DRILL-DOWN MONITORING

The time has come. Armed only with your wits, a little common sense, and some basic system knowledge, you're going to crack the performance problem bedevilling your database server. You roll up your sleeves and seat yourself firmly in front of a keyboard. A cluster of slightly awed colleagues watches wide-eyed over your shoulder.

OK, perhaps I'm getting a bit carried away here. Suffice it to say that the aim of this chapter is to develop a simple method for identifying performance problems on database servers.

I'm assuming you have already looked at the issues affecting server performance (covered in earlier chapters of *Configuring and Tuning Databases on the Solaris Platform*). Your application's behavior is well understood and consultants in small doses have already done wonders with application performance. You've done what you can to fix any database schema design problems and the addition of a couple of crucial indexes has already calmed the users down a little.

You've checked out obscure things like environment variables and racked your brains for other issues that might need attention. But performance problems still persist. Perhaps you need to upgrade your hardware, but at this point you're not sure.

Where should you start? I'm going to suggest a five-step process that will walk you through the major components of the system: memory, disk I/O, network, and CPU, followed by database monitoring and tuning. If a problem becomes apparent in one of these areas, there may be further steps to narrow the problem. This kind of "drill down" approach is an effective way to identify and ultimately solve problems.

If you find a bottleneck (by which I mean a constriction of performance, just as the neck of a bottle limits the flow of liquid into or out of a bottle), does that mean you should look no further? I would suggest going through the whole process anyway to see what you can discover.

Bear in mind, though, that fixing a bottleneck in one place might expose another elsewhere. Suppose, for example, your system is paging severely due to a lack of memory, but no problems are apparent elsewhere. Adding memory might allow your throughput to improve to the point where one of the disks becomes overutilized, resulting in a disk bottleneck. Checking out the disks as well might give a hint of problems to come.

Once you've found and resolved a bottleneck, go through the entire process again.

Finally, is the order of the steps important? Of course there are many possible ways of tackling system monitoring, but I suggest you go through the steps in the order shown.

STEP 1. Monitoring Memory

To check for a memory bottleneck, use the vmstat utility, which shows, among other things, memory behavior for the system. A 5-second interval is a good choice for live monitoring.

The vmstat trace in Figure 1 shows a system with no evidence of memory shortfall.

Figure 1 *vmstat trace with no memory shortfall*

procs			memory			page						disk				faults			cpu		
r	b	w	swap	free	re	mf	pi	po	fr	de	sr	m1	m2	s6	sd	in	sy	cs	us	sy	id
0	8	0	3557104	1359368	0	621	0	0	0	0	0	0	0	0	0	417	11922	1190	12	2	86
0	7	0	3555728	1358080	0	729	0	0	0	0	0	0	0	0	0	449	12797	1985	20	2	78
5	9	0	3512184	1318120	0	3666	0	12	12	0	0	0	0	0	6	2198	32163	7404	70	10	20
3	15	0	3485016	1293944	0	939	0	24	24	0	0	0	0	0	1	892	30760	2842	50	4	46
0	18	0	3480520	1289912	0	813	0	0	0	0	0	0	0	0	1	616	27887	2895	31	4	65
0	17	0	3476216	1285992	0	547	0	3	3	0	0	0	0	0	0	516	31687	1716	21	2	77
1	16	0	3473000	1283232	0	542	0	6	6	0	0	0	0	0	0	663	43993	2112	30	3	67
2	16	0	3469568	1280368	0	712	0	8	8	0	0	0	0	0	1	666	39791	3176	34	3	62

Figure 2 shows a vmstat trace from the same system during a severe memory shortfall.

Figure 2   *vmstat trace with severe memory shortfall*

procs			memory		page						disk				faults			cpu		
r	b	w	swap	free	re	mf	pi	po	fr	de	sr	m1	m2	s6	sd	in	sy	cs	us	sy	id
0	31	0	2175384	47800	3	542	1	166	593	38656	141	0	0	0	1	689	26511	3549	19	4	78
0	27	0	2170608	47552	2	790	4	305	1116	45208	269	0	0	0	2	711	36787	6050	27	6	66
0	28	0	2168088	48704	4	788	1	190	432	47256	92	0	0	0	2	718	30558	3291	23	4	73
0	29	0	2164592	47664	1	699	8	158	574	47712	136	0	0	0	1	777	29870	3400	19	3	78
1	27	0	2162136	48184	2	734	9	140	403	42944	105	0	0	0	2	708	28258	3027	22	4	74
1	27	0	2158560	47688	0	498	4	166	606	38656	146	0	0	0	1	750	41527	3034	20	4	76
0	27	0	2155136	47408	1	489	6	240	796	38656	179	0	0	0	1	754	31926	3275	17	4	79
0	27	0	2151664	47824	1	581	6	187	622	47712	145	0	0	0	2	946	36169	3741	19	5	76

What to Look For

Look for `po` (pageouts—the kilobytes paged out per second) and `sr` (scan rate—the number of pages scanned by the clock algorithm). When both are consistently high at the same time (much more than 100 per second, say, on a system with up to 4 Gbytes of memory, more on a larger system), then it is possible the page daemon is being forced to steal free memory from running processes. Do you need to add more memory to the system? Maybe.

More memory may not help, though. That might sound crazy, but unfortunately the water is a little muddy here. Some explanation might help clarify the situation.

Pageouts can happen for a number of reasons, including the following:

- Dirty (modified) file system pages are being flushed to disk. Such flushing is normal behavior and does not represent a problem. If database files are placed on file system files, expect to see this kind of pageout.

- Application pages are being pushed out to the swap device to free up memory for other purposes. If the applications in question are active or about to become active, paging is bad!

- New memory has been allocated by an application and swap space is being assigned to it. This, too, is normal behavior and does not represent a problem.

- Memory pages have been freed by applications and are being flushed to disk. Isn't paging a waste of time if the memory is no longer required by the applications? You bet! Solaris 8 introduced a new `madvise()` flag called `MADV_FREE` to enable developers to tell the operating system not to bother to flush such pages to swap.

The *scan rate* is a measure of the activity of the page daemon. The page daemon wakes up and looks for memory pages to free when an application is unable to find enough memory on the free list (memory has fallen to the `lotsfree` system parameter). The greater the memory shortfall, the faster the page daemon will scan pages in main memory.

The major consumers of memory in a system are:

- Applications, including text (binary code), stack (which contains information related to the current phase of execution of the program and its functions), heap (which contains program working space), and shared memory.

- The file system page cache, which contains file system data (all file system disk blocks must first be read into memory before they can be used). This cache becomes important when database files are stored on file systems.

- The operating system kernel.

Normal Paging Behavior Prior to Solaris 8

Before Solaris 8, the `free` column reported by `vmstat` may not be a good indication of the available memory in the system. The reason is that once memory pages are used by the file system page cache, they are not returned to the free list. Instead, the file system data blocks are left in the cache in case they are needed again in the future.

When the page daemon detects a memory shortfall and scans for pages to free, it may well choose to free some of the pages in the file system page cache. If the pages have been modified, they are first flushed to disk. There is no simple way of finding out how much of main memory is being used by the file system page cache at any point, but you can bet it will be substantial if database files are located on UFS files rather than raw devices. The `memtool` package (Richard McDougall's memory monitoring tool), available on the book website, can identify memory use by UFS files.

The problem is that the page daemon may free application memory pages as well as file system page cache pages since it doesn't know which is which. The result can be severe paging and major performance problems. Adding more memory won't help much, either. It will simply mean that more database pages can be cached in the memory. Fortunately, there is a solution.

Priority Paging

As of Solaris 7, a new feature called *priority paging* has been added. Priority paging lowers the priority of file system pages in memory so that the page daemon will choose to free them ahead of application pages. This behavior can make a huge difference to paging problems; priority paging should be enabled wherever databases coexist with active file systems, and especially where database files are placed on file systems.

You can activate priority paging by adding the following line to `/etc/system` and rebooting:

```
set priority_paging = 1
```

Patches are available for Solaris 2.5.1 and Solaris 2.6 to add priority paging functionality. From Solaris 8, changes to the virtual memory system mean that priority paging is no longer required and should not be used.

UFS Files and Paging

If your database files are UFS files rather than raw devices, you may observe significant scanning even once you have enabled priority paging. In fact, the scan rate may increase since priority paging causes the page daemon to become active sooner. This behavior is a natural consequence of the need to bring all database pages into the UFS page cache before they can be accessed by the database. The ongoing need to find free memory gives rise to constant scanning activity on busy database servers using UFS files.

If your application carries out updates, inserts, and deletes, you should also expect to see pageout activity. All database writes must go through the UFS page cache before being written to disk. Although the page being written to disk would previously have been read into the UFS page cache, the memory might have since been reused if the scan rate is high. In that case the page must be reread from disk before the write to disk can proceed. This process in turn displaces another page, and the cycle continues.

How do you stop all this paging activity? To eliminate scanning (assuming you have enough memory for your applications), either use raw devices or mount your database partitions with the Direct I/O flag (forcedirectio).

A word of caution: eliminating paging activity with Direct I/O may not always result in instant performance improvements. The file system page cache acts as a second-level cache for database pages, and removing it from the picture will expose any inadequacies in the sizing of your database buffer cache. Make sure that your database buffer cache is adequately sized; otherwise, you may find yourself with plenty of free memory and a database buffer cache starved of buffers.

Enabling Direct I/O for database files, and especially for database logs, can offer significant performance benefits as of the Solaris 8 1/01 release; earlier versions of Direct I/O may not offer significant performance gains. Direct I/O is described in more detail in "Unix File System Enhancements" on page 21 of *Configuring and Tuning Databases on the Solaris Platform*.

As a final caution, although Direct I/O can prove very useful for database files, do not enable it for nondatabase files without first examining carefully the performance implications of doing so.

Normal Paging Behavior as of Solaris 8

As we have seen, priority paging doesn't go all the way to solving the problem. Although the page daemon will choose file system pages in preference to application pages, the page daemon still has to search through the whole of memory to find them. And if large database buffer caches are being used, file system pages may only represent a small proportion of the total memory, so a lot of searching will be necessary.

As of Solaris 8, file system pages are separately accounted for, so they can be freed without a memory scan to find them. Consequently, the page daemon is not needed at all unless there is a major memory problem. As a result, the likelihood of paging problems is greatly diminished.

Drilling Down Further

If you want to find out where the memory is going, there are a number of options:

- For the final answer on memory consumption, use `memtool`. The `procmem` script described below will provide a detailed breakdown of process memory usage. Both `memtool` and `procmem` are available on the book website.

- Run `dmesg` and look for `Avail mem`. The difference between available memory and physical memory (use `prtconf` or `/usr/plat-form/`arch -k`/sbin/prtdiag` to find out the physical memory) indicates the amount of memory reserved for the kernel.

- Use `/usr/ucb/ps -aux` to find out which processes are the major memory hogs. This command lists the percentage of memory used by each process. Beware, though! The memory listed is virtual memory, not physical memory, and it may not be a good indication of how much physical memory is actually being consumed by the process at any given moment.

If you have installed the unbundled `memtool` package, the `procmem` script will use the `pmem` program (similar to the standard Solaris `pmap` program, but with various bugs fixed on some Solaris releases). Some users are unwilling to install an unbundled package on a production system—`procmem` uses `pmap` instead if `memtool` (and therefore `pmem`) is not available. The `procmem` script summarizes memory use for all processes and gives a breakdown into resident, shared, and private memory usage. Please note that both `procmem` and `memtool` are unsupported software.

Passing the `-h` parameter to `procmem` results in the following usage information:

```
usage: procmem [-v] [-h | -p pidlist | [ -u username ] [ searchstring ]]

Examples:
procmem -p 10784 10759
        - show memory usage for processes with pids 10784 10759
procmem -u root
        - show memory usage for all processes owned by the root user
procmem -u "daemon root"
      - show memory usage for all processes owned by the root & daemon users
procmem netscape
        - show memory usage for process(es) in 'ps -ef' with "netscape"
procmem -u fred netscape
        - show memory usage for "netscape" processes owned by fred
procmem
        - show memory usage for all processes (provided current user has
          superuser access privileges)

Definition of terms
  'Kbytes' is the total memory size of the process or file.
  'Resident' is that portion currently occupying physical memory.
  'Shared' is resident memory capable of being shared.
  'Private' is resident memory unique to this process or file.
  Resident = Shared + Private
Sizing
```

> For reporting purposes, the 'Shared' component has been counted once
> only while the 'Private' component has been summed for each process
> or file. The /usr/lib shared libraries have been reported separately
> since they tend to be widely used across applications. To be totally
> accurate, though, the shared component of these shared libraries
> should only be counted once across all applications, not once for
> every group of applications. The same logic may apply to other
> shared libraries also used by multiple applications.

The -v flag offers additional detail. An example of procmem output follows
for all processes on a server running an Oracle database.

```
Processes                            Kbytes Resident   Shared  Private
---------                            ------ --------   ------  -------
Process Summary (Count)
 -csh (3)                              5376     2456     1064     1392
 -ksh (8)                             14624     2912     1456     1456
 automountd (1)                        4088     3656     1792     1864
 cimomboot (1)                         1576     1384     1256      128
 cron (1)                              1936     1744     1464      280
 devfsadmd (1)                         2776     2536     1592      944
 devfseventd (1)                       1272     1232      952      280
 dmispd (1)                            3160     2648     1744      904
 dtlogin (1)                           4920     2856     2192      664
 dwhttpd (2)                          20080     7440     4496     2944
 esd (2)                              24712    21768     3880    17888
 grep (1)                               968      936      832      104
 in.ndpd (1)                           1856     1488     1304      184
 in.rdisc (1)                          1616     1416     1272      144
 in.rlogind (6)                       10368     2464     1360     1104
 inetd (1)                             2648     2384     1384     1000
 init (1)                              1888     1608     1136      472
 iostat (1)                            1824     1784      904      880
 ksh (5)                               9040     2320     1456      864
 lockd (1)                             1896     1656     1264      392
 lpsched (1)                           3040     1768     1552      216
 mibiisa (1)                           2952     2752     1504     1248
 mountd (1)                            2952     2536     1528     1008
 nfsd (1)                              1888     1672     1264      408
 nscd (1)                              3200     2928     1528     1400
 ora_ckpt_bench (1)                 1405264  1372488  1371824      664
 ora_dbw0_bench (1)                 1406976  1374208  1371824     2384
 ora_dbw1_bench (1)                 1406968  1374200  1371824     2376
 ora_dbw2_bench (1)                 1406968  1374200  1371824     2376
 ora_dbw3_bench (1)                 1406968  1374200  1371824     2376
 ora_lgwr_bench (1)                 1405248  1372472  1371824      648
 ora_pmon_bench (1)                 1405696  1372920  1371824     1096
 ora_reco_bench (1)                 1405160  1372384  1371824      560
 ora_smon_bench (1)                 1405192  1372424  1371824      600
 oraclebench (50)                  70258944  1400768  1371824    28944
 powerd (1)                            1632     1576      976      600
 rpcbind (1)                           2584     2088     1296      792
 sac (1)                               1736     1488     1296      192
 sendmail (1)                          2936     2280     1816      464
 sh (4)                                4176     1320      912      408
 snmpXdmid (1)                         3744     3184     2024     1160
 snmpdx (1)                            2144     1920     1520      400
 statd (1)                             2592     2208     1464      744
 syslogd (1)                           4192     3008     1456     1552
 tail (1)                               968      936      792      144
 tee (2)                               1808      952      792      160
 tpccload (50)                       418800    38944     3344    35600
 ttymon (2)                            3464     1744     1336      408
 utmpd (1)                             1000      944      816      128
```

```
   vmstat (1)                                   1312      1280      808       472
   vold (1)                                     2696      2408     1768       640
   vxconfigd (1)                               14656     13944     1328     12616
----------------------------------------------------------------------------

File (Count)                              Kbytes Resident   Shared  Private
-----------                               ------ --------   ------  -------
/usr/lib Shared Library Totals            291936    27928     2856    25072
Other Shared Library Totals              1473544    29304     7248    22056
Mapped File Totals                           560      488      488        0
Binary File Totals                       1378632    18864    13328     5536
Shared Memory Totals                    80280128  1360688  1360688        0
Anonymous Memory Totals                    89680    84008        0    84008
----------------------------------------------------------------------------

Grand Totals                            83514480  1521280  1384608   136672
```

The bulk of the 1.5 Gbytes of resident memory used on this server is accounted for by 1.3 Gbytes of shared memory, which also constitutes the major component of the memory used for the Oracle processes.

The script can be used to report the physical and virtual memory consumption for a group of processes. For example, `procmem ora` will report memory consumption for all processes that have the string `ora` in a `ps -ef` report (Oracle processes typically meet this criterion). If another Oracle user running the same applications is added to the system, you would not expect an increase in the `Shared` component of memory for `/usr/lib` shared libraries, other shared libraries, binary files, or shared memory segments. The `Private` component would be expected to grow, though, for the shared libraries, the mapped files, and anonymous memory. The additional private memory required would probably be roughly equivalent to the private memory total for these applications divided by the current number of users.

Detail is available for all processes, as well as summaries for `/usr/lib` shared libraries (which tend to be used by many processes throughout a system and so should be counted only once for sizing purposes), other shared libraries (for example, Oracle shared libraries), mapped files (memory-mapped file system files), binary files (executable programs), shared memory segments, and anonymous memory (heap and stack).

The `procmem` script will accurately show all memory directly used by processes, but not memory belonging to UFS files that are resident in the file system page cache. Since pages from UFS database files are not directly mapped into the address spaces of database processes, they will not appear in the totals. The `memps -m` command from `memtool` provides this information (it requires the `memtool` kernel module—installed when the `memtool` package is first set up—to be loaded).

What You Can Do to Reduce Paging

If you are using file systems for your database files, the first step is to upgrade to Solaris 8 or else enable priority paging for earlier releases of Solaris. If necessary, you could also consider include the following steps to relieve memory pressures on your database server:

- Add more memory to the system.
- Use Direct I/O for database files.
- Reduce the size of the database buffer cache. This reduction may result in additional database I/O, but that is almost always preferable to paging.
- Remove applications from the server. If applications are running on the server, move them to a client system and run the applications in client/server mode. Memory should be freed up as a result.
- Reduce the number of users on the system.

STEP 2. Monitoring Disks

If you've made it to Step 2, then you know by now exactly what is going on with memory on your system. The next step is to find out whether you have any disk bottlenecks or disks that may soon become bottlenecks.

Use `iostat`, `statit`, or `sar` to check for a disk bottleneck. The `statit` utility is available on the book website.

Try `iostat -xn 5` (the `-n` option, which displays disk names in the c*nt*-n*d*n format, is only available from Solaris 2.6 on). If you have a lot of disks, you may be so overwhelmed by the output that you find it hard to make sense of all that data. You can use `grep` to remove idle disks from the display (after asking yourself why you have idle disks!):

```
iostat -xn 5 | grep -v "0.0  0.0    0.0    0.0 0.0 0.0    0.0   0   0"
```

Don't try to key this command in—you need exactly the right delimiters between all the `0`s. I simply extract the `grep` string shown above from an `iostat` trace and save the whole command as `iostat2` for future use. Solaris 8 adds a new `-z` option to achieve the same thing (you can simply use `iostat -xnz 5`).

If you want to save a disk activity report for later reference, you will find the `sar` binary file format useful since each data point has an associated timestamp.

What to Look For

The `statit` extract in Figure 3 shows disk behavior for three disks. The first disk is fully utilized, the second is almost idle, and the third is appropriately utilized. The key information is `util%` (the percentage utilization of the disk) and `srv-ms` (service time in milliseconds). Note that service time (the time taken to complete the I/O at the disk) is mislabeled; it is actually response time: the time taken to complete the I/O from the time it leaves the disk device driver on the host, including queuing effects at the controller and the disk. `iostat` also reports the same values (`util%` is shown as `%b`, and `srv-ms` as `svc_t`).

Figure 3   *statit output for three disks*

```
Disk I/O Statistics (per second)
Disk    util%   xfer/s   rds/s  wrts/s rdb/xfr wrb/xfr wtqlen svqlen  srv-ms
c2t4d2  100.0   137.4   133.8     3.7    4081    2048    0.00  12.34   89.8
c2t4d3    0.3     1.0     0.0     1.0       0   14811    0.00   0.02   16.5
c2t4d4   35.5    46.7    42.7     4.1    2048    2048    0.00   0.48   10.3
```

For OLTP workloads, if utilization is consistently greater than about 60% or response time is consistently greater than about 35 msecs, the load on this disk is likely to negatively affect application performance.

For DSS workloads, utilization may exceed 60% and response times may exceed 35 msec—a single 1-Mbyte transfer from a 36-Mbyte disk could take 35 msecs. The wtqlen field in Figure 21.3 (wait in iostat) reports how many other I/O requests are queued and, therefore, how much of the response time is due to queuing time. The svqlen field (actv in iostat) shows the number of requests taken off the queue and actively being processed. With queue lengths consistently greater than 1.0 and response times consistently larger than 35 msecs, disk load is likely to negatively affect application performance.

For both workloads, the key issue is to check how busy the other disks are. You want to avoid the situation where some disks are busy and others are idle. In that respect, the disk utilization and service times shown in Figure 4 reveal a disk layout that is sadly lacking. Disk layout recommendations are discussed in Chapter 17 of *Configuring and Tuning Databases on the Solaris Platform*.

An extract from an iostat trace (iostat -xn 5) is shown in Figure 4 for reference.

Figure 4   *iostat trace*

```
                              extended device statistics
    r/s   w/s    kr/s    kw/s wait actv  svc_t  %w  %b device
   33.2   0.0    66.4     0.0  0.0  0.6   18.4   0   25 c2t4d0
    0.0   2.4     0.0    27.8  0.0  0.0    1.7   0    0 c2t4d1
   60.2   0.0   120.4     0.0  0.0  1.2   19.4   0   48 c2t4d2
   53.8   0.0   107.6     0.0  0.0  0.6   11.8   0   26 c2t4d3
    4.0   0.0   254.3     0.0  0.0  0.1   13.5   0    5 c2t4d4
    0.0   1.2     0.0    19.8  0.0  0.0    2.6   0    0 c3t0d0
    0.0   5.6     0.0    54.4  0.0  0.0    1.9   0    1 c3t0d1
```

The disk utilization shown in this trace is once again unbalanced, suggesting that improvements in the disk layout are needed.

Some utilities, such as sar, for example, report disk names as sd*n*, or ssd*n*, rather than c*n*t*n*d*n*. Having identified a hot disk, you may then find it difficult to locate the disk in question. Thanks to the /etc/path_to_inst file, it is possible to convert the name to a more recognizable form. The procedure is illustrated below.

Suppose a sar trace on host pae280 identifies a hot disk with the name ssd4. First we need to find out more about ssd4.

```
pae280% grep " 4 " /etc/path_to_inst | grep ssd
"/pci@8,600000/SUNW,qlc@4/fp@0,0/ssd@w2100002037e3d688,0" 4 "ssd"
```

The complicated string returned from the /etc/path_to_inst file (the first string surrounded by double quotes) corresponds to the details for the disk in the /devices tree. Entries in the /dev/rdsk directory (and also in /dev/dsk) are actually symlinks to the /devices tree, so we can search for the entry above in the /dev/rdsk directory:

```
pae280% ls -l /dev/rdsk/*s2 | grep \
"/pci@8,600000/SUNW,qlc@4/fp@0,0/ssd@w2100002037e3d688,0"
lrwxrwxrwx   1 root  root   74 Jun 27 18:43 /dev/rdsk/c2t1d0s2 ->
../../devices/pci@8,600000/SUNW,qlc@4/fp@0,0/ssd@w2100002037e3d688,0:c,raw
```

This final step shows that the disk corresponding to ssd4 is /dev/rdsk/c2t1d0s2.

You can use the same procedure by substituting the appropriate details for 4 and ssd in the first step. The string returned can then be substituted for the string beginning /pci in the second step.

Drilling Down Further

Note that recent versions of iostat have the -p option, which shows per partition disk statistics. This option can be helpful in tracking down exactly which database device is responsible for a performance problem.

For systems using Veritas Volume Manager (Veritas), the partition is less useful because Veritas places all its volumes in partition 4. However, Veritas provides the vxstat program to monitor I/O activity per volume. This program is invaluable for drill-downs to find the volumes associated with heavy I/O, especially important when multiple volumes reside on the same disk.

The vxstat utility can be run as follows:

```
vxstat -g group -i interval -c iterations
```

BEWARE: Unlike vmstat and iostat, the statistics reported by vxstat represent totals for the whole interval period, not per second. Consequently, you need to divide the reported bytes read and written by the number of seconds in the interval (to get bytes read and written per second) and also by 1024 (to get kilobytes per second).

What You Can Do to Avoid Bottlenecks

To overcome a disk bottleneck, try one of the following:

- Stripe the data on the disk across a greater number of disks. Take into account, though, the recommendations in "Deciding How Broad to Make a Single Stripe" on page 249 of *Configuring and Tuning Databases on*

the Solaris Platform. Bear in mind, too, that the wider the stripe, the greater the number of disks that will be affected by the failure of a disk within the stripe.

- If there is more than one database volume on the disk, move one or more volumes to other disks.

- Increase the size of the database buffer cache to try to reduce the number of reads to the disk.

- Add more spindles and disk controllers.

An effective disk layout will avoid most disk bottlenecks. If you see uneven disk utilization, revisit the disk layout recommendations in Chapter 17 of *Configuring and Tuning Databases on the Solaris Platform*.

STEP 3. Monitoring Networks

After checking memory and disks for bottlenecks, look next at any networks connected to the server. Although network bottlenecks are not likely to directly affect the performance of the database server, they can have a big impact on application response times.

When database applications are running in client/server mode, a slow network between the client and the server impacts interactions between the database and the applications. When the slow network sits between the applications and the user interface, the user perception could well be that the database server is slow.

What to Look For

A simple way of determining the impact of network latency on response times is to log and plot `ping` round-trip times from the client to the server. The following command pings a host called `adelaide` every 5 seconds and reports the round trip time in milliseconds.

```
alpaca% ping -s -I 5 adelaide
PING adelaide: 56 data bytes
64 bytes from adelaide (129.158.93.100): icmp_seq=0. time=147. ms
64 bytes from adelaide (129.158.93.100): icmp_seq=1. time=150. ms
64 bytes from adelaide (129.158.93.100): icmp_seq=2. time=150. ms
64 bytes from adelaide (129.158.93.100): icmp_seq=3. time=150. ms
^C
----adelaide PING Statistics----
4 packets transmitted, 4 packets received, 0% packet loss
round-trip (ms) min/avg/max = 147/149/150
```

Some application transactions involve multiple trips to the database server, each of which incurs the round-trip penalty. I have seen network latencies account for a significant portion of application response time in wide area networks.

One effective way of quantifying application response times on a wide area network is to enter a dummy transaction with a remote terminal or browser emulator and measure the response time. Dummy transactions can be entered from each remote location at regular intervals. The transaction response times in conjunction with round-trip times captured with `ping` can help determine whether the server or the network has the major impact on performance.

The `netstat` utility shows packet activity on a network; see Figure 5 for an example.

Figure 5   *Network traffic on hme0*

```
alameda% netstat -i -I hme0 5
    input   hme0      output                      input (Total)   output
packets    errs   packets  errs colls    packets   errs   packets  errs   colls
36717966 261164 25214401    12 187668    36718099 261164 25214534  12     187668
       2        0         1   0      0          2        0        1   0         0
```

Watch for collisions (`colls`) greater than 10% of output packets (`output packets`). The use of switches makes collisions less an issue than in the past when many devices shared the same subnet.

Unfortunately, this `netstat` report shows only the number of packets sent and received and not the size of the packets. Without the size of packets it is difficult to assess the effective throughput of the network.

A number of tools are available to provide the number of bytes as well as the number of packets transmitted on a network. The `tcp_mon` script, which is part of the SE toolkit (available on the book website) reports network traffic in both packets and bytes. For the sake of simplicity, divide the theoretical bandwidth by 10 to get the effective throughput in Mbytes. So, a 10-Mbit Ethernet subnet will not be able to exceed approximately 1 Mbyte per second, and a 100-Mbit Ethernet subnet will not be able to exceed 10 Mbytes per second.

The undocumented `netstat` option, `-k`, reports the number of packets received and sent by each network interface (`ipackets` and `opackets`), and as of Solaris 2.6, `netstat -k` also reports the number of bytes received and sent (`rbytes` and `obytes`). The `kstat` utility, introduced in Solaris 8, allows network statistics to be selectively extracted. The following example displays the number of packets and bytes sent and received by all network interfaces on a host called `apollo`.

```
apollo% kstat -p -s "*packets"
hme:0:hme0:ipackets 362997
hme:0:hme0:opackets 480774
hme:1:hme1:ipackets 0
hme:1:hme1:opackets 0
ipdptp:0:ipdptp0:ipackets 124649
ipdptp:0:ipdptp0:opackets 180857
lo:0:lo0:ipackets 45548
lo:0:lo0:opackets 45548
```

```
apollo% kstat -p -s "*bytes"
hme:0:hme0:obytes 394165502
hme:0:hme0:rbytes 47823373
hme:1:hme1:obytes 0
hme:1:hme1:rbytes 0
```

The interfaces in the above example are two 100-Mbit Ethernet interfaces (hme0 and hme1), a dial-up PPP connection (ipdptp), and the loopback interface (lo). The numbers are cumulative; that is, they represent the total since the last reboot. Calculating the average packet sizes (rbytes/ipackets and obytes/opackets) shows that the average packet received on the hme0 interface was 131 bytes in size and the average packet sent was 819 bytes in size.

Although we are focusing on database servers and not NFS file servers, for completeness it is worth mentioning that nfsstat monitors NFS traffic. From a client, use nfsstat -c. Watch for timeouts greater than 5% of calls, or not responding messages when the server was running: they indicate either network problems or an overloaded NFS server.

As of Solaris 2.6, iostat also shows NFS mounts, so all disk statistics available under iostat are also available for NFS mounts.

For a comprehensive treatment of NFS monitoring, refer to Chapter 9 of *Sun Performance and Tuning* by Adrian Cockcroft and Richard Pettit, Second Edition, Sun Press, 1998.

What You Can Do to Minimize Network Bottlenecks

To overcome a network bottleneck, try one of the following:

- Install multiple network adapters and split the traffic across multiple subnets if network traffic becomes an issue. Expanding the network in this way is usually easier in the case of a local area network (LAN) than a wide area network (WAN). Current LAN technology is relatively inexpensive and performs acceptably in most environments. WAN technology is available to satisfy even heavy throughput requirements, although it is still relatively expensive.

- Use Solaris Bandwidth Manager to manage network traffic on servers running mixed workloads.

STEP 4. Monitoring CPUs

Having identified any memory, disk, and network bottlenecks, we are finally ready to look at CPU utilization.

One of the reasons for leaving CPU until last is that there is more to monitor with CPUs. If you start here, you risk getting bogged down in detail and

losing sight of the big picture. But the main reason for monitoring CPU last is that it isn't necessarily bad if your CPUs are heavily utilized.

Why would server CPUs be less than heavily utilized? CPUs on a well-tuned server (one with no memory, disk, or network bottlenecks) will either be idle because there is no work to do or will be busy much of the time.

If there is work to do, you should expect the CPUs to be doing it. If there is work to do and the CPUs aren't busy, it is probably because there is a bottleneck somewhere else, perhaps in the I/O or memory subsystems. Non-CPU bottlenecks should be resolved if at all possible to allow work to proceed uninterrupted.

The aim, then, is to ensure that your workload is CPU-limited rather than limited by memory availability, disk performance, or network performance. Once you have achieved that, optimization remains important, especially application optimization, to ensure that the CPUs are not wasting cycles.

If CPU power were infinite, a server would never be CPU-bound. In the real world, however, significant idle CPU suggests the system has been oversized.

That said, on a multiuser system there are nearly always periods when some users are idle. If CPUs are heavily utilized doing useful work all or most of the time, check user response times and batch job completion times. If response and completion times prove barely acceptable during periods of normal processing load, the server is unlikely to be able to handle peak periods gracefully. On a large multiuser SMP system, a reasonable average CPU utilization is 70%, increasing to 90% during peak periods.

Don't immediately assume that a CPU-limited system is behaving normally, though. To monitor the health of a system with respect to CPU, start by looking at system utilization.

What to Look For: System Utilization

First, use `vmstat` to check how busy the CPUs are. We're not looking for detail initially—the aim at this point is to get the view from 20,000 feet. The relevant statistics to look at are CPU user% (`us`) and system% (`sy`), and the size of the run queue (`r`).

Consider the `vmstat` trace in Figure 6.

Figure 6   *vmstat trace of a lightly loaded system*

procs			memory			page						disk				faults			cpu		
r	b	w	swap	free	re	mf	pi	po	fr	de	sr	m1	m2	s6	sd	in	sy	cs	us	sy	id
0	1	0	3294648	1215024	0	980	1	9	9	0	0	0	0	0	1	1057	13629	5621	35	4	61
0	1	0	3294256	1214664	0	642	0	3	3	0	0	0	0	0	0	746	13709	4285	20	3	78
0	2	0	3292192	1212320	0	473	0	0	0	0	0	0	0	0	0	825	11790	4292	17	3	81

The CPU is only lightly utilized: `id` (CPU idle%) is significantly greater than zero. Not surprisingly, the run queue (`r` under `procs`) is zero, meaning no runnable processes are waiting for CPU time.

By contrast, the vmstat trace in Figure 7 shows a fully utilized system.

Figure 7   *vmstat trace of a fully utilized system*

procs			memory		page						disk				faults			cpu		
r	b	w	swap	free	re	mf	pi	po	fr	de	sr	m1	m2	s6	sd	in	sy	cs	us	sy	id
49	4	0	2984600	916176	0	31	0	0	0	0	0	0	0	0	0	652	5459	2003	84	16	0
31	5	0	2983504	914880	0	22	0	0	0	0	0	0	0	0	0	653	4957	1980	82	18	0

The run queue shows between 30 and 50 runnable processes and 4 or 5 blocked for I/O, and no idle CPU at all. The run queue does not include processes currently executing on the CPUs, only processes waiting for CPU time. A large number of processes blocked for I/O (the b column under procs) can suggest a disk bottleneck.

Is it a problem to have an average of 40 processes waiting on the run queue for a turn on the CPUs? That depends entirely on the number of CPUs in the system: on a 64-CPU system, that situation may not be an issue; on a single CPU server, it is likely to be a major problem.

The *us/sy* (user/system) ratio in Figure 7 is over 4.5/1, which typically indicates a very healthy balance between CPU time spent on user applications and on kernel activity (including I/O). If system% approaches or exceeds user%, a lot of time is being spent processing system calls and interrupts, possibly indicating that excessive time is being spent on disk or network I/O.

What to Look For: Kernel Statistics

The information in Figure 8 is extracted from a statit trace monitoring system activity over a 30-second period (run with statit sleep 30). Most of the disk information has been removed to reduce the size of the output.

Figure 8   *statit trace*

```
  Hostid: 808d5e57 Hostname: "alameda" Version: 6.00 Command: sleep 30
                    Elapsed Time Statistics
     30.03 time (seconds)          100.00 % Start time: Wed Mar 10 16:04:50 2000
      0.00 idle time                 0.00 %
     24.21 user time                80.62 %
      3.70 system time              12.32 %
      2.12 wait time                 7.06 %


  CPU Stats       idle%       user%       system%      wait%  Total%   Total (secs)
  CPU  0          0.0         79.7        13.3          7.0   100.0       30.0
  CPU  1          0.0         81.6        11.4          7.1   100.0       30.0
  Totals          0.0        161.2        24.6         14.1   200.0       60.1


                    Average Load Statistics
       30 secs - monitoring interval       6.20 avg jobs waiting on I/O
      5.00 avg runnable processes          0.87 avg runque occupancy
      0.00 avg swapped jobs                0.00 avg swap device occupancy
```

```
                    Average Swap Statistics in Pages
  111814.16 avg freemem                103848.90 avg reserved swap
   96036.93 avg allocated swap         370944.53 avg unreserved swap
  378756.50 avg unallocated swap
                    Sysinfo Statistics (per second)
       0.00 phys block reads               4.96 phys block writes
(sync+async)
     160.54 logical block reads           29.10 logical block writes
     189.94 raw I/O reads                  0.00 raw I/O writes
    3264.90 context switches             958.11 traps
    1620.41 device interrupts            9813.75 system calls
    2202.96 read+readv syscalls         1501.63 write+writev syscalls
 6066097.10 rdwr bytes read            254117.65 rdwr bytes written
       0.43 forks + vforks                 0.53 execs
       0.00 msgrcv()+msgsnd() calls       128.07 semop() calls
      16.68 pathname lookups (namei)        1.20 ufs_iget() calls
       0.00 inodes taken w/ attach pgs      0.00 inodes taken w/ no attach pgs
       5.59 directory blocks read           0.00 inode table overflows
       0.00 file table overflows            0.00 proc table overflows
        100 % bread hits                  687.41 intrs as threads(below clock)
       3.73 intrs blkd/released (swtch)   143.62 times idle() ran (swtch)
       0.00 rw reader fails (swtch)         0.00 rw writer fails (swtch)
    1166.60 involuntary context switches  687.85 xcalls to other cpus
       0.67 thread_create()s              221.65 cpu migrations by threads
     179.35 failed mutex enters            0.00 times module loaded
       0.00 times module unloaded          4.23 physical block writes (async)
                    Vminfo Statistics (per second)
       0.13 page reclaims (w/ pageout)      0.13 page reclaims from free list
       0.00 pageins                         0.00 pages paged in
       0.13 pageouts                        0.17 pages paged out
       0.00 swapins                         0.00 pages swapped in
       0.00 swapouts                        0.00 pages swapped out
      39.96 ZFOD pages                      0.17 pgs freed by daemon/auto
       0.00 pgs xmnd by pgout daemon        0.00 revs of page daemon hand
       0.00 minor pgflts: hat_fault       101.70 minor pgflts: as_fault
       0.00 major page faults             12.89 copy-on-write faults
      21.84 protection faults              0.00 faults due to s/w locking req
       0.43 kernel as as_flt()s            0.00 times pager scheduled
                    Directory Name Cache Statistics (per second)
      87.28 cache hits  ( 96 %)            3.23 cache misses (  3 %)
       0.23 enters into cache              0.00 enters when already cached
       0.00 long names tried to enter      0.00 long names tried to look up
       0.00 LRU list empty                 0.00 purges of cache
                    Segment Map Operations (per second)
      12.22 number of segmap_faults        0.00 number of segmap_faultas
    1334.90 number of segmap_getmaps       8.39 getmaps that reuse a map
    1324.48 getmaps that reclaim           1.80 getmaps reusing a slot
       0.00 releases that are async        0.00 releases that write
       2.63 releases that free             3.03 releases that abort
       0.00 releases with dontneed set     0.00 releases with no other action
       2.63 # of pagecreates
```

```
                   Buffer Cache Statistics (per second)
   160.64 total buf requests                160.64 buf cache hits
     0.00 times buf was alloced               0.00 times had to sleep for buf
     0.13 times buf locked by someone         0.00 times dup buf found
                   Inode Cache Statistics (per second)
     1.20 hits                                0.00 misses
     0.00 mallocs                             0.00 frees
     0.00 puts_at_frontlist                   0.00 puts_at_backlist
     0.00 dnlc_looks                          0.00 dnlc_purges
                    Char I/O Statistics (per second)
   187.48 terminal input chars            20398.83 terminal output chars
 Network Statistics (per second)
Net      Ipkts    Ierrs    Opkts    Oerrs    Colls    Dfrs   Rtryerr
hme0       404        0      248        0        1        0        0
lo0          0        0        0        0        0        0        0
                    Disk I/O Statistics (per second)
Disk     util%   xfer/s   rds/s    wrts/s  rdb/xfr wrb/xfr wtqlen  svqlen srv-ms
md10      0.0      0.0      0.0      0.0        0        0    0.00    0.00    0.0
md20      0.0      0.0      0.0      0.0        0        0    0.00    0.00    0.0
c6t6d0    0.0      0.0      0.0      0.0        0        0    0.00    0.00    0.0
c2t0d0   97.1    125.6    124.6      1.1     3822     2048    0.00    5.10   40.6
c2t0d1    0.3      0.6      0.0      0.6        0    18312    0.00    0.01   13.2
c2t0d2    0.0      0.0      0.0      0.0        0        0    0.00    0.00    0.0
c2t0d3   46.8     67.8     63.2      4.6     2048     2048    0.00    0.66    9.8
c2t0d4    0.0      0.0      0.0      0.0        0        0    0.00    0.00    0.0
c2t1d0    8.6     83.5      2.2     81.3    56763     2048    0.00    0.76    9.1
c2t1d1    0.0      0.0      0.0      0.0        0        0    0.00    0.00    0.0
```

statit shows a lot of information, including CPU, memory paging, and network and disk statistics. Part of the attraction of statit is the comprehensiveness of the information it provides. Let's look at a few highlights.

- When looking at CPU utilization, don't be confused by I/O wait time. I/O wait time is highly misleading and should be regarded simply as idle. So add wait time and idle time together to determine the true idle time. Do the same for sar also (add wio and idl to determine idle time).

- Check context switches and involuntary context switches. A *context switch* occurs when a process or thread is moved onto or off a CPU. An *involuntary context switch* occurs when a running process or thread has consumed its allocated time quantum or when it is preempted by a thread with a higher priority. If the ratio of context/involuntary is significantly less than about 3/1, it can indicate that processes are being preempted before they have completed processing (usually processes will yield—that is, give up—the CPU when they request an I/O). A high level of involuntary context switching suggests there might be a benefit from using a modified TS dispatch table if your server is not a Starfire server (refer to "The TS Class" on page 220 of *Configuring and Tuning Databases on the Solaris Platform* for more information).

- Semaphore operations (`semop()` calls) and message queue calls (`msgrcv()`+`msgsnd()` calls) are the typical mechanisms used by databases for Interprocess Communication (IPC) and indicate the degree of synchronization traffic between database processes (usually primarily for internal locks and latches).

  Semaphore operations can increase exponentially when a database server becomes significantly overloaded. Such behavior is a symptom rather than a cause of poor performance, but it is a good indication that the CPU is unable to effectively complete the work it is doing and that more CPU resource is required.

- For the sake of reference, `pageouts` and `pgs xmnd` by `pgout daemon` are equivalent to `po` and `sr`, respectively, in a `vmstat` trace.

- A high level of `faults due to s/w locking reqs` can suggest that ISM is not being used when shared memory is attached (ISM is described in "Intimate Shared Memory" on page 24 of *Configuring and Tuning Databases on the Solaris Platform*). Oracle and Sybase, for example, will try to attach shared memory as ISM, but if unsuccessful will attach shared memory without ISM. In each case an advisory message is placed in the database log file, but the onus is on you to notice it. A method of determining whether ISM is being used is discussed in "EXTRA STEP: Checking for ISM" on page 24.

Drilling Down Further

Monitoring Processes

Sometimes individual processes hog CPU resource, causing poor performance for other users. Use `/usr/ucb/ps -aux`, or `prstat` as of Solaris 8, to find out the processes consuming the most CPU (note that the `ps` process is itself a reasonably heavy consumer of CPU cycles, especially on systems with many processes). The `sdtprocess` utility (shipped as part of the CDE package within Solaris) offers a useful X11-based representation of the same data.

Figure 9 shows a trace where no particular process is hogging CPU.

Figure 9   *ps trace of multiple Oracle shadow processes*

```
alameda% /usr/ucb/ps -aux | head -10
USER       PID %CPU %MEM   SZ  RSS TT       S    START  TIME COMMAND
oracle   12545  0.4  2.57955276608 ?        R 17:37:52  0:09 oraclegl P:4096,4,
oracle   13200  0.3  2.57804075000 ?        S 17:40:11  0:04 oraclegl P:4096,4,
oracle   13250  0.3  2.57804074960 ?        R 17:40:21  0:04 oraclegl P:4096,4,
oracle   12236  0.3  2.57804074960 ?        S 17:36:47  0:10 oraclegl P:4096,4,
oracle   13102  0.3  2.57808075000 ?        R 17:39:51  0:04 oraclegl P:4096,4,
oracle   12598  0.3  2.57804074960 ?        S 17:38:02  0:09 oraclegl P:4096,4,
oracle   12323  0.3  2.57804074960 ?        R 17:37:04  0:11 oraclegl P:4096,4,
oracle   12263  0.2  2.57804074960 ?        R 17:36:51  0:11 oraclegl P:4096,4,
oracle   13615  0.2  2.57833675400 ?        R 17:41:30  0:02 oraclegl P:4096,4,
```

In this example, a lot of processes are running, but none are taking more than 0.4% of all available CPU.

In Figure 10 a couple of processes are consuming many times more CPU than other processes.

Figure 10   *ps trace of CPU hogging processes*

```
alameda% /usr/ucb/ps -aux | head -5
USER       PID %CPU %MEM   SZ  RSS TT        S    START  TIME COMMAND
root     13428 12.5  0.0  808  400 pts/2     O 17:41:00  5:38 /tmp/badboy2
root     13422 12.4  0.0  808  400 pts/2     O 17:40:58  5:40 /tmp/badboy1
oracle   14850  0.6 2.57804874952 ?          R 17:46:19  0:02 oraclegl P:4096,4,
oracle   14838  0.5 2.57808075040 ?          R 17:46:10  0:00 oraclegl P:4096,4,
```

The TIME column also shows that the CPU hogs have each consumed 5 minutes of CPU time. How much performance impact they will have depends on the number of online CPUs and other active processes. The %CPU column shows the percentage of all available CPUs, not the percentage of a single CPU. In this example, the two rogue processes are each consuming one full CPU out of eight (hence 12.5 %CPU). You can use pstack and truss to get an indication of what these CPU-hogging processes are doing.

Figure 11 illustrates a method of finding the process consuming the most CPU (with ps), then listing the system calls the process is running (with truss). I stopped truss after about 10 seconds with Control-C; at that point the system call stats were printed.

Figure 11   *truss system call trace of single process*

```
alameda./opt/bin /usr/ucb/ps -aux | head -2
USER       PID %CPU %MEM   SZ  RSS TT        S    START  TIME COMMAND
oracle   14870  1.2 2.57849675552 ?          S 17:48:24  0:06 oraclegl P:4096,4,
alameda./opt/bin truss -c -p 14870
^Csyscall       seconds    calls  errors
read               .06      239
lseek              .02      197
semsys             .00        5
context            .00        3
setitimer          .00        6
                  ----      ---    ---
sys totals:        .08      450      0
usr time:         1.18
elapsed:         19.45
```

Read system calls dominated, with lseek close behind. The use of lseek indicates that the application is not using the pread(2) system call, which saves a system call by eliminating the need for lseek(2).

Monitoring Interrupts

The trace in Figure 12 is from mpstat on an 8-CPU server.

Figure 12   *mpstat trace for an 8-CPU server*

CPU	minf	mjf	xcal	intr	ithr	csw	icsw	migr	smtx	srw	syscl	usr	sys	wt	idl
6	84	0	87	3	1	221	1	10	8	0	403	6	3	91	0
7	24	0	98	4	1	256	2	14	9	0	261	8	2	90	0
10	76	0	81	4	0	149	3	8	138	0	735	44	3	53	0
11	99	0	44	237	228	251	2	14	8	0	722	6	3	91	0
14	28	0	253	120	118	261	2	11	139	0	1139	7	4	90	0
15	115	0	30	2	0	174	2	10	7	0	353	26	3	72	0
18	89	0	74	204	3	245	2	13	10	0	853	6	4	89	0
19	24	0	119	5	0	303	4	11	8	0	374	13	3	85	0

Note that interrupts (`intr`) are not evenly spread across all CPUs. CPUs 11, 14, and 18 are processing more interrupts than the other CPUs. These CPUs show the highest system (`sys`) activity, but not the highest user (`usr`) activity.

Disk array and network drivers are bound to specific CPUs when Solaris boots. These CPUs handle interrupts related to these devices on behalf of all other CPUs. Notice, too, that Solaris has scheduled the running processes on the CPUs that are not busy servicing interrupts (CPUs 10, 1, and 19 show significantly greater user activity).

What You Can Do to Optimize CPU Usage

Following is a list of some causes of heavy CPU utilization, along with ways to track down the causes, and remedies to apply.

- **An unusually high level of system calls.** If the ratio of user% to system% is low, try to find out the causes by using `truss -c` to identify the main system calls for a few of the most CPU-intensive processes. If read I/Os are a major factor, increasing the size of the database buffer cache might reduce the number of read I/Os.

- **A low ratio of context switches to involuntary context switches.** On non-Starfire platforms, load the Starfire TimeShare Dispatch Table (see Chapter 15 of *Configuring and Tuning Databases on the Solaris Platform*).

- **One or more inefficient applications.** Monitor processes with `ps` to identify the applications consuming the most CPU. Poorly written applications can have a major impact on CPU requirements and therefore offer one of the most fruitful places to begin looking when you are trying to free up CPU resources.

- **Poor database tuning.** The next chapters of *Configuring and Tuning Databases on the Solaris Platform* might help you identify inefficient database behavior. A high level of latch contention, for example, can cause CPU to be consumed for little benefit. If latch contention is the cause of heavy CPU utilization, adding more CPUs may not always help.

- **Insufficient CPU resources.** If your CPUs are consistently fully or heavily utilized and there are many more processes on the run queue than there are CPUs, you may simply need to add more CPUs. Fortunately, Solaris scales well enough that more adding CPUs is likely to help when CPU resources are scarce.

  An alternative might be to remove applications from the database server onto another server to use the database's client/server capabilities. Removing applications can make a big difference to CPU utilization on the database server.

STEP 5. Monitoring and Tuning a Database

A well-tuned database system has the following characteristics:

- The system is mostly CPU-bound.
- Disk I/O is well-balanced.
- The buffer cache is working effectively.
- The database is configured to run efficiently.

As we have already noted, the applications must also be coded efficiently if the database system is to run efficiently.

Having investigated memory, disks, networks and CPUs, you need to undertake the final step: monitor and tune the database, a process explored in detail in the following chapters of *Configuring and Tuning Databases on the Solaris Platform* for Oracle, Sybase, Informix, and DB2. Monitoring and tuning the buffer cache is a vital element of database tuning. A brief review of the main issues is presented in the next section, but before proceeding to the database tuning chapters, you would do well to review Chapter 7, which is dedicated to the buffer cache. Finally, although application efficiency is a crucial component in any well-tuned system, we will not be considering it since it is beyond the scope of this document. Refer to *Techniques for Optimizing Applications: High Performance Computing*, by Rajat Garg and Ilya Sharapov, Sun Microsystems Press, 2001 for a discussion on application optimization.

The Buffer Cache

One of the main metrics in monitoring database performance for OLTP workloads is the buffer cache hit rate. The buffer cache stores in memory as many database blocks read from disk as possible. The expectation is that the same database blocks will often be used by different transactions; the *buffer cache hit rate* shows how often a requested block was retrieved from the cache rather than from the disk. Since memory access is so much faster than disk

access, a high cache hit rate is important for good performance with OLTP workloads.

For DSS workloads the buffer cache is less important. Some databases (for example, Oracle and Informix) avoid the buffer cache entirely when carrying out table or index scans. DB2 for Solaris does use the buffer cache, but there may be little opportunity for reuse of buffers since the volume of data brought into the cache is typically much greater than the cache size. So, cache buffers tend not to stay in the buffer for long. Similar behavior applies to batch jobs processing large volumes of data.

A Closer Look at the Cache Hit Rate

The cache hit rate can be misleading. For example, how much better is a 95% cache hit rate than a 90% cache hit rate? Does it represent a 5% improvement?

The answer is no! A 90% cache hit rate means 90% of all read I/Os are satisfied from the cache—only 10% of the reads result in a physical disk access. A 95% cache hit rate means only 5% of reads go to disk. So, improving the cache hit rate from 90% to 95% means physical disk reads are *halved*, not improved by only 5%! Depending on the number of physical reads, halving them may make a significant difference to the load on the disks.

Given this potential confusion, it is often more useful to think in terms of the *miss rate* (100 minus cache hit rate).

You can usually improve the cache hit rate by increasing the size of the database buffer cache, although a point is eventually reached when any improvement is not worth the extra memory. Cache sizing and effectiveness are discussed in more detail in Chapter 7 of *Configuring and Tuning Databases on the Solaris Platform*.

An Appropriate Cache Hit Rate

A cache hit rate of 80% might be quite suitable in one situation while a much higher hit rate of 95% might be inadequate in another. How can you tell when you need to increase the size of the buffer cache to try to improve the cache hit rate?

You should consider three factors when determining an appropriate cache hit rate:

- **The amount of memory available.** Avoid paging at all costs. If increasing the size of the buffer cache causes application paging, then it is better to leave the cache as it is. If free memory is available, then increasing the cache is an option.

- **The load on the database disks.** Determine the disk utilization of the database disks by looking at either iostat, sar, or statit statistics for the relevant disks. Check, too, how many of the I/Os are due to reads. If the database disks are showing high utilization or high service times

(as defined in "STEP 2. Monitoring Disks" on page 9) and a significant number of the I/Os are reads, then increasing the cache hit rate could take the pressure off the disks.

- **Application response times.** Retrieving disk blocks from the cache is faster than retrieving them from disk and also consumes less CPU. So improving cache hit rate might also reduce response times. Many factors contribute to response times, though, so any improvement may be less than you were hoping for.

Aftereffects of Database Monitoring and Tuning

Given that the database may be the major application on the system, investigating the behavior of the database should shed further light on the data already collected. Monitoring and tuning the database is an iterative process. Tuning the database may change the behavior of the system, in which case it will be important to briefly revisit the previous steps covered in the chapter.

EXTRA STEP: Checking for ISM

Unfortunately, before Solaris 7 it isn't easy to tell whether a shared memory segment is using ISM (described in "Intimate Shared Memory" on page 24 of *Configuring and Tuning Databases on the Solaris Platform*). As of Solaris 7, the ipcs utility includes a -i parameter that shows how many shared memory segments have been attached as ISM (the ISMATTCH column). Since Solaris 8, pmap shows whether a particular process has attached a shared memory segment as ISM (ism will appear immediately before the shmid parameter). For earlier Solaris releases, the procedures shown in Figure 13 through Figure 17 can be used to answer the question.

Figure 13 is based on a server running an Oracle database.

Figure 13   *ps trace of Oracle shadow process*

```
    UID   PID  PPID  C    STIME TTY      TIME CMD
 oracle  4498  4479  0 15:58:20 ?        0:02 oraclegl P:4096,3,8,
```

The ps -aef command shows a number of Oracle processes, including pid 4498.

In Figure 14, the pstack command shows that pid 4498 is currently doing a read.

Figure 14   *pstack trace for process*

```
alameda# /usr/proc/bin/pstack 4498
4498:   oraclegl P:4096,3,8,
 ef7385e8 read     (3, 681150, 1000)
 ef7385e8 _libc_read (3, 681150, 1000, 0, ef7a227c, 5b4fc) + 8
 0005b4fc osnprc  (ffffffff, 6800d8, 681150, 681150, 3, 0) + 4a8
 000d91e4 opitsk  (67df80, 67dfcc, 67dfc8, 0, a, 6801d8) + 334
 001396c4 opiino  (0, 67e000, 64, 67c7f8, 0, 67dd7c) + 53c
 000daf44 opiodr  (0, 1, 1, 0, 0, 67df84) + f64
 000cb43c opidrv  (67c7f8, 0, 0, 3c, 0, 67c7f8) + 56c
 000ca020 sou2o   (effff844, 3c, 4, effff834, 0, 0) + 10
 00058e70 main    (2, 0, effff8d0, 67f800, 0, 0) + 8c
 00058dcc _start  (0, 0, 0, 0, 0, 0) + 5c
```

In Figure 15, the pmap command shows that pid 4498 has attached to a shared memory segment shmid=0x5401, size 69144K bytes, base address 0xD0000000.

Figure 15   *pmap trace for process*

```
alameda# /usr/proc/bin/pmap 4498
4498:   oraclegl P:4096,3,8,
00010000   6504K read/exec          /FIN/ora713fin/oracle7.1.3/product/
  7.1.3/bin/oracle.ORIG.before_debug.121694
00678000     32K read/write/exec    /FIN/ora713fin/oracle7.1.3/product/
  7.1.3/bin/oracle.ORIG.before_debug.121694
00680000    408K  read/write/exec      [ heap ]
D0000000  69144K  read/write/exec/shared  [ shmid=0x5401 ]
EF5A0000     16K  read/exec          /usr/platform/sun4u/lib/libc_psr.so.1
EF5B0000      8K  read/write/exec      [ anon ]
EF5C0000     16K  read/exec          /usr/lib/libmp.so.2
EF5D2000      8K  read/write/exec   /usr/lib/libmp.so.2
EF5E0000     24K  read/exec          /usr/lib/libaio.so.1
EF5F4000      8K  read/write/exec   /usr/lib/libaio.so.1
EF5F6000      8K  read/write/exec      [ anon ]
EF600000    448K  read/exec          /usr/lib/libnsl.so.1
EF67E000     32K  read/write/exec   /usr/lib/libnsl.so.1
EF686000     24K  read/write/exec      [ anon ]
EF6B0000     88K  read/exec          /usr/lib/libm.so.1
EF6D4000      8K  read/write/exec   /usr/lib/libm.so.1
EF6E0000     32K  read/exec          /usr/lib/libsocket.so.1
EF6F6000      8K  read/write/exec   /usr/lib/libsocket.so.1
EF6F8000      8K  read/write/exec      [ anon ]
EF700000    592K  read/exec          /usr/lib/libc.so.1
EF7A2000     24K  read/write/exec   /usr/lib/libc.so.1
EF7A8000      8K  read/write/exec      [ anon ]
EF7B0000      8K  read/exec/shared /usr/lib/libdl.so.1
EF7C0000    112K  read/exec          /usr/lib/ld.so.1
EF7EA000     16K  read/write/exec   /usr/lib/ld.so.1
EFFF8000     32K  read/write/exec      [ stack ]
 total    77616K
```

Now, use the crash program to discover more about this process. In Figure 16, the data from the other processes on the system has been removed to reduce the size of the following trace.

Figure 16   *crash program proc output*

```
alameda# crash << EOF
proc
EOF
dumpfile = /dev/mem, namelist = /dev/ksyms, outfile = stdout
proc
> PROC TABLE SIZE = 16394
SLOT ST  PID  PPID  PGID   SID   UID  PRI      NAME         FLAGS
1705 s  4498  4479  4498  4498  2256  58    oracle.ORIG.befo   load
```

Next, use the SLOT information to find out whether this process is using ISM. In Figure 17, the output from crash has been reduced to save space.

Figure 17   *crash program as output*

```
alameda# echo "as -f 1705" | crash
dumpfile = /dev/mem, namelist = /dev/ksyms, outfile = stdout
>
PROC         PAGLCK   CLGAP  VBITS HAT       HRM         RSS
 SEGLST      LOCK            SEGS     SIZE      LREP     TAIL        NSEGS
1705         0               0       0x0       0x66104370  0x0
0x63a592a0  0xeffff318  0x67de6ce0  79978496  0           0x65ccf5c0   26
  BASE         SIZE     AS        NEXT        PREV        OPS        DATA
   0x00010000  65a000 0x680d5f40 0x68b4f260 0x00000000  segvn_ops  0x63aca2f0
   0x00678000    8000 0x680d5f40 0x6547d9a0 0x67de6ce0  segvn_ops  0x6440cbb0
   0x00680000   66000 0x680d5f40 0x63a592a0 0x68b4f260  segvn_ops  0x64422fe0
   0xd0000000 4400000 0x680d5f40 0x67263520 0x6547d9a0  segspt_shm 0x6721a060
   0xef5a0000    4000 0x680d5f40 0x63905ac0 0x63a592a0  segvn_ops  0x67fcab38
   0xef5b0000    2000 0x680d5f40 0x65ccee40 0x67263520  segvn_ops  0x64898b00
```

We saw before that the shared memory segment had a base address of 0xD0000000 (hex). The equivalent line on this trace shows a size of 4400000 (hex), which is equivalent to 71,303,168 bytes, or 69,632 Kbytes. This size agrees with that shown by pmap (rounded up to the nearest megabyte). The OPS value shows segspt_shm. This value definitely shows that the ISM driver is in use for this shared memory segment. If the value is segvn_ops, then ISM is not in use.