sleep   60  -20   0:03:09 0.2% nfsd/619
102602 root     5240K 4328K sleep   60  -20   0:01:20 0.2% nfsd/654
102602 root     5240K 4328K sleep   60  -20   0:01:19 0.2% nfsd/652
102602 root     5240K 4328K sleep   60  -20   0:01:19 0.2% nfsd/651
102602 root     5240K 4328K sleep   60  -20   0:00:54 0.2% nfsd/664
102602 root     5240K 4328K sleep   60  -20   0:01:20 0.2% nfsd/655
102602 root     5240K 4328K sleep   60  -20   0:03:21 0.2% nfsd/609
102602 root     5240K 4328K sleep   60  -20   0:07:04 0.2% nfsd/559
100502 root     2912K 2320K sleep   59    0   0:50:02 0.2% rpc.rstatd/1
102602 root     5240K 4328K sleep   60  -20   0:00:54 0.2% nfsd/661
102602 root     5240K 4328K sleep   60  -20   0:06:28 0.2% nfsd/598
102602 root     5240K 4328K sleep   60  -20   0:00:54 0.2% nfsd/659
102602 root     5240K 4328K cpu16   60  -20   0:06:02 0.2% nfsd/602
102602 root     5240K 4328K sleep   60  -20   0:01:15 0.2% nfsd/647
102602 root     5240K 4328K sleep   60  -20   0:05:37 0.2% nfsd/604
102602 root     5240K 4328K sleep   60  -20   0:01:29 0.2% nfsd/645
102602 root     5240K 4328K sleep   60  -20   0:01:30 0.2% nfsd/640
102602 root     5240K 4328K sleep   60  -20   0:07:53 0.2% nfsd/422
102602 root     5240K 4328K sleep   60  -20   0:06:33 0.2% nfsd/601
102602 root     5240K 4328K sleep   60  -20   0:06:36 0.2% nfsd/578
102602 root     5240K 4328K sleep   60  -20   0:01:15 0.2% nfsd/649
   534 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0  15   0  15   0 httpd/1
Total: 163 processes, 275 lwps, load averages: 0.07, 0.07, 0.07
```

The `prstat -m` output (CODE EXAMPLE 3) includes the percentage of time a process spends in system traps, text page faults, data page faults, waiting for user locks, and waiting for CPU time. This will give you more information about where a slow application is using its time. This may get you closer to answer the question "Why is the application slow?"

CODE EXAMPLE 3 `prstat -m` Output

```
 PID USERNAME USR SYS TRP TFL DFL LCK SLP LAT VCX ICX SCL SIG PROCESS/NLWP
   739 root     0.3 0.3 0.0 0.0 0.0 0.0  99 0.0 126   3 345   5 Xsun/1
 15611 root     0.1 0.3 0.0 0.0 0.0 0.0 100 0.0  23   0 381   0 prstat/1
  1125 tlc      0.3 0.0 0.0 0.0 0.0 0.0 100 0.0  29   0 116   0 gnome-panel/1
 15553 rmc      0.1 0.2 0.0 0.0 0.0 0.0 100 0.0  24   0 381   0 prstat/1
  5591 tlc      0.1 0.0 0.0 0.0 0.0  33  66 0.0 206   0  1K   0 mozilla-bin/6
  1121 tlc      0.0 0.0 0.0 0.0 0.0 0.0 100 0.1  50   0 230   0 metacity/1
  2107 rmc      0.0 0.0 0.0 0.0 0.0 0.0 100 0.0  25   0  36   0 gnome-termin/1
   478 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0  17   0  14   0 squid/1
   798 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0  11   0  23   0 Xsun/1
  1145 tlc      0.0 0.0 0.0 0.0 0.0 0.0 100 0.0  25   1  34   0 mixer_applet/1
  1141 rmc      0.0 0.0 0.0 0.0 0.0 0.0 100 0.0  25   0  32   0 mixer_applet/1
  1119 tlc      0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   5   0  40   0 gnome-smprox/1
```

CODE EXAMPLE 3 `prstat -m` Output *(Continued)*

```
 PID USERNAME USR SYS TRP TFL DFL LCK SLP LAT VCX ICX SCL SIG PROCESS/NLWP
  1127 tlc      0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   7   0  29   0 nautilus/3
  1105 rmc      0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   7   0  27   0 nautilus/3
   713 root     0.0 0.0 0.0 0.0 0.0  85  15 0.0   2   0 100   0 mibiisa/7
   174 root     0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   5   0  50   5 ipmon/1
  1055 tlc      0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   5   0  30   0 dsdm/1
 15493 rmc      0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   2   0   4   0 rlogin/1
  1103 rmc      0.0 0.0 0.0 0.0 0.0 0.0 100 0.0   6   0   4   0 gnome-panel/1
  1025 tlc      0.0 0.0 0.0 0.0 0.0  20  80 0.0  29   0  45   0 utaudio/5
```

The `prstat -t` output (CODE EXAMPLE 4) lists the CPU and memory resource usage summary for each user. This is also very useful to determine where the resource consumption is. As an example, if an application is slowed down because a single user has issued a complex and inefficient query.

CODE EXAMPLE 4 `prstat -t` Output

```
 NPROC USERNAME  SIZE   RSS MEMORY      TIME  CPU
    128 root      446M  333M   1.4%  47:14:23  11%
      2 measter  6600K 5016K   0.0%   0:00:07 0.2%
      1 clamb    9152K 8344K   0.0%   0:02:14 0.1%
      2 rmc      7192K 6440K   0.0%   0:00:00 0.1%
      1 bricker  5776K 4952K   0.0%   0:00:20 0.1%
      2 asd        10M 8696K   0.0%   0:00:01 0.1%
      1 fredz    7760K 6944K   0.0%   0:00:05 0.1%
      2 jenks    8576K 6904K   0.0%   0:00:01 0.1%
      1 muffin     15M   14M   0.1%   0:01:26 0.1%
      1 dte      3800K 3016K   0.0%   0:00:04 0.0%
      2 adjg     8672K 7040K   0.0%   0:00:03 0.0%
      3 msw        14M   10M   0.0%   0:00:00 0.0%
      1 welza    4032K 3248K   0.0%   0:00:29 0.0%
      2 kimc     7848K 6344K   0.0%   0:00:25 0.0%
      4 jcmartin   13M 9904K   0.0%   0:00:03 0.0%
      1 rascal     17M   16M   0.1%   0:02:11 0.0%
      1 rab      3288K 2632K   0.0%   0:02:11 0.0%
      1 gjmurphy 3232K 2392K   0.0%   0:00:00 0.0%
      1 ktheisen   15M   14M   0.1%   0:01:16 0.0%
      1 nagendra 3232K 2400K   0.0%   0:00:00 0.0%
      2 ayong    8320K 6832K   0.0%   0:00:02 0.0%
 Total: 711 processes, 902 lwps, load averages: 3.84, 4.30, 4.37
```

2. **Next, try to find out if there is a real memory crunch on this system.**

   At this point, it is assumed that there is a performance problem perceived by the application users, or an application developer has found out that some batch processes are running slower than usual.

   Run `vmstat`. In Solaris 8 OS and above, if the scan rate (sr column) is very high, you have a real memory shortage. In older versions of the Solaris OS (before Solaris 8), in addition to non-zero scan rates, if you see physical disk I/Os to the swap device, you have a real memory shortage.

   A scan rate of a few hundred pages for a few seconds cannot be considered as high on a server with 128 gigabytes of real memory that is using a 16-megabyte memory page size. Conversely, a scan rate of a few hundred pages is high for a server with eight gigabytes of real memory that is using a 1-megabyte memory page size.

   On systems running Solaris OS versions older than Solaris 8, if priority paging is not turned on, the high scan rate may result in application pages currently being used to be paged out by the page scanner and the freed up memory could be used to cache file system pages. By turning on priority paging, (priority paging is obsoleted in the Solaris 8 and Solaris 9 OSs by the new cyclic VM page cache) pages associated with executables, shared libraries, and application process memory are given the highest priority and file cache pages are given the lowest priority. This method prevents application pages from being paged out by the page scanner until there is a real memory shortage.

Caution – DO NOT use `set priority_paging=1` on systems running Solaris 8 OS and above.

If the `vmstat` output shows high scan rates, further diagnosis can be done using `vmstat -p` (Solaris 8 OS and above) to find out if the pageouts are very high for application pages (which is bad for application performance) or for file cache pages.

Note – This option is not available before the Solaris 8 OS. In the Solaris 7 OS, this option is available as `memstat`, a simple command -line utility.

CODE EXAMPLE 5 shows the output of `vmstat -p`. In addition to overall paging statistics, it breaks down page-ins, page-outs, and pages freed for executable and library pages, anonymous pages (application heap and stack), and file pages.

CODE EXAMPLE 5 `vmstat -p` Output

```
<hostname>:/home/rramesh 3 % vmstat -p 5 10
     memory           page              executable    anonymous       filesystem
   swap  free     re   mf  fr  de  sr  epi  epo  epf  api  apo  apf  fpi  fpo  fpf
 48781328 2016832 60  1546 28   0  16    9    0    1    7   13   14   46   11   13
 40812320 242936  13  3417 27   0   0    2    0    0    2    0    0  158   27   27
```

CODE EXAMPLE 5 `vmstat -p` Output *(Continued)*

```
<hostname>:/home/rramesh 3 % vmstat -p 5 10
  40812384 237768  33  2043 65   0   0    2    0    0    0    0    0   18   65   65
  40811960 235752  71  1571 2    0   0    0    0    0   13    0    0  752    2    2
  40808736 229576  7   2305 3    0   0    8    0    0  513    0    0   32    3    3
  40806672 227608  198 3074 10   0   0  125    0    0  304    0    0  363   10   10
```

3. **For a more detailed analysis of memory usage, install the MemTool program.**

   This program is available at:

   `http://www.solarisinternals.com/si/tools/memtool/index.php.`

   This tool comes with several utilities and also has a GUI interface. After you install `memtool`, run the `prtmem` utility. This utility generates an overall breakdown of the memory layout on your system. For example, running `prtmem` on a system produces the following output:

```
Total Real Memory - 40 GBytes
Application Memory - 12 GBytes
Kernel Memory - 1 GBytes
File System Cache Memory - 26 GBytes
Free Memory - 1 GBytes
```

   In this system, the Solaris OS uses one gigabyte of memory for kernel pages. Oracle and other applications use 12 gigabytes. Because the Oracle application is very I/O intensive, most of the remaining memory is used to cache file-system pages.

Note – This is true only if the Oracle data files reside on file systems. This is not true if the Oracle database is on a raw device, or if direct I/O is turned on for the file systems in which the Oracle data files exists.

   This output gives you a clear idea of how memory usage is distributed in this system.

   Use caution while using `memtool` on a production system. Occasionally it could cause system panics if there are patch incompatibilities on the system.

An alternative to memtool's `prtmem` command in the Solaris 9 OS is the `::memstat dcmd` that is integrated into `mdb` (CODE EXAMPLE 6).

CODE EXAMPLE 6 `mdb` Output

```
# mdb -k
Loading modules: [ unix krtld genunix ip ufs_log logindmux ptm cpc sppp ipc
random nfs ]
> ::memstat
Page Summary                Pages                MB  %Tot
------------     ----------------  ----------------  ----
Kernel                       4550                35   15%
Anon                        13416               104   44%
Exec and libs                9222                72   30%
Page cache                   2945                23   10%
Free (cachelist)              433                 3    1%
Free (freelist)               114                 0    0%

Total                       30680               239
```

4. **Run the `memtool` GUI using the following command:**

```
# /opt/RMCmem/bin/memtool &
```

The `memtool` GUI has three types of displays—Process Memory, Process Matrix, and VFS Memory. First select the Process Memory display from the display type list. FIGURE 3 shows this display.

You will see a list of processes. For each process, the virtual column tells you the total virtual memory (RAM plus swap space) used by that process. The resident column tells you how much actual physical memory is used by the process. The shared column tells you how much resident memory is shared with other processes. For example, a shared library used by several processes. Finally, the private column tells you how much of the resident memory is used only by this process. This information is useful to determine how much real memory is used by this process.

5. **Click the Process Matrix tab on the Display type.**

The resulting display can be used to look at the virtual memory address space used by each process. FIGURE 4 shows this display. The process matrix shows the relationship between processes and their memory mapped files. Within a process's address space you can see the breakdowns of memory used by the various libraries and by heap (private, application allocated memory). This gives you more details about where the memory usage is located.

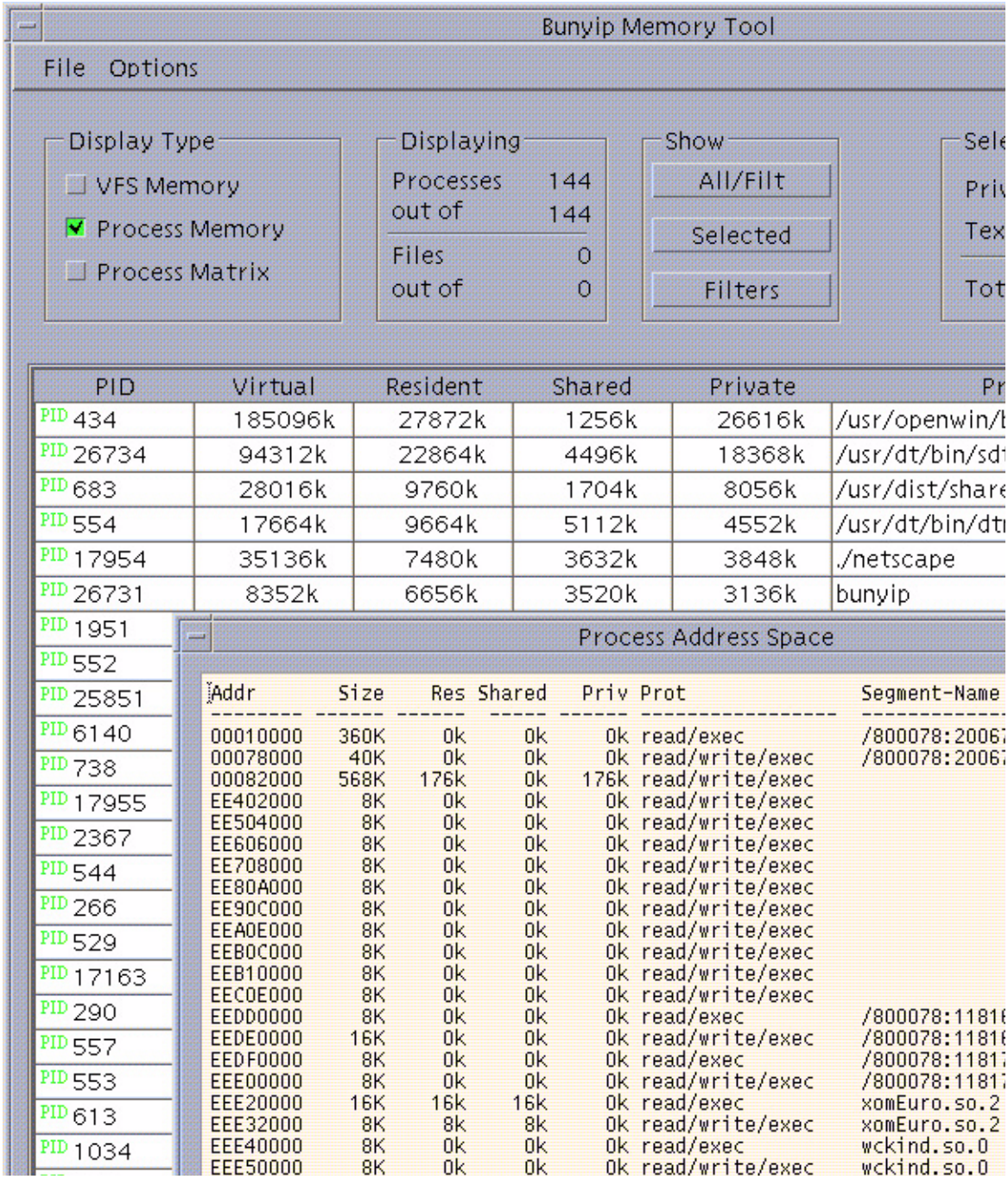
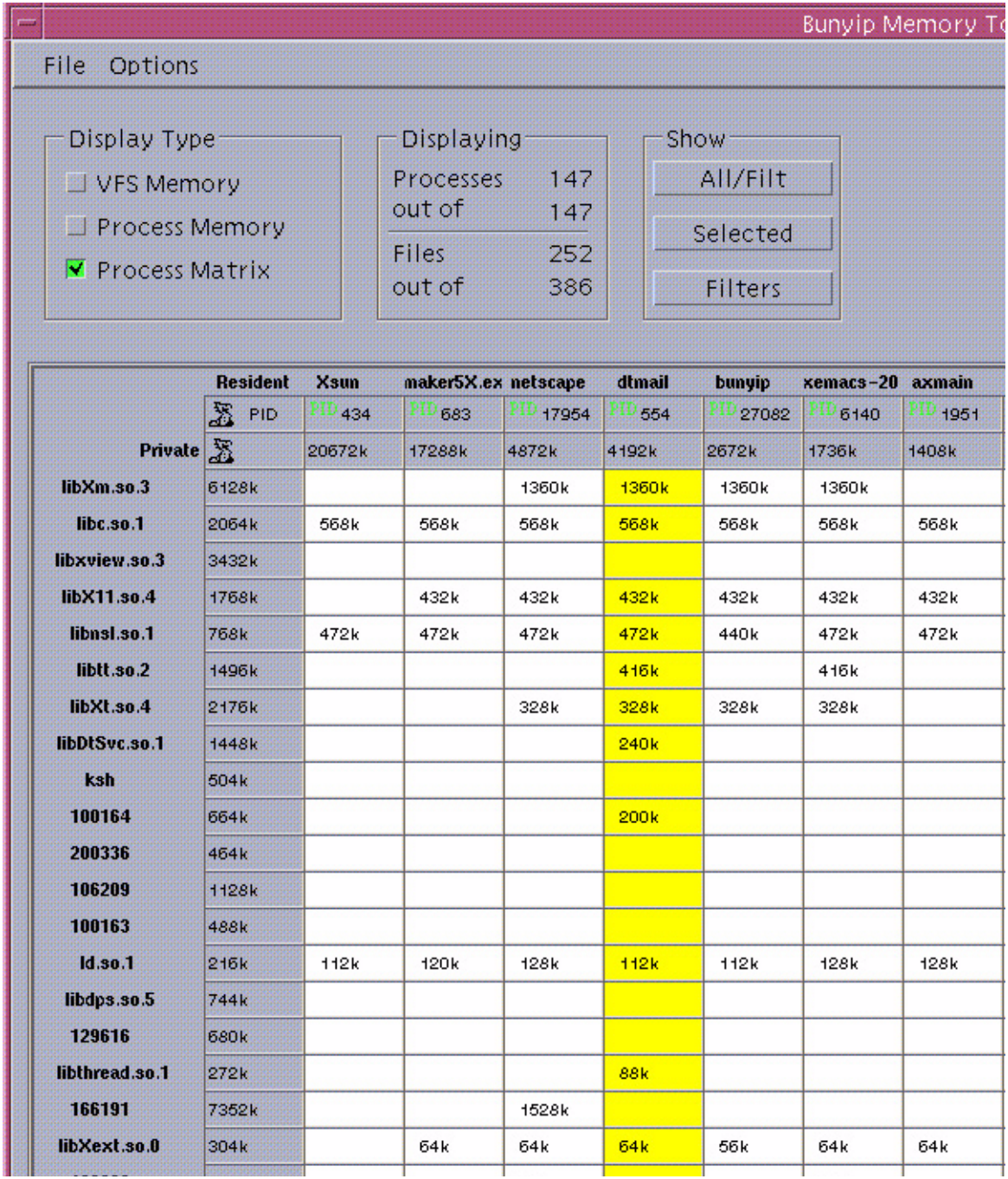


FIGURE 3 Process Memory Display



	Resident	Xsun	maker5X.ex	netscape	dtmail	bunyip	xemacs-20	axmain
	PID	PID 434	PID 683	PID 17954	PID 554	PID 27082	PID 6140	PID 1951
Private		20672k	17288k	4872k	4192k	2672k	1736k	1408k
libXm.so.3	6128k			1360k	1360k	1360k	1360k	
libc.so.1	2064k	568k	568k	568k	568k	568k	568k	568k
libxview.so.3	3432k							
libX11.so.4	1768k		432k	432k	432k	432k	432k	432k
libnsl.so.1	768k	472k	472k	472k	472k	440k	472k	472k
libtt.so.2	1496k				416k		416k	
libXt.so.4	2176k			328k	328k	328k	328k	
libDtSvc.so.1	1448k				240k			
ksh	504k							
100164	664k				200k			
200336	464k							
106209	1128k							
100163	488k							
ld.so.1	216k	112k	120k	128k	112k	112k	128k	128k
libdps.so.5	744k							
129616	680k							
libthread.so.1	272k				88k			
166191	7352k			1528k				
libXext.so.0	304k		64k	64k	64k	56k	64k	64k

FIGURE 4 Process Matrix Display

6. **Click on the VFS memory tab.**

   This displays the list of files that are being cached by the file system cache. FIGURE 5 shows this display. Note that all remaining memory not used by any process is automatically used for caching file-system pages.

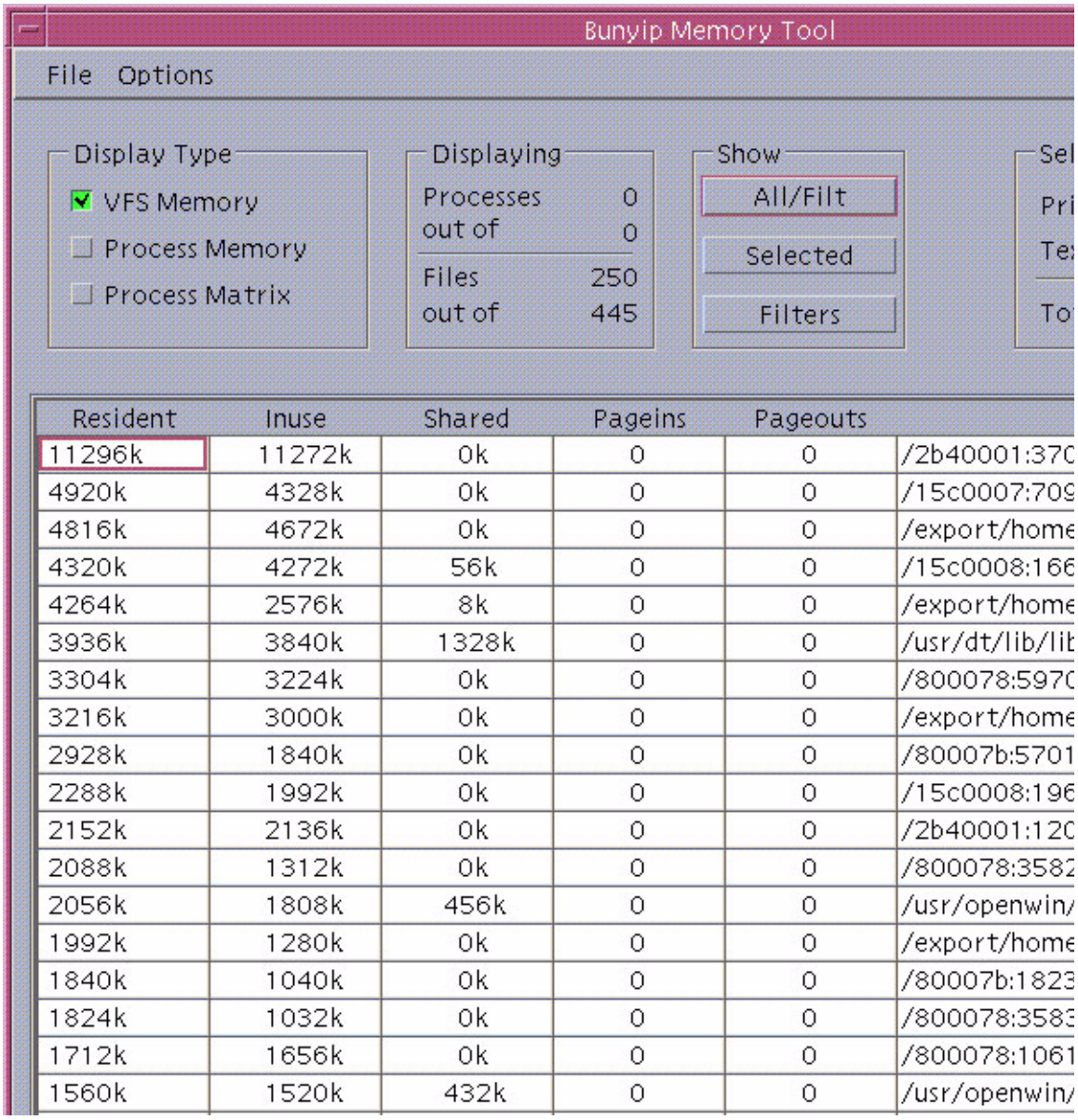


FIGURE 5 VFS Memory Display

So far you have seen how to use `memtool` to get detailed breakdowns of memory usage in the system. When used effectively, you can determine the reasons for memory shortage, which is usually caused either by very high application loads or poor application programming. Also, it is possible that the system may not be configured with enough memory for the amount of load on the system. You can identify memory leaks in an application by carefully observing the "private" column. If the number keeps growing indefinitely, it is possible that the application has a memory leak.

Intimate Shared Memory

Now you should investigate to see if Intimate Shared Memory (ISM) has been turned on. ISM is a performance-enhancing feature that can be used on systems that use shared memory. This feature is useful only if the application uses shared memory extensively.

ISM has the following key advantages and, in some cases, can boost performance by 100 percent or more:

- In addition to letting applications share the same real-memory segment, ISM locks the pages in real memory and prevents them from being paged out to disk.
- ISM causes all application processes to share the same memory segment and the page table entries (PTEs) for that segment. This results in fewer lookups in the address translation tables, which are used to translate virtual-memory addresses to real-memory addresses.
- ISM also uses a four-megabyte (large) page size, which decreases the number of address translations per application.

How do you determine whether or not an application is using ISM? Unfortunately, in Solaris OS versions prior to version 8, this information is not easy to determine. For example, there is no single command that will tell you this, but here is a simple four-step process that will help.

1. **Find the PID of the application process by using** `ps -ef | more`**.**
2. **Use** `pmap -x` *PID* **to find out if the application is using shared memory.**

   The third column is labeled Shared; if this column has a significant number of kilobytes for the corresponding row where the application name is found, this application uses shared memory. For example, there are other processes sharing the same memory segment.

3. **Find the process table slot for this PID.**

   Start the crash utility as the root user. Then type `p` to find out the slot number in the first column for the corresponding PID in the third column. Then type `p` *slot #* to confirm the slot number.

4. **Find out what segment driver this process is using to map the shared memory segment.**

   Type -f *slot #* from Step 2. This command lists the address space for this process. The sixth column lists the segment driver used to map this segment into the address space. If the driver used by this application is `segvn_ops`, the application does not use ISM. If the driver used is `segspt_shm`, this application is using ISM. This driver applies Oracle applications too. Solaris 8 OS and above, the `pmap` utility displays ISM segments in a process address space.

   In the Solaris 8 OS, it is easy to find out whether or not a process is using ISM. Find the PID of the process, then type the following command

```
# pmap -x PID
```

   Look for the `segspt_shm` driver in the shared column. If this driver is being used, ISM is being used.

Detecting Memory Leaks

Sometimes legacy applications have memory leaks, and the customer may already suspect this. To confirm whether or not an application has a memory leak, use the following procedure.

1. **Find the PID of the application process and run** `pmap -x` *PID*.
2. **Look at the column labelled Mapped file.**

   Under this column, look for the heap segment.

3. **Find the corresponding entry in the Private column.**

   This is the heap size of the application. The heap size corresponds to the amount of memory allocated by the application through `malloc`.

   For applications that have memory leaks, the heap size will continue to grow indefinitely at different rates based on how serious the leak is. Running `pmap` repeatedly can help detect memory leaks.

Note – In some applications, the heap size will grow to a large number immediately after startup, and then reach a steady-state value. In that case, it does not constitute a memory leak.

Once a memory leak is confirmed, there are several tools you can use to further narrow down the root cause. You can use the `truss` command to trace through all the system calls executed by the application, and to identify the behavior of the application during the leak. At this point, further understanding of the application is required.

The following is example for a hard-to-detect memory leak problem that was encountered at a customer site. The customer's application was loading rows from a database table into a fixed-size array. The application developer thought that the length of the row could never exceed a certain number *x* based on recent activity, so the developer fixed the size of the array to *x*. But when the system went into production, there were other Internet-based systems that updated the database with row lengths longer than the value *x*. This situation resulted in a memory out-of-bounds error.

Memory Tuning Preferred Practices

1. For the Solaris 7 OS, explicitly turn on priority paging to improve application performance. Set the following tuning parameter in the `/etc/system` file:

```
set priority_paging=1    (Solaris 7 Only)
```

This parameter should not be set for Solaris OS versions 8 and above because the file system pages are considered separately and can be freed without the page scanner. This reduces the possibility of severe paging problems.

2. Page-outs are normal if the database is on a UFS file system. Using direct I/O (by setting the `forcedirectio` while mounting the database file systems) and bypassing the file system cache may improve database performance significantly, but this should be done only for file systems in which database files and redo log files exist. If direct I/O is used and there is not enough database buffer cache, it may even decrease the performance by moving the problem from double buffering to a lack of database buffer cache. So, this performance tuning must be planned carefully, and the database buffer cache should be sized properly. The direct I/O option should not be used for other file systems used by other applications because they still need the UFS buffer cache.

2. Make sure that ISM is turned on at the database level for database servers. ISM will prevent database pages from being paged out.

3. Move applications, other than the database, to other servers if they are high-resource consumers.

4. Add more memory to your system only if:

- You are certain that you are already using direct I/O for the database.

- You have low database level cache hit rates
- You need more memory to increase the Oracle system global area (SGA) size (by increasing the database buffers)

Identifying CPU Level Bottlenecks

The SE Toolkit helps you identify whether there are CPU bottlenecks. If the `runqueue` size (r column) is large in the output of `vmstat`, it is generally not good, but for a system with a large number of CPUs, this may be normal. If the blocked on I/O (b column) has a high number, it is not a CPU issue, but rather a disk I/O issue.

By default, running `toptool` (see "High-Level Diagnosis" on page 2) sorts the output based on CPU utilization. At the top of this output, you can see the top few applications with heavy CPU usage. You can use the `mpstat` utility to check CPU utilizations. Sometimes, even when all of the CPUs on a system show some percentage of idle time, you can get a multiprocessor overload message from the SE Toolkit utilities. This message further indicates that there are too many spins on `mutex` locks.

The mutex locking mechanism in the Solaris OS is used to share certain resources. Traditionally, if a process needs a mutex lock, it will either *spin* or *block*. If a process spins, it essentially wastes CPU cycles but does not get context switched out. If it blocks, it gets context switched out but lets other processes use the CPU cycles without wasting them. The disadvantage of this is that the blocked processes have to be context switched back into the CPU, which may result in several wasted CPU cycles.

To take care of this problem, the adaptive mutex feature was introduced in Solaris OS version 2.6. With this feature, if a process needs a `mutex` lock, and if the process holding it is already running on another CPU, this process will spin, otherwise it will block.

On a customer's system, running `lockstat` to further analyze the spin on mutexes showed that there were a total of 96,921 adaptive mutex locks at the time of the 5-second snapshot. There were actually 8,000 mutex stalls per second.

The top lock (highest number of locks held) was for ph_mutex+0xe8. The caller was page_create_va+0x694. The lock count for this particular lock was 50,223.

Note – At this point, the `page_create_va kernel` routine must be further investigated. This gets into the realm of the kernel developers at Sun. You can defer to Sun support services to escalate the problem so it can be brought to the attention of Sun's kernel engineers.

You can also use the `truss` system call to trace a process. If `truss` shows a high number of `lseek` system calls being used, it indicates that the application is using the `read`() and/or `write`() system call.

To reduce the number of system calls, the preferred practice is to use the `pread`() and `pwrite`() system calls which do not need an `lseek`.

Preferred Practices Related to CPU Performance

1. Use the `vmstat` output r `column` to find the run queue size. Use this information and the number of CPUs on the system to find out if the run queue size is really too high.
2. Use `toptool` or `prstat` to find out the top few processes using most of the resources. If any of them are not database processes, see if you can move them to other systems.
3. Use `prstat` with various options as discussed in "To Diagnose Memory-Related Problems" on page 4.
4. Use Solaris OS dynamic reconfiguration (DR) to add resources to the system. Adding resources is possible only on high-end Sun systems using dynamic system domains, and only if there is an open slot in which to add another system board.

Preferred Practices Related to Tuning Disk I/O Performance

Run the `statit` program on your system for a few minutes. The `statit` program is available at:

`Http://www.solarisdatabases.com/#Utilities`

1. When VERITAS volumes striped out of several disk drives are used to store Oracle data, it will not be clear from the output of `iostat` or `statit` whether the I/O load balancing is good enough. In that case, you must ensure that the VERITAS volumes are load balanced properly. This can be done by running `vxstat`, which is available in the VxVM binaries directory. Run `vxstat` by entering:

```
# vxstat
```

The output looks as follows:

```
OPERATIONS                  BLOCKS                AVG TIME(ms)
TYP NAME   READ     WRITE    READ       WRITE     READ WRITE

Fri Jul 25 17:46:27 2003
vol disk01 6130656  804432   265482998  75692270    3.1 7.0
vol disk02 11581349 761603   471784276  75035712    2.8 7.4
vol disk03 17430583 19262819 3127664814 810497081  15.1 3.3
vol disk04 17223344 20607064 3076194388 985526668  14.8 3.7
vol disk05 18016195 19362227 3165996510 847390081  14.3 3.5
vol disk06 18052481 21862803 3154963120 1016905958 14.4 3.6
vol disk07 18120065 19020421 3146701564 788101424  13.2 3.6
vol disk08 18157402 20811471 3133572096 956931071  14.0 3.9
vol disk09 17510325 18539485 3092326865 753458786  13.9 3.6
vol disk10 17663278 19499439 3114184122 831997766  13.5 3.9
vol disk11 17285112 17906006 3111502168 664347037  13.7 3.6
vol disk12 17045511 19081933 3093295772 758543460  14.9 3.8
vol disk13 11857014 20131782 3106909298 651882491  21.7 3.2
vol disk14 15490428 35125035 3299428944 1252929805 18.9 2.6
vol disk15 14723793 33604499 3217815302 1140042147 20.7 2.4
vol disk16 14935665 35615012 3227176072 1308088459 20.5 2.6
vol disk17       71 31527    520        63346       3.5 1.4
```

If the last two columns, namely the average read and write times in milliseconds is too high for some volumes, compared to all the other volumes, balancing the I/O evenly across all volumes may be required.

2. For Online Transaction Processing (OLTP), Disk I/O utilization greater than 60 percent and disk response time (`srv-ms` output from running `statit`) greater than 35 milliseconds is not good.

3. For DSS, in addition to the OLTP numbers, if many other I/O requests are queued (as seen in the wait queue length in the output of `statit`), that is, if queueing time contributes to response time, it is not good.

4. The `iostat -p` output shows disk statistics at the partition level, this information is useful to find out which data file is responsible for a performance problem. If you are using the VERITAS Volume Manager, using `iostat` is not very useful. Instead, use `vxstat`; it will give you information about volumes that are associated with high I/O.

5. Increasing the stripe width (the number of disks used to create a volume) improves performance.

6. Try not to have more than one volume associated with a database on the same disk.

7. Increase the database buffer cache size in addition to using direct I/O.

8. Use more disk controllers and disk spindles for each data file.

Tuning Swap Space

Solaris OS version 8 and above have a new redesigned virtual memory (VM) system. A new utility called `prtswap,` which is part of `memtool` can be found in `/opt/RMCMem/bin/prtswap`. Running `prtswap` on the customer's system discussed previously produces the following output:

Virtual Swap Total	68.359 GBytes
Virtual Swap Reserved	12 GBytes
Free	56.359 GBytes
Physical Swap Configured	28.359 GBytes
Physical Swap Free	28.289 GBytes

Consider this output. The total physical memory configured on this system is 40 gigabytes. The total physical swap space (swap space on disk) configured on this system is 28.359 gigabytes as shown in Physical Swap Configured. The sum of the total physical memory (RAM) and the Physical Swap Configured will be the Total Virtual Swap on this system, shown in Virtual Swap Total as 40 + 28.359 = 68.359 gigabytes. The new virtual memory system considers all of the real memory and all of the swap space as one big virtual memory address space.

Note that Physical Swap Free on this system is 28.289 gigabytes. This means 28.359-28.289 gigabytes = 0.07 gigabytes, that is, only 70 megabytes of actual disk swap is being used, or only 70 megabytes of pages have been paged out to disk. This tells you that this system is not paging out too much.

The Virtual Swap Reserved is 12 gigabytes. This is the same as the application memory size shown in the `prtmem` output. This is because for each application page a corresponding backing storage is reserved on disk even though it is not being used. This storage is just in case the system reaches critical shortage and the page has to be copied out to disk during a write.

On the customer's system, when running `vmstat`, the scan rate shown in the `sr` column is very high, on the order of 3000+ pages per second. This does not necessarily indicate a problem, because they have set the `fastscan` parameter in the `/etc/system` file on this system to the maximum allowable. This makes sense because they are using the file system extensively for I/O to the database, the page scanner has to scan 26 gigabytes of file-system cache pages as fast as possible to get the best possible file system performance.

If the scan rate is too high, it is likely that the database is running on the Unix File System (UFS) without direct I/O. At a minimum, the redo log files should be on direct I/O, and if the SGA buffer cache is large enough, the data files should also be on direct I/O.

Preferred Practices Related to Swap Space

Make sure that there is enough backing storage for application pages in memory.

ORACLE Performance Tuning

ORACLE has been working on improving the performance of the Oracle databases at a feverish pace, as evidenced by several recent new versions of the software. Oracle realized that customers had started using Oracle for very large (multiple-terabyte) databases. To help customers maintain and tune these large databases, the company has come up with several self-tuning features in Oracle9i.

This section discusses the latest tools available to tune Oracle up to the latest release of Oracle8i.

The first thing to check when tuning Oracle is making sure ISM is turned on. In Oracle version 8.0.x, this can be done by setting the init.ora parameter `use_ism=true`. In Oracle 8.1.x, ISM is turned on by default, so no action is needed. If you are still not sure if the system you are tuning has ISM turned or not for the Oracle database, follow the procedure outlined in “Intimate Shared Memory” on Page 12. The advantages of using ISM are also clearly outlined in that section.

Collecting and Analyzing Oracle Performance Data

This section discusses how to tune the Oracle database if you suspect a performance issue. and outlines two methods of collecting Oracle performance data. The first method uses SQL queries to collect data from the well-known v$ Oracle tables such as `v$system_event`, `v$latch`, and so on. The second method uses Oracle-supplied utilities such as `utlbstat` and `utlestat` (formerly known as `bstat` and `estat`) to collect Oracle performance data.

Method 1

In this method, you start by looking at the alert log for the last few days for the Oracle database instance for which performance data is being collected. Often the database administrator for the system can tell you where it resides.

Look for anything unusual in the alert log. This requires a good understanding of how the Oracle SGA works to be able to glean relevant information. In the alert log, you may notice too many log switches;. For example, there is approximately one log switch every 35 seconds or so, and there are some messages indicating that the archiver cannot keep up.

A log switch occurs when the log writer (LGWR) process in Oracle has completed filling up a redo log file group and starts writing to a new one. A redo log file group consists of mirror files for a redo log file. Most systems, and one customer's system in particular, have the `log_check_point_timeout` interval set to a value to ensure that a checkpoint does not take place in between log switches—this is good for this system. The log switches are quite frequent and, during each one, the checkpoint process (CKPT) causes the DBWR process to write all the dirty database buffer contents to a database file. You should investigate the reason for the frequent redo log switches.

First you may want to confirm that the log switch is indeed a performance bottleneck on this system. Run a simple query to check for event waits that will tell you if the DBWR or LGWR is waiting on anything. The DBWR is the Oracle database writer process. This process reads the Oracle buffers in the SGA (in real memory) and writes them to database files on disk. The LGWR is the log writer process, which reads the redo log buffers in memory and writes them to redo log files on disk.

Here is the query:

```
Select event, total_waits, time_waited
From v$system_event
Where event like `%file%'
Order by total_waits desc;
```

The output of this query lists various event waits. Look at each one of these event waits. The average wait time is reported in centiseconds (1/100th of a second), but the author has converted the averages to milliseconds in this document.

```
Event Wait 1
Event: Log file switch completion
```

Total waits:	801
Total time waited:	78613 (1/100th of a second)
Average time waited:	981 ms (786130 ms / 801 = 981ms)

The `logfile` switch completion is taking a long time because there are too many log switches. A log switch takes place when a redo log file is full and the next redo log file has to be used. The redo log files are too small (only 50 megabytes each) for a 1.6 terabyte database. Although there are eight redo log groups, there is not enough time for the archiver (ARCH) to finish its work (copying redo log files to archive log files), and this results in waits for redo-log-switch completion.

Recommendations:

The following methods are ways to fix this problem.

- Increase the redo log file sizes to 500 megabytes each and check if the log-switch times have improved; if not, increase them to one gigabyte each. One possible side effect of making the log files too large, is that, if the database crashes just before a log switch, the time to recover the database will be higher. This is because, in this case, the log checkpoint interval is set to checkpoint only during log switches and there are no checkpoints in between log switches. To resolve this problem, the log checkpoint interval can be tuned so that there will be a couple of additional checkpoints in between log switches. There is always a trade off between increasing and decreasing the frequency of checkpoints. Increasing the frequency of checkpoints results in more writes to disks which will affect database performance. Decreasing the frequency of checkpoints will affect the time to recover the database.
- Increase the number of ARCH processes so that archive logging can go on in parallel.
- Separate the archive log, redo log, and data files on different physical disk spindles and disk controllers.
- Disable redo logging and archive logging on certain tables (for example, the temporary table space). This feature (no logging) can be used to turn off archive logging when large tables are being loaded.

```
Event Wait 2
Event: Db file sequential read
```

Total waits:	8547549
Time waited:	3245440
Average wait time per event:	3.79 ms

Note – An average wait time of 15 milliseconds or more is considered poor response time for read requests on a cached read / write system. So this is within the range.

```
Event Wait 3
Event: Log File Sync
```

Total waits:	135871
Time waited:	320101
Average time waited per event:	23.5 ms

Note – This wait event occurs when a commit is issued, and the session must wait for the redo buffer entry to be written to disk to guarantee that instance failure will not roll back the segment. Since the average wait time is higher than 15 milliseconds, this is considered poor performance.

Recommendations:

The following measures will help alleviate this problem.

- Make sure the redo log files and the data files are not on the same physical disk spindle or disk controller.
- Ensure that the redo log files are striped across several disks.
- Ensure that the redo log mirrors are on separate physical disk spindles.
- Make sure there is enough cache in your storage arrays.

Although the DBA is able to separate the files across different logical volumes presented, it is possible that the underlying physical disk spindle / controller is the same.

Note – The following naming scheme for presenting the volumes to the DBA is suggested. All volumes coming out of the same physical disk could be given the same prefix, so that the DBA will clearly know whether or not any two volumes are on the same physical disk.

Example:

Physical disk A, Physical disk B, Physical disk C, each 36 gigabytes in size.

VolA1, VolA2, VolA3 from disk A
VolB1, VolB2, VolB3 from disk B
VolC1, VolC2, VolC3 from disk C

Note – This naming scheme looks simple, but if striping is used, it can become more complex when each volume is pieced together by using parts of disks A, B and C.

For example:

VolABC1 from disk A, disk B and disk C
VolABC2 from disk A, disk B and disk C
VolDEF1 from disk D, disk E and disk F
VolDEF2 from disk D, disk E and disk F

```
Event Wait 4
Events: DB File Parallel write, DB File Single write, DB file
Parallel read
```

```
Average Wait Times in order: 182 milliseconds, 28.4
milliseconds, 220 milliseconds
```

Recommendation:

All of these values are too high and are affecting performance on this system. The solution is to balance the I/O across several faster disk spindles and disk controllers.

```
Event Wait 5
Event: Redo log space requests
```

A non-zero value for this column in the `v$system_event` view means that the redo buffer size is not big enough.

Recommendation:

Increase the redo buffer size. Currently it is at one megabyte. It should be at 128 kilobytes times the number of CPUs, or 128 kilobytes, x 40 = 5 megabytes.

Tuning Redo Buffer Latches

First, issue the following query to find out if there are any waits on allocation latches.

```
Select name,
     Sum (gets) "Gets"
            Sum (misses) "Misses"
            Sum (immediate_gets) "IM_GETS"
            Sum (immediate_misses) "IM_MISSES"
            Sum (average_wait) "AVERAGE_WAIT"
 From v$latch
 Where name like `%redo%'
 Group by name;
```

Oracle has only one allocation latch per instance. On this system, the wait on allocation latch is still very high. According to the DBA, in this version of Oracle, the `init.ora` parameters `log_small_entry_max_size=0` and `log_simultaneous_copies=0` (2x the number of CPUs) are not supported. Tuning these parameters sets up 80 copy latches to remove the bottleneck of having just one allocation latch.

Recommendation:

Generally, the recommendation is to make the instance use 80 copy latches instead of one allocation latch. Check with Oracle to find out how to implement this in the latest version of Oracle.

Data Dictionary Cache and Library Cache Misses

Recommendation

Increase the SGA size and also increase the `shared_pool` size to improve the hit rate on the data dictionary caches and library caches.

Method 2

This method involves running tools supplied by Oracle for data collection, `utlbstat` and `utlestat`. The `utlbstat` script gathers the initial performance statistics. The `utlestat` component gathers performance statistics at the end of an observation period.

Here is a five-step process to collect data using these tools.

1. **Choose the correct time slice.**

   This involves deciding when to collect data. For example, if you are tuning an online transaction processing (OLTP) database, you may want to choose peak hours of usage when the most users are logged in and are issuing transactions. On the other hand, in the case of a decision support system (DSS) on which loads of the database take place for a few hours at night, those will be the hours best suited to run the tools.

2. **Check the initialization parameter file.**

   Set the `init.ora` parameter `timed_statistics=true`. If you are not able to restart the `database`, use the following command to change it online:

   ```
   Alter system set timed_statistics=true
   ```

3. **Turn on `utlbstat` at the appropriate starting time.**

   ```
   Svrmgrl> $Oracle_HOME/admin/utlbstat
   ```

4. **Run `utlestat` at the end of the data-collection period.**

   ```
   Svrmgrl> $Oracle_HOME/admin/utlestat
   ```

5. **Analyze the output.**

   You can use the output from running these tools can be used to conduct most of the analysis outlined in “Method 1” on page 24.

   Alan Packer states in his book, *Configuring & Tuning Databases on the Solaris Platform*:

   > Oracle 8.1.6 also introduces the statspack scripts. With statspack, more data is collected and some useful ratios are precalculated. Refer to `$ORACLE_HOME/rdbms/admin/spdoc.txt` in the Oracle9i release and `$ORACLE_HOME/rdbms/admin/statspack.doc` in the Oracle8i release to learn how to install statspack.

After installing statspack using the `spcreate.sql` script (Oracle 9i) or the `statscre.sql` (Oracle 8i), create snapshots as the "use sqldba" using the following commands:

```
SQL> Connect perfstat/perfstat
SQL> execute statspack.snap;
```

To create a report, run the `spreport.sql` script (Oracle 9i) or the `statsrep.sql` (Oracle8i). The following example shows the appropriate syntax for Oracle 9i

```
SQL> @?/rdbms/admin/spreport
```

This script prompts for the IDs of the two previously created snapshots and, after prompting for a report file name, creates a report based on the activity occurring between the two snapshots.

Method 3

A GUI-based tool called "Spotlight on Oracle" from Quest Software does all the queries for you and shows you all the problem areas, such as top sessions, inefficient SQLs, wait on locks, latches, disk I/O, and other events. This software reduces the complexity of setting up the queries and scripts. It also increases accuracy. This software recommends appropriate corrective actions to alleviate performance bottlenecks. It is located at:

`http://www.quest.com/quest_central/qco/performance_diagnostics/.`

FIGURE 6 shows top sessions. The Session Details tab of the Top Sessions drill down lists all users connected to the Oracle database. User V31 is selected in the Top Sessions screen shown here, so session details on that connection are displayed in the bottom panel.

FIGURE 7 shows an inefficient SQL statement. The Top SQL drill down shows full SQL text and performance statistics for the SQL statement highlighted. From this window, you can also read a detailed explanation plan, which helps you pinpoint resource usage.

FIGURE 8 shows various activities including wait events. The Activity drill down shown here displays the Activity Summary tab, showing an overview of the activity on the database you are monitoring.

FIGURE 9 shows the disk I/O waits. The I/O Summary tab of the I/O drill down shows various indicators of I/O requests, such as total I/O rates and times as well as specific I/O rates per tablespace. This information summary is useful for isolating bottlenecks in disk and cache systems.

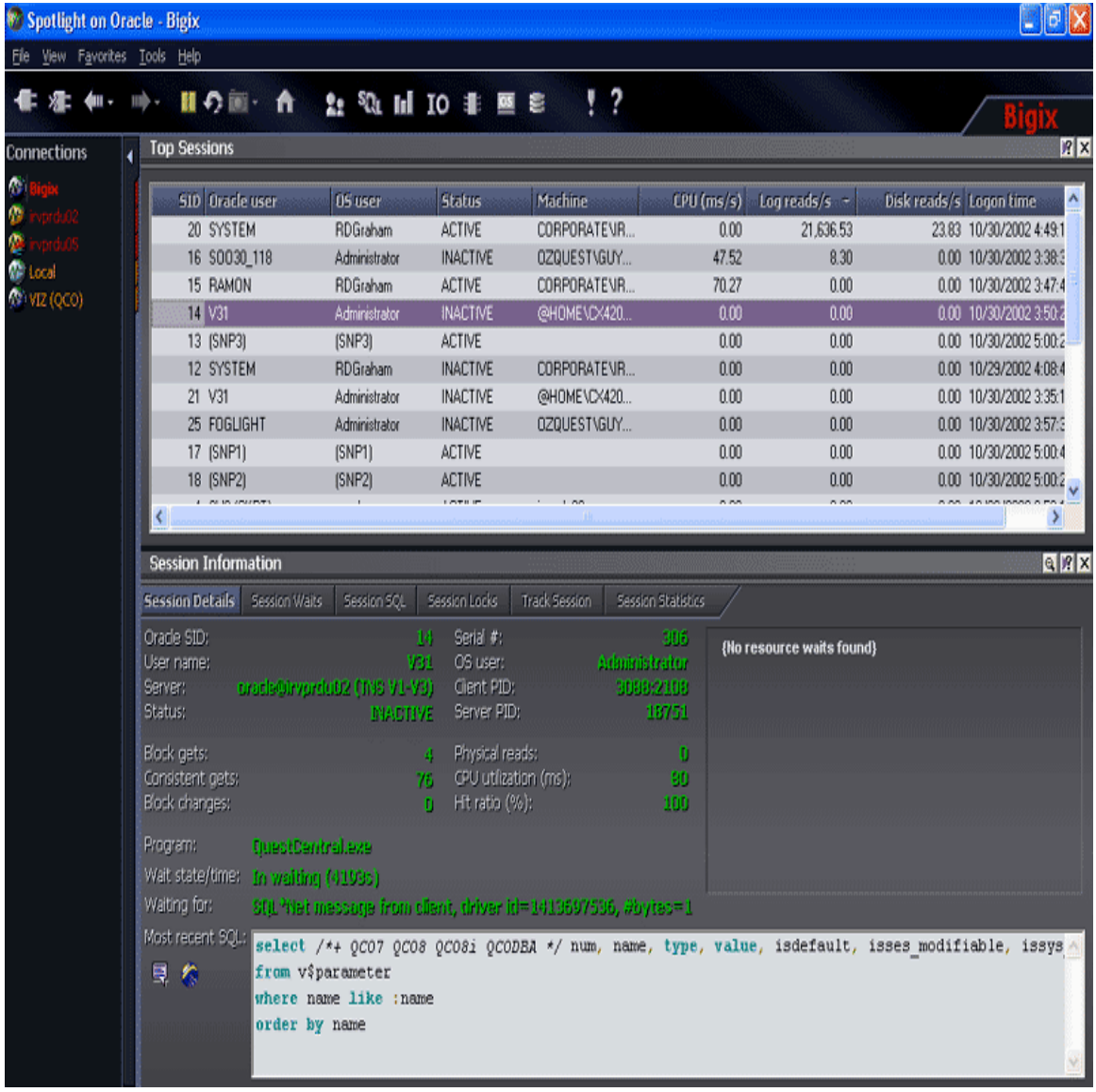


FIGURE 6 Top Sessions

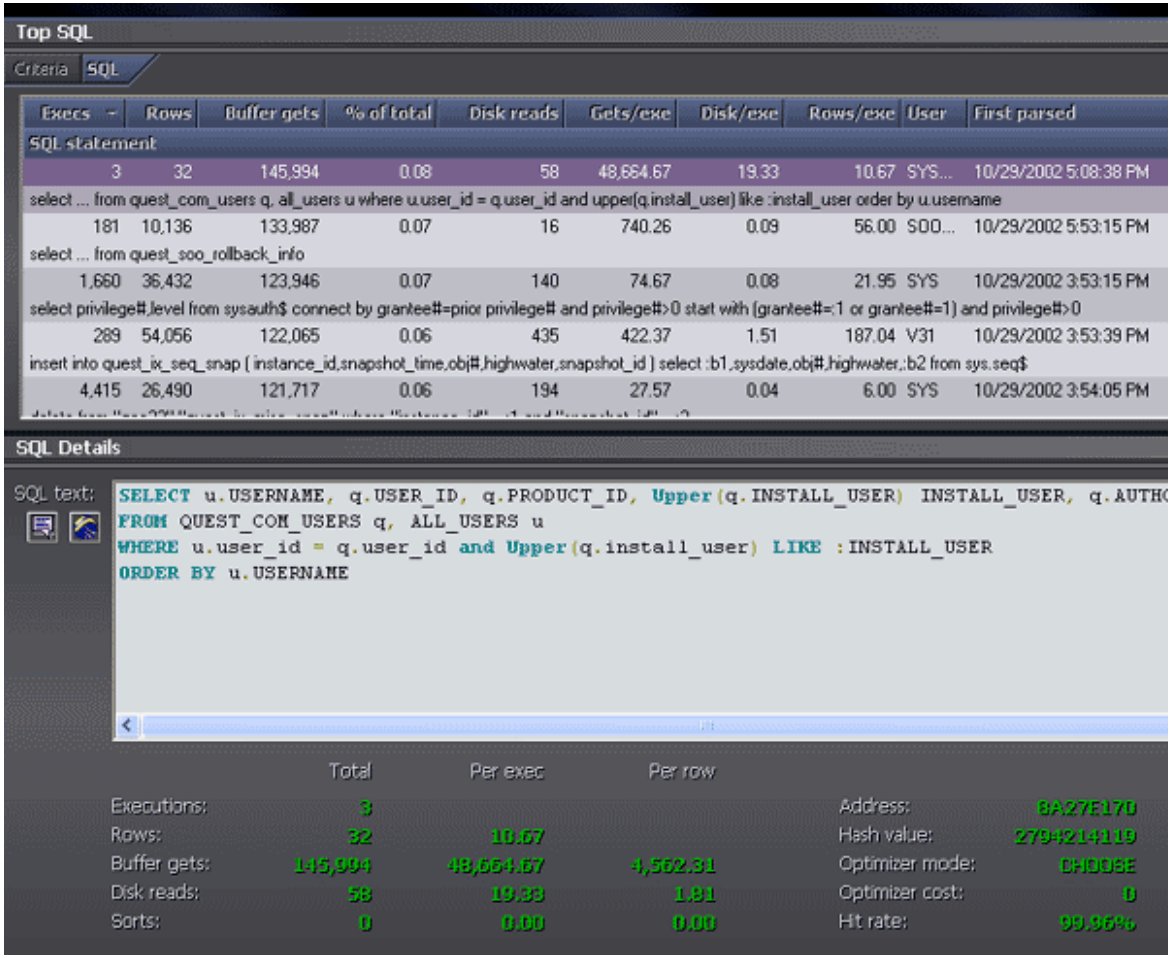


FIGURE 7 Inefficient SQL Statement

FIGURE 8 Activity Summary

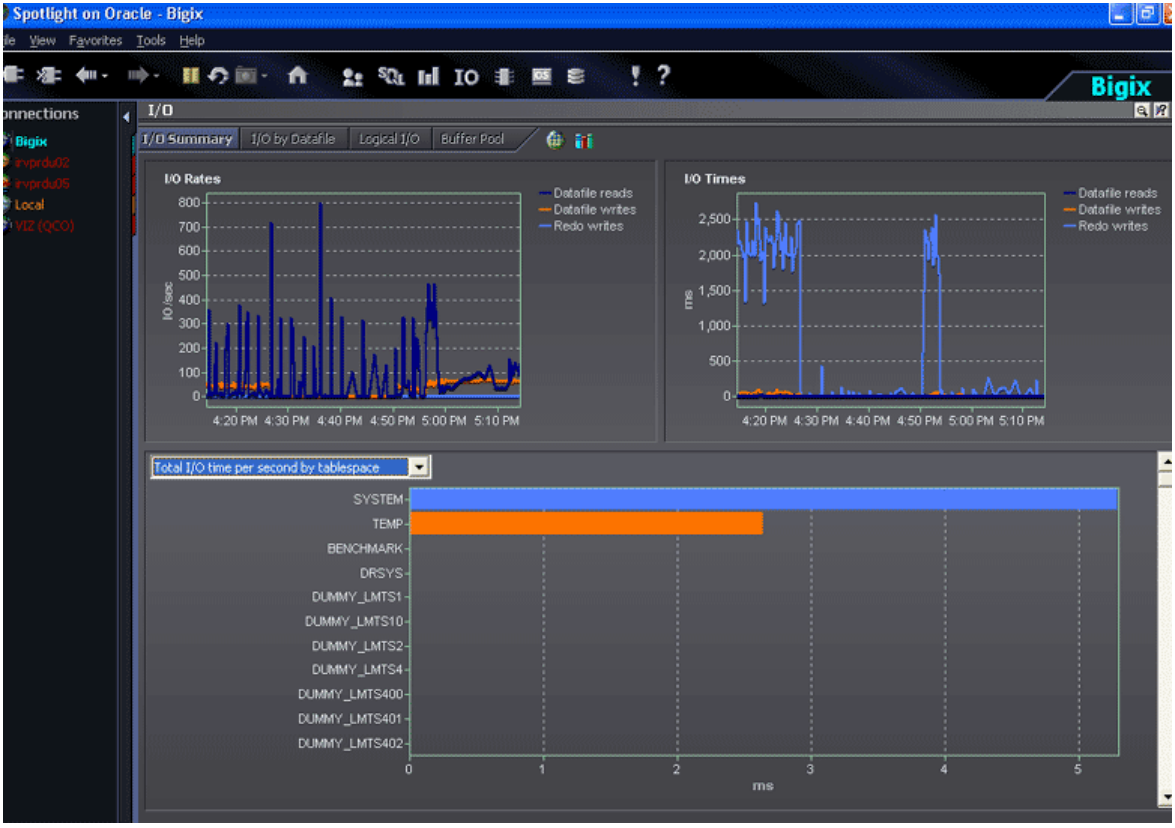


FIGURE 9 Disk I/O Waits

About the Author

Ramesh Radhakrishnan has been an IT Architect and Consultant at Sun Microsystems for the past four years. He conducts availability and architecture assessments and designs IT environments for several of Sun's mission critical customers. He is also called by customers to conduct performance analysis and patch management. He is the author of the *Sun BluePrints OnLine* article "A Patch Management Strategy in the Solaris OE."

Before joining Sun, Ramesh worked as a system administrator, IT consultant, and ClearCase Consultant. He has a Master's degree in Computer Science from Old Dominion University. Over the years he has gained experience in the areas of backup and recovery architecture, disaster recovery architecture, and IT processes, along with many other IT infrastructure management areas.

Recently Ramesh was part of a team that developed an architecture basics course for Sun engineers. He is currently working on obtaining his IT Service Management (ITSM) Master's Certification.

References

[1] Mauro, Jim and McDougall, Richard. *Solaris Internals—Core Kernel Architecture*, Sun Microsystems Press ISBN No. 0-13-022496-0

To access this book online, go to:
`http://www.sun.com/books/catalog/mauro_mcdougall.xml`

[2] Sneed, Bob. "Sun/Oracle Best Practices," *Sun BluePrints Online*, January 2001

[3] Packer, Allan, *Configuring & Tuning Databases on the Solaris Platform*, Sun Microsystems Press ISBN No. 0-13-083417-2

To access this book online, go to:
`http://www.sun.com/books/catalog/packer.xml`

[4] Cockcroft, Adrian and Pettit, Richard, *Sun Performance and Tuning—Java and the Internet*, Second Edition, Sun Microsystems Press ISBN No. 0-13-095249-4

Acknowledgements

We would like to acknowledge the contributions of Jim Mauro, Ted Persky, Allan Packer, Gamini Bulumulle, all of SMI, and David Berney and Scott Brye, both of Quest Software, to this article.