

## 第22章 鉴别资源密集型用户

本章要点：

不同的资源

CPU

文件输入/输出

内存

每一个数据库管理员都面对过这样一种情形，即数据库开始拖延并由于某种原因速度突然变慢。这种现象能够发生在一个已被很好调整过的数据库中，它会在不知不觉中影响你。这时数据库管理员需要用技巧来辨认资源密集型用户并做一些必要的影响数据库性能的事情。我曾多次查明用光全部系统资源并引起数据库整体性能下降的会话。资源密集型用户可能是一个执行大型查询的用户，其中该查询拥有很多表并在 SELECT子句中遗漏了几个键列连结，或者它可以是一个未优化的语句。本章的目的就是要使数据库管理员具备必要的技巧以便相对容易地鉴别这样的用户。本章提供鉴别资源密集型用户的方法和技术，但没有详细地讨论纠正问题的细节。关于如何解决它们的细节将在本书中其他有关性能调整的章节中讨论。

### 22.1 不同的资源

你可以采用许多方法鉴别资源密集型用户。下面这个方法首先对环境进行了概述。假如觉察到出错的地方，那么应进一步地在该区域中调查以便对用户进行详细描述。你要查看的资源是CPU、文件输入/输出和内存。假如CPU被严重加载并且完全被某个执行大量的CPU密集型工作的用户占用的话，那么该用户会降低系统速度。类似，频繁地文件输入/输出也能够引起系统性能下降。某些进程对内存的频繁请求会导致内存分页与交换，这将严重阻碍系统性能。你应把注意力集中在这三种资源上。基本上，在服务器上执行的每一个进程试图获取如下资源：

**CPU**——每一个进程都需要一些CPU时间片以执行它的工作。有些进程夺取了大量CPU时间，有些进程完成时间较早。可以使用的有限CPU能力必须在系统上的Oracle进程与内部操作系统（OS）进程之间共享。显而易见，CPU密集型用户是指消耗大量CPU时间的用户。

**文件输入/输出**——每当一个进程需要存取数据时，如果该数据已经由以前的某个进程带入的话，那么该进程就首先查找缓存。如果数据已经存在于该缓存中，则读取很快便完成。如果需要的数据不在缓存中，那么就必须从磁盘上读取该数据。这时必须执行一个物理读取。这种从磁盘中读取数据的操作非常耗时，因为从磁盘中读数据所用的时间几乎是从高速缓存中读数据所用时间的50倍。当系统开始密集地进行输入/输出时，系统性能开始下降，因为所有进程在大部分时间里所做的工作就是等待将从磁盘中返回的数据。

我正在讨论的资源是时间。由于较繁忙的磁盘文件存取而导致大量的时间被消耗，所以

时间显得越来越珍贵。最终，所有资源的使用将归结为一个资源——时间。

内存——由于某些进程对内存的大量使用而导致内存不足也可能造成性能下降。在UNIX操作系统中，利用虚拟内存的原理可以有效地使用内存。在一个虚拟内存系统中，仅仅最常使用的页面才被保留在物理内存中；其余的或当前非活动页面被保存在交换空间中，在需要时才被引入物理内存中。当由于物理内存较少或进程对内存的频繁使用而使得系统中的内存短缺时，为满足对内存的需要，系统首先开始分页。假如内存仍然短缺，系统便求助于交换物理进程，实际上这样做会降低系统性能。分页与交换将在稍后的22.4节中讨论。

当大量使用这些资源时，系统会经历性能的下降。因此，给每一个用户分配多少资源非常重要。要鉴别严重资源消耗，你应每次把注意力集中在一种资源上并设法鉴别由进程消耗的个别资源。

## Windows NT性能监控

用于Windows NT的Oracle 8i包含如下三种性能监控工具：

Oracle 8数据字典视图。

Windows NT性能监控程序。

Oracle 8性能监控程序。

### 1. Oracle 8数据字典视图

有多组含有大量的有关数据库性能信息的数据字典视图。有两个常见的数据字典视图类别：静态（static）与动态（dynamic）。静态视图包含有关数据库整体结构的信息，例如名称、数据库文件的位置、用户信息、存储器使用率、重做日志和控制文件信息等。动态视图（也称为V\$表）反映了诸如物理输入/输出、内存使用率、用户活动等实时性能信息。

### 2. Windows NT性能监控程序

Windows NT可以调整的参数不多。例如，在UNIX操作系统中，系统管理员可以修改大量的直接影响系统性能的核心参数。相反，Windows NT根本不能接触到核心参数。Windows NT被认为是自调整的，因为它包含了许多自适应算法，这些算法在系统运行时可以动态地调整性能。

Windows NT操作系统的大部分性能信息可以在Windows NT性能监控程序（PerfMon）中找到。Windows NT性能监控程序是一个灵活的图形工具，你可以使用它来收集有关Windows NT的实时性能信息。它也为被选取的每一个计数器产生一个数值。这些值显示在图的低端。你可以使用该信息确定系统中正经历性能瓶颈的部分并采取相应的步骤消除这些问题。

Windows NT几乎把整个操作系统中的每一个事物作为一个对象对待。

### 3. Oracle 8性能监控程序

Oracle 8性能监控程序是一系列定制的Windows NT性能监控程序计数器，它们能够用来测量Oracle数据库的性能。此性能监控工具采用数据字典视图的信息并以图形方式显示该信息。计数器全部基于依靠数据字典发出的SQL语句。你可以在PERF 81.ORA文件中看到由Oracle 8 PerfMon发出的SQL语句，PERF 81.ORA文件位于ORANT\DBS目录中。当你单击Oracle 8性能监控程序图标时便自动建立一个到数据库的连接。用于此连接的用户名、口令和其他信息位于Windows NT注册表中。要查看此信息，运行注册表编辑程序（REGEDT32.EXE）。打开HKEY\_LOCAL\_MACHINE键，然后定位到SYSTEM\CurrentControlSet\Service\Oracle80\Performance。

通过双击用户名、口令或主机名来相应地改变它们的值（参见图 22-1）。缺省设置是 system/manager@orcl。



图22-1 用于Oracle 8性能监控工具的注册项

用于Oracle 8的性能监控程序计数器显示于表 22-1中。

表22-1 性能监控程序计数器

| 对 象                   | 计 数 器                                  |
|-----------------------|--|
| Oracle 8缓存            | %物理读/获取的次数                             |
| Oracle 8字典缓存          | %遗漏/获取的次数                              |
| Oracle 8数据文件          | 物理读/秒的次数；物理写/秒的次数<br>(每一个实例是一个单独的数据文件) |
| Oracle 8 DBWR stats 1 | 已搜索的缓冲区/秒；LRU搜索/秒                      |
| Oracle 8 DBWR stats 2 | 检查点/秒；停工时间/秒                           |
| Oracle 8动态空间管理        | 循环调用次数/每秒                              |
| Oracle 8自由表           | %自由表等待次数/请求                            |
| Oracle 8库缓存           | %重新加载/针                                |
| Oracle 8重做日志缓冲区       | 重做日志空间请求                               |
| Oracle 8排序            | 内存中的排序数/秒；磁盘上的排序数/秒                    |

## 22.2 CPU

UNIX系统是一个多进程操作系统；也就是说，UNIX操作系统能够同时管理多个进程。每一个必须执行一个作业的进程在运行队列中等待得到CPU时间片来执行它的作业。当轮到某一个进程时，该进程便被分配一个时间片，它在该时间片中执行它的作业。一个经过良好调整的CPU在执行进程的运行队列中不会有很多的进程等待，但是它应充分忙碌。22.2.2节将鉴别那些消耗过多CPU资源的进程。当做此练习时，记住，在一个较轻负载的系统中，一个进程可以独占整个CPU，这很普通并且不会成为惊慌的理由。当你试图鉴别CPU密集型用户时，你务必保证在任何时间点上系统中有足够数量的作业正在运行并且存在性能问题。仅在

这种情况下你才可以采用这里提到的方法，因为在低负载的系统中，少量的用户消耗了大量的CPU时间,这确实很普通。

22.2.1 CPU预览

在你鉴别正引起CPU资源问题的用户以前，你必须大致了解一下CPU是如何运转的。在UNIX系统中，你可以使用sar -u命令确定CPU的工作情况。sar -u命令有4列输出，见表22-2。

表22-2 sar -u命令输出中的重要列

| 列名   | 描 述   |
|------|---|
| usr  | 花费在为 用户请求服务上的CPU时间的百分数                                      |
| sys  | CPU参与系统调用所花费的时间百分数  |
| wio  | CPU等待从磁盘中输入/输出的完成所花费的时间百分数。假如该数值较高的话，表明磁盘中存在瓶颈或输入/输出系统的效率低下 |
| Idle | 空闲CPU时间的百分数   |

一个正常的或经过良好调整的CPU通常使用户CPU时间是系统CPU时间的两倍；也就是说，同服务于系统调用相比，CPU花费更多时间服务于用户请求。

sar命令的示例输出稍后显示。基本的命令格式如下：

```
sar -u n t
```

n是执行监控的时间间隔，t是执行监控的次数。下面是一条sar CPU Utilization命令的输出示例：

```
$ sar -uM 2 10
HP-UX arms1 B.10.10 U 9000/819 09/04/97
19:53:57      cpu      %usr      %sys      %wio      %idle
19:53:59        0        38         9        40        12
              1        43        10        37        10
        system    40        10        39        11
19:54:01        0        51        14        29         6
              1        45        17        33         6
        system    48        15        31         6
19:54:03        0        42         8        44         6
              1        44         6        42         8
        system    43         7        43         7
19:54:05        0        52        10        34         4
              1        34        17        42         6
        system    43        14        38         6
19:54:07        0        30        14        45        11
              1        37        10        46         7
        system    34        12        46         9
19:54:09        0        44        10        38         8
              1        44         6        43         6
```

假如CPU含有大量的空闲时间，这可能意味着CPU没有被最充分地利用。假如%wio、%usr和%sys列的数值较低且%idle列的数值较高，这就意味着当前在系统中没有东西正在运行。

假如CPU花费的系统时间高于用户时间，这也需要进行检查。你旨在使用户CPU时间成倍于或超过系统CPU时间；否则，你需要进一步的检查。在前面的输出中，%wio列具有较高的数值；这意味着在当前的方案中，CPU没有出现问题，但是在磁盘上可能存在瓶颈。你需要进一步执行sar -d命令以便发现哪一张磁盘正面临输入/输出负载。

正如较早前所提及的那样，每一个必须在系统中执行的进程要在运行队列中等待。假如有太多的作业在运行队列中等待，这表明系统中有大量的资源密集型作业正在运行，此时CPU不能处理系统的请求。下面是运行队列的一个评定示例，同时也是一个查明CPU是否由于系统中运行的进程而停顿的好起点：

```
$ sar -qu 2 10
HP-UX arms1 B.10.10 U 9000/819    09/06/97
17:35:37 runq-sz %runocc swpq-sz %swpocc
          %usr   %sys   %wio   %idle
17:35:39      1.0    25     0.0     0
          39     11     43     6
17:35:41      0.0     0     0.0     0
          46    10     40     5
17:35:43      1.0    25     0.0     0
          17     5     55    24
17:35:45      1.0    25     0.0     0
          38     9     42    10
17:35:47      1.0    25     0.0     0
          39     9     45     8
17:35:49      0.0     0     0.0     0
          36     6     52     6
17:35:51      1.0    25     0.0     0
          22    10     51    16
17:35:53      1.0    25     0.0     0
          44     7     46     3
17:35:55      2.0    25     0.0     0
          41     8     45     6
17:35:57      0.0     0     0.0     0
          25     8     50    17

Average      1.1    18     0.0     0
Average      35     8     47    10
```

表22-3解释输出中的列的含义。

表22-3 对sar -qu输出中的列的解释

| 列 名     | 列 描 述  |
|---------|--|
| runq-sz | 运行队列的大小。它不包括正在睡眠或等待输入/输出完成的进程；它包括存在于内存中并且正在等待运行的进程 |
| %runocc | 等待执行的进程占用的运行队列时间百分率                                |
| swpq-sz | 在执行监控的间隔期内交换队列的平均长度。准备运行但已经被换出的进程包含在此数量中           |
| %swpocc | 能够运行的进程（已换出但准备运行的进程）占用的交换队列的时间百分率                  |

从上面的输出中，你可以看到运行队列的容量仅为 1；这意味着在监控间隔期内，队列中仅有一个进程正在等待 CPU。在间隔期内，占用的运行队列百分率仅为整个时间的 25%。你可以推断出CPU在此时刻负载较轻。一般要注意一个超过 5或6的运行队列。假如该队列大于这些值，要么减少运行进程的数量、增加 CPU的数量，要么升级已有的 CPU。你可以鉴别正等待CPU资源的进程或使用下一节提到的技术加载 CPU。

### 22.2.2 寻找CPU密集型用户

有几种方法用来寻找系统中的 CPU密集型用户。你既可以使用操作系统工具，也可以使用

用存储在Oracle系统表中的信息来鉴别CPU密集型用户。在本节中，两种方法你都要检验一下。事实上，这两种方法都可以用来寻找系统中的CPU密集型用户。

### 1. 使用Oracle途径

Oracle动态系统性能表提供了有关系统中正在发生的事件的大量信息。V\$SYSSTAT与V\$SESSTAT视图是两个动态性能视图，可以使用它们来查找需要的信息。

存储于V\$SYSSTAT视图中的数值的通用列表在下面的清单 22-1中给出。你感兴趣的信息是CPU的使用情况，但是该视图可以用于许多其他的监控目的。

```
SQL> select * from v$sysstat order by class,ststistic#;
```

清单22-1是上述SQL语句的输出结果。

清单22-1 使用V\$SYSSTAT视图查找系统统计信息

| STATISTIC# | NAME                                   | CLASS | VALUE      |
|------------|--|-------|------------|
| 0          | logons cumulative                      | 1     | 2051       |
| 1          | logons current                         | 1     | 68         |
| 2          | opened cursors cumulative              | 1     | 72563      |
| 3          | opened cursors current                 | 1     | 636        |
| 4          | user commits                           | 1     | 289212     |
| 5          | user rollbacks                         | 1     | 7299       |
| 6          | user calls                             | 1     | 2574300    |
| 7          | recursive calls                        | 1     | 4726090    |
| 8          | recursive cpu usage                    | 1     | 1334927    |
| 9          | session logical reads                  | 1     | 205058382  |
| 10         | session stored procedure space         | 1     | 0          |
| 12         | CPU used by this session               | 1     | 4896925    |
| 13         | session connect time                   | 1     | 6.2794E+11 |
| 15         | session uga memory                     | 1     | 8599758248 |
| 16         | session uga memory max                 | 1     | 169781672  |
| 20         | session pga memory                     | 1     | 311833920  |
| 21         | session pga memory max                 | 1     | 324695784  |
| 101        | serializable aborts                    | 1     | 0          |
| 133        | bytes sent via SQL*Net to client       | 1     | 206511175  |
| 134        | bytes received via SQL*Net from client | 1     | 174496695  |
| 135        | SQL*Net roundtrips to/from client      | 1     | 2588500    |
| 136        | bytes sent via SQL*Net to dblink       | 1     | 0          |
| 137        | bytes received via SQL*Net from dblink | 1     | 0          |
| 138        | SQL*Net roundtrips to/from dblink      | 1     | 0          |
| 84         | redo entries                           | 2     | 2040536    |
| 85         | redo size                              | 2     | 502002531  |
| 86         | redo entries linearized                | 2     | 0          |
| 87         | redo buffer allocation retries         | 2     | 2391       |
| 88         | redo small copies                      | 2     | 1807035    |
| 89         | redo wastage                           | 2     | 76782505   |
| 90         | redo writer latching time              | 2     | 888        |
| 91         | redo writes                            | 2     | 183133     |
| 92         | redo blocks written                    | 2     | 576689     |
| 93         | redo write time                        | 2     | 502618     |
| 94         | redo log space requests                | 2     | 66         |
| 95         | redo log space wait time               | 2     | 2635       |
| 96         | redo log switch interrupts             | 2     | 0          |
| 97         | redo ordering marks                    | 2     | 0          |
| 22         | enqueue timeouts                       | 4     | 25         |
| 23         | enqueue waits                          | 4     | 57         |
| 24         | enqueue deadlocks                      | 4     | 0          |
| 25         | enqueue requests                       | 4     | 707487     |
| 26         | enqueue conversions                    | 4     | 7259       |
| 27         | enqueue releases                       | 4     | 707376     |

|   |    |           |
|---|----|-----------|
| 37 db block gets                                    | 8  | 5809539   |
| 38 consistent gets                                  | 8  | 202261999 |
| 39 physical reads                                   | 8  | 124879028 |
| 40 physical writes                                  | 8  | 367214    |
| 41 write requests                                   | 8  | 25221     |
| 42 summed dirty queue length                        | 8  | 37479     |
| 43 db block changes                                 | 8  | 3994241   |
| 44 change write time                                | 8  | 56399     |
| 45 consistent changes                               | 8  | 822006    |
| 46 redo synch writes                                | 8  | 23407     |
| 47 redo synch time                                  | 8  | 103518    |
| 48 exchange deadlocks                               | 8  | 0         |
| 49 free buffer requested                            | 8  | 123453575 |
| 50 dirty buffers inspected                          | 8  | 42044     |
| 51 free buffer inspected                            | 8  | 86462     |
| 52 commit cleanout failure: write disabled          | 8  | 0         |
| 53 commit cleanout failures: hot backup in progress | 8  | 0         |
| 54 commit cleanout failures: buffer being written   | 8  | 164       |
| 55 commit cleanout failures: callback failure       | 8  | 2769      |
| 56 total number commit cleanout calls               | 8  | 547911    |
| 57 commit cleanout number successfully completed    | 8  | 538392    |
| 58 DBWR timeouts                                    | 8  | 10503     |
| 59 DBWR make free requests                          | 8  | 38615     |
| 60 DBWR free buffers found                          | 8  | 18186694  |
| 61 DBWR lru scans                                   | 8  | 42222     |
| 62 DBWR summed scan depth                           | 8  | 20800400  |
| 63 DBWR buffers scanned                             | 8  | 20699788  |
| 64 DBWR checkpoints                                 | 8  | 998       |
| 70 recovery blocks read                             | 8  | 0         |
| 71 recovery array reads                             | 8  | 0         |
| 72 recovery array read time                         | 8  | 0         |
| 73 CR blocks created                                | 8  | 125716    |
| 74 Current blocks converted for CR                  | 8  | 14791     |
| 99 background checkpoints started                   | 8  | 18        |
| 100 background checkpoints completed                | 8  | 18        |
| 28 global lock gets (non async)                     | 32 | 1         |
| 29 global lock gets (async)                         | 32 | 0         |
| 30 global lock get time                             | 32 | 0         |
| 31 global lock converts (non async)                 | 32 | 0         |
| 32 global lock converts (async)                     | 32 | 0         |
| 33 global lock convert time                         | 32 | 0         |
| 34 global lock releases (non async)                 | 32 | 0         |
| 35 global lock releases (async)                     | 32 | 0         |
| 36 global lock release time                         | 32 | 0         |
| 78 next scns gotten without going to DLM            | 32 | 0         |
| 79 Unnecessary process cleanup for SCN batching     | 32 | 0         |
| 80 calls to get snapshot scn: kcmgss                | 32 | 2619387   |
| 81 kcmgss waited for batching                       | 32 | 0         |
| 82 kcmgss read scn without going to DLM             | 32 | 0         |
| 83 kcmccs called get current scn                    | 32 | 0         |
| 65 DBWR cross instance writes                       | 40 | 0         |
| 66 remote instance undo block writes                | 40 | 0         |
| 67 remote instance undo header writes               | 40 | 0         |

|  |     |            |
|--|-----|------------|
| 68 remote instance undo requests                               | 40  | 0          |
| 69 cross instance CR read                                      | 40  | 0          |
| 98 hash latch wait gets  | 40  | 0          |
| 118 table scans (short tables)                                 | 64  | 366085     |
| 119 table scans (long tables)                                  | 64  | 10819      |
| 120 table scans (rowid ranges)                                 | 64  | 15         |
| 121 table scans (cache partitions)                             | 64  | 0          |
| 122 table scans (direct read)                                  | 64  | 15         |
| 123 table scan rows gotten                                     | 64  | 904956876  |
| 124 table scan blocks gotten                                   | 64  | 129018277  |
| 125 table fetch by rowid                                       | 64  | 31893640   |
| 126 table fetch continued row                                  | 64  | 506029     |
| 127 cluster key scans  | 64  | 135034     |
| 128 cluster key scan block gets                                | 64  | 363979     |
| 129 parse time cpu   | 64  | 34740      |
| 130 parse time elapsed   | 64  | 55148      |
| 131 parse count  | 64  | 532516     |
| 132 execute count  | 64  | 2390178    |
| 139 sorts (memory)   | 64  | 40412      |
| 140 sorts (disk)   | 64  | 268        |
| 141 sorts (rows)   | 64  | 49141580   |
| 142 session cursor cache hits                                  | 64  | 0          |
| 143 session cursor cache count                                 | 64  | 0          |
| 11 CPU used when call started                                  | 128 | 4896911    |
| 14 process last non-idle time                                  | 128 | 6.2794E+11 |
| 17 messages sent   | 128 | 160593     |
| 18 messages received   | 128 | 160593     |
| 19 background timeouts   | 128 | 31883      |
| 75 calls to kcmgcs   | 128 | 90759      |
| 76 calls to kcmgrs   | 128 | 4168092    |
| 77 calls to kcmgas   | 128 | 306090     |
| 102 transaction lock foreground requests                       | 128 | 0          |
| 103 transaction lock foreground wait time                      | 128 | 0          |
| 104 transaction lock background gets                           | 128 | 0          |
| 105 transaction lock background get time                       | 128 | 0          |
| 106 transaction tables consistent reads - undo records applied | 128 | 143669     |
| 107 transaction tables consistent read rollback                | 128 | 131        |
| 108 data blocks consistent reads - undo records applied        | 128 | 678275     |
| 109 no work - consistent read gets                             | 128 | 194428691  |
| 110 cleanouts only - consistent read gets                      | 128 | 25194      |
| 111 rollbacks only - consistent read gets                      | 128 | 86452      |
| 112 cleanouts and rollbacks - consistent read gets             | 128 | 54180      |
| 113 rollback changes - undo records applied                    | 128 | 14117      |
| 114 transaction rollbacks                                      | 128 | 1520       |
| 115 immediate (CURRENT) block cleanout applications            | 128 | 73789      |
| 116 immediate (CR) block cleanout applications                 | 128 | 79374      |
| 117 deferred (CURRENT) block cleanout applications             | 128 | 264576     |
| 144 cursor authentications                                     | 128 | 95248      |

注意 当你使用该视图时要注意。由于在某些Oracle版本中存在程序错误，所以该视图中的有些统计值非常大，这些统计值或许会显示错误的数值。

你所感兴趣的统计量是CPU used by this session，它的statistic#值为12：

```
12 CPU used by this session          1      4896925
SQL> select * from v$sysstat WHERE name = 'CPU used by this session' order by
class,statistic#
```

现在CPU used by this session的值是在百分之一秒内。通过修改如下的查询把该值转化成分钟：

```
col name format a35
col value format 999.99 heading "Time in | Mins"
select statistic#,name,class ,value/60/100 value
from v$sysstat
where statistic# = 12
/
```

如下的列表是上述SQL语句的输出值。

| STATISTIC# | NAME                     | CLASS | Mins   |
|------------|--------------------------|-------|--------|
| 12         | CPU used by this session | 1     | 817.56 |

该输出表明RDBMS与它所有的后台和前台服务器进程自启动以来已经消耗了大约 817分钟的CPU时间。这是又一条总体水平上的信息。你现在想要查找某一个会话所消耗的 CPU资源的数量。

还有一个叫做 V\$SESSTAT的动态性能视图。基本上，这两个视图之间唯一的区别就是 V\$SYSSTAT存储摘要级的信息，而 V\$SESSTAT存储会话级的信息；V\$SESSTAT可以被称为 V\$SYSSTAT的子表。你可以使用 V\$SESSTAT中的信息来发现哪一个会话消耗 CPU的时间最多。

你通过使用清单22-2中的脚本执行此操作。

清单22-2 产生CPU使用报表的脚本和输出值

```
col prog format a10
col value format 9999.99 heading "Time In| Mins"
```

```
Select a.sid,
        spid,
        status,
        substr(a.program,1,10) prog,
        a.terminal,
        osuser,
        value/60/100 value
From    v$session a,
        v$process b,
        v$sesstat c
Where   c.statistic# = 12
And     c.sid       = a.sid
And     a.paddr     = b.addr
Order by value desc;
```

The following is the output of the above SQL statement.

| Time In | SID | SPID  | STATUS   | PROG               | TERMINAL | Schema  | Mins  |
|---------|-----|-------|----------|--------------------|----------|---------|-------|
|         | 45  | 11145 | INACTIVE | C:\ARMS\BS Windows | PC       | TIFFANY | 95.87 |
|         | 95  | 12370 | INACTIVE | C:\WINDOWS Windows | PC       | SUSANE  | 22.99 |
|         | 47  | 9778  | INACTIVE | C:\ARMS\BS Windows | PC       | LIN     | 10.22 |

|     |       |          |            |         |    |           |      |
|-----|-------|----------|------------|---------|----|-----------|------|
| 22  | 9295  | ACTIVE   | C:\ARMS\DE | Windows | PC | INGEMAR   | 3.96 |
| 37  | 14427 | INACTIVE | C:\ARMS\CS | Windows | PC | SHARON    | .50  |
| 107 | 9454  | INACTIVE | sqlplus@ar | ttyp2   |    | MARK      | .34  |
| 65  | 16959 | INACTIVE | C:\ARMS\CS | Windows | PC | APEKSH    | .32  |
| 35  | 17077 | INACTIVE | C:\ARMS\CS | Windows | PC | TEUTA     | .30  |
| 34  | 18995 | INACTIVE | C:\ARMS\CS | Windows | PC | KASTURI   | .23  |
| 23  | 578   | INACTIVE | C:\ARMS\CS | Windows | PC | REEHER    | .20  |
| 33  | 16965 | INACTIVE | C:\ARMS\CS | Windows | PC | JULIE     | .20  |
| 71  | 16973 | INACTIVE | C:\ARMS\CS | Windows | PC | AMY       | .19  |
| 92  | 16989 | INACTIVE | C:\ARMS\CS | Windows | PC | BARRETT   | .17  |
| 82  | 8004  | INACTIVE | C:\ARMS\CS | Windows | PC | GABRIELLE | .17  |
| 96  | 17090 | INACTIVE | C:\ARMS\CS | Windows | PC | MANISH    | .16  |
| 20  | 5466  | INACTIVE | C:\ARMS\CS | Windows | PC | BURTNER   | .16  |
| 29  | 17033 | INACTIVE | C:\ARMS\CS | Windows | PC | DAVID     | .14  |
| 43  | 16953 | INACTIVE | C:\ARMS\CS | Windows | PC | ADITI     | .13  |
| 77  | 16947 | INACTIVE | C:\ARMS\CS | Windows | PC | GARY      | .12  |
| 54  | 8971  | INACTIVE | C:\ARMS\CS | Windows | PC | RANDY     | .11  |

从清单 22-2 中，你可以推断出会话 45 与 95 是系统中的顶端 CPU 用户，它们消耗 CPU 的时间分别为 95 分钟和 22 分钟。此输出存在的唯一问题是它反映了自会话开始以来的累积统计值，而不是一个时间间隔内的统计值。

假如会话 A 在 8 小时内已占用了 50 分钟的 CPU 时间，会话 B 在 1 小时中已经占用了 30 分钟的 CPU 时间，此输出将报告会话 A 较会话 B 来说是高 CPU 用户——事实并非如此，因为会话 B 的 CPU 使用率远超过会话 A 的 CPU 使用率。然而，此输出对系统中的顶端 CPU 用户做了一个相当精确的概述。

假如你希望更精确的话，你必须捕获经过一个时间间隔的统计值。选择系统在一天中最忙的一段时期，然后捕获该会话的 CPU 使用信息并把它存储在一个临时表中。在繁忙期过去之后，捕获每一个使用 V\$SESSTAT 视图的会话的 CPU 使用信息并把它存储在一个临时表中。在存储于临时表中的两个快照中捕获的信息能够用来了解时间间隔期内单个会话的 CPU 使用率以及总 CPU 使用情况。这可以用清单 22-3 中的脚本来实现。

清单 22-3 创建存储快照信息的临时表

```

REM Table to store the start snapshot values
Create table stat$cpu_begin
(sid number,
spid varchar2(9),
schemaname varchar2(30),
cpu_usage number(10),
program varchar2(48),
time_stamp date);
REM Table to store the end snapshot values
Create table stat$cpu_end
(sid number,
spid varchar2(9),
schemaname varchar2(30),
cpu_usage number(10),
program varchar2(48),
time_stamp date);

```

在该表创建之后，使用清单 22-4 中的脚本可以对第一个快照进行处理。

清单 22-4 用于捕获 stat\$cpu\_begin 表中的起始 CPU 统计信息的脚本

```

Insert Into stat$cpu_begin

```

```

Select a.sid,
       b.spid,
       a.schemaname,
       value,
       a.program,
       sysdate
From   v$session a,
       v$process b,
       v$sesstat c
Where  c.statistic# = 12
And    c.sid       = a.sid
And    a.paddr     = b.addr;

```

类似地，在繁忙期结束后，你可以再运行清单 22-5中的脚本并生成stat\$cpu\_end表。

清单22-5 用于捕获stat\$cpu\_end表中的结束CPU统计信息的脚本

```

Insert Into stat$cpu_end
Select a.sid,
       b.spid,
       a.schemaname,
       value,
       a.program,
       sysdate
From   v$session a,
       v$process b,
       v$sesstat c
Where  c.statistic# = 12
And    c.sid       = a.sid
And    a.paddr     = b.addr;

```

现在准备生成CPU密集型用户报告以及CPU使用率。该报表更精确，因为它报告了CPU的使用率，该使用率是确定资源密集型用户的一个重要因素。该报表用每小时使用CPU的分钟数显示了每个会话的CPU消耗率。此快照技术消除了清单 22-5中的差异。清单 22-6中显示的脚本用来产生CPU使用报告。

清单22-6 用于使用快照生成CPU使用报告的脚本

```

Break on sid on spid on tot_usg on mins_per_hr
Select e.sid,
       e.spid,
       e.cpu_usage,
       b.cpu_usage,
       to_char((e.cpu_usage - b.cpu_usage)/(6000),'9999.99') tot_usg_mins
,
       to_char(((e.cpu_usage - b.cpu_usage)/(6000))/((e.time_stamp - b.time_stamp)*24),'99.99') mins_per_hr
From   stat$cpu_end e, stat$cpu_begin b
Where  b.sid = e.sid
And    b.spid = e.spid
order by mins_per_hr;

```

注意 此SQL代码运行在使用V\$SESSTAT视图捕获的统计数字基础之上，此V\$SESSTAT视图是对于统计量为12（CPU used by this session）的视图。如果该值在返回到客户进程前的退出用户调用时被Oracle加1，那么使用该值有时会出问题。因此，假如你正在监控一个具有还没有返回的用户调用的批处理作业，那么CPU使用值或许不能在V\$SESSTAT视图中更新并且你可能会得到不准确的结果。当你正在使用以前的SQL列

表时要注意这一点。假如监控时间间隔较大且所有的会话在此间隔内的某个时间点上更新V\$SESSTAT视图，这样就可以得到准确的结果。

既然你已经识别了大量使用 CPU资源的会话，那么你可以确定正在执行的大量使用 CPU的会话是什么。

清单22-7中的脚本可以用来发现会话当前正在执行的 SQL语句。这清楚地指出了会话正在执行哪一个程序，当前代码段是什么以及是否需要在该 SQL上执行一个查询优化。

清单22-7 用于发现会话当前正在执行的 SQL语句的脚本

---

```
set head off
set long 9000
set linesize 100
set pagesize 30
set feedback off
set wrap on
set verify off

select sql_text
from v$sqltext , v$session
where sid = 45
and v$sqltext.address = v$session.sql_address
order by piece

The following listing is the output of the above SQL statement.

SQL_TEXT
-----
Select * from employee where emp_id =800
```

---

注意 这里提到的监控技术假定用户不经常注册或注销。假如用户经常注册或注销，那么由Oracle分配的会话标识会变化，使得在会话级跟踪或报告会话变得困难起来。

注意 因为繁忙的时间间隔已经过去，所以它与执行清单22-7中的脚本无关。然而，通过在间隔期内周期性地查询，你可以使用该脚本在临时表中存储会话正在执行的所有 SQL命令。你能够使用下面的策略完成这项工作。

使用清单22-8中的脚本创建临时表并存储由会话执行的 SQL语句。

清单22-8 用于创建SQL执行历史记录表stat\$session\_hash的脚本

---

```
Create table stat$session_hash
(sid number,
spid varchar2(9),
schemaname varchar2(30),
hash_value number,
time_stamp date,
num_occ number);
```

---

在创建了该表后，你可以运行清单22-9中的脚本以创建sp\_stat\$sample\_hash过程，该过程在SQL执行历史记录表stat\$session\_hash中存储由监控期内的不同会话执行的所有 SQL语句。

清单22-9 用于创建sp\_stat\$sample\_hash过程的脚本

---

```
Cursor c_stat$session is
Select sid,
```

---

```

        spid,
        schemaname,
        sql_hash_value
    From v$session a,
        v$process b
    Where a.paddr = b.addr;

v_stat$session      c_stat$session%RowType;
v_stat$session_hash stat$session_hash%RowType;
e_next              exception;

Begin

    Open c_stat$session ;

    Loop

    Begin
    Fetch c_stat$session
    Into v_stat$session;

    Begin

        Select *
        Into   v_stat$session_hash
        From   stat$session_hash
        Where  sid = v_stat$session.sid
        And    spid = v_stat$session.spid
        And    hash_value = v_stat$session.sql_hash_value;

        Insert Into stat$session_hash
        values
        (v_stat$session.sid,
         v_stat$session.spid,
         v_stat$session.schemaname,
         v_stat$session.sql_hash_value,
         sysdate,
         1
        );

    Exception
    When No_Data_Found Then
    Update stat$session_hash
    Set num_occ = num_occ + 1
    Where  sid = v_stat$session.sid
    And    spid = v_stat$session.spid
    And    hash_value = v_stat$session.sql_hash_value;

    When Others Then
    Raise e_next;

    End;

Exception

When e_next Then
Null;

When Others Then
Null;

End;

Commit;

```

```

End Loop;

Close c_stat$session;

End;

```

该过程 ( sp\_stat\$sample\_hash ) 能够在监控期内间歇地运行。在捕获了开始和结束 CPU 使用信息以及 SQL 执行历史之后, 你可以产生一个 CPU 密集型用户以及每一个会话在快照期内所执行的 SQL 语句的报告。该报告随后便可以用来调整引起大量使用 CPU 的 SQL 语句。清单 22-10 给出了产生大量使用 CPU 报告的脚本以及由会话执行的 SQL 语句的哈希值。

清单 22-10 用于产生 CPU 使用报告以及在监控期内执行的 SQL 语句的哈希值的脚本

```

Break on sid on spid on tot_usg_mins on mins_per_hr

Select e.sid,
       e.spid,
       to_char((e.cpu_usage - b.cpu_usage)/(6000), '9999.99') tot_usg_mins,
       to_char(((e.cpu_usage - b.cpu_usage)/(60100))/((e.time_stamp - b.time_stamp)
)*24), '99.99') mins_per_hr,
       hash_value,
       num_occ
From   stat$cpu_end e, stat$cpu_begin b , stat$session_hash c
Where  b.sid = e.sid
And    b.spid = e.spid
And    e.sid = c.sid
And    e.spid = c.spid
order by mins_per_hr;

```

The following listing is the output of the above SQL statement.

| SID | SPID  | Tot Usg<br>In Mins | Mins<br>Per Hr | HASH_VALUE | NUM_OCC |
|-----|-------|--------------------|----------------|------------|---------|
| 23  | 9461  | 29.04              | 9.68           | -1.447E+09 | 7       |
| 24  | 9776  | 25.38              | 8.46           | 1739700154 | 1       |
|     |       |                    |                | -1.306E+09 | 5       |
|     |       |                    |                | 1014890971 | 1       |
| 34  | 11525 | 22.41              | 7.47           | 573420945  | 7       |
| 35  | 9459  | 17.55              | 5.85           | 1739700154 | 2       |
|     |       |                    |                | 1561770773 | 1       |
| 35  | 9459  | 10.32              | 3.44           | 2113573249 | 1       |
| 59  | 9531  | .00                | .00            | 1739700154 | 6       |
| 56  | 10010 | .00                | .00            | 573420945  | 7       |
| 53  | 11114 | .00                | .00            | -2.056E+09 | 7       |
| 52  | 10014 | .00                | .00            | 1739700154 | 3       |
| 105 | 9533  | .00                | .00            | 573420945  | 7       |
| 86  | 9529  | .00                | .00            | 573420945  | 7       |
|     |       |                    |                | 1907631373 | 1       |
| 73  | 9527  | .00                | .00            | 573420945  | 7       |
| 62  | 9541  | .01                | .00            | -1.306E+09 | 5       |
| 109 | 16988 | .00                | .00            | 1848020776 | 7       |
| 52  | 10014 | .00                | .00            | -1.306E+09 | 1       |

从此输出中, 你能够很容易地确定会话 23 与 24 是顶端 CPU 用户, 他们每小时使用 CPU 的时间分别是 9.68 分钟与 8.46 分钟。使用 sql\_hash\_value 与 V\$SQLTEXT 视图, 你可以确定长时间使用 CPU 的 SQL 语句并采取相应的措施。清单 22-10 中的 num\_occ 列显示了在监控期内执行 SQL 语句的次数。

注意 大量使用CPU的SQL语句很可能是全表扫描或系统中引起更多物理读取的SQL语句。当所需要的数据已经不在数据块缓冲区时会发生遗漏，数据必须从物理设备中读取。假如没有空闲的空间，数据遗漏会引发缓冲区管理器在缓冲区缓存中为将要读取的数据分配空间。然后，缓冲区管理器把要被写出的数据块移动到脏列表中以便它们可以随后被写出到磁盘中。与直接从高速缓存中读出数据相比，缓冲区管理器的处理要求使物理读取成为一个CPU密集型操作。

## 2. 使用操作系统途径

你可以使用top命令查找UNIX系统中的CPU密集型用户。top命令按照等待时间由高到低的顺序列举了最大量使用CPU用户。当顶端CPU用户被找到后，你可以发现该用户正在做什么。清单22-11是top命令的一个输出示例。

清单22-11 top命令的输出示例

```
%top
System: arms1                               Sat Aug 30 16:35:57 1997
Load averages: 0.59, 0.61, 0.69
20 processes: 18 sleeping, 2 running
Cpu states:
CPU   LOAD   USER    NICE    SYS   IDLE   BLOCK  SWAIT   INTR   SSYS
0     0.56  34.6%   0.0%    1.4%  64.0%  0.0%   0.0%   0.0%   0.0%
1     0.61  57.6%   0.0%    1.2%  41.3%  0.0%   0.0%   0.0%   0.0%
...
avg   0.59  46.2%   0.0%    1.2%  52.7%  0.0%   0.0%   0.0%   0.0%

Memory: 25956K (10740K) real, 37996K (19316K) virtual, 15364K free Page# 1/14

CPU  TTY  PID USERNAME  PRI NI  SIZE  RES  STATE  TIME %WCPU %CPU COMMAND
1    ?  28586 manish   238 20  9164K 1072K run   157:22 90.13 89.97 oracleor
1    ?  1020 oracle    154 20  8712K  616K sleep 0:10  1.47  1.46 oracleor
1    ?  1900 oracle    154 20  8692K  588K sleep 0:05  1.10  1.09 oracleor
1    ?  1988 oracle    154 20  8620K  528K sleep 0:04  0.55  0.55 oracleor
0    ?  822 oracle     154 20  8764K  656K sleep 0:24  0.51  0.51 oracleor
1    ?  1793 oracle    154 20  8636K  524K sleep 0:01  0.43  0.43 oracleor
1    ?  53 root        100 20    0K    0K sleep 226:02 0.37  0.37 netisr
0    ?  304 root        154 20   24K   24K sleep 258:04 0.33  0.33 syncer
0    ?  0 root         127 20    0K    0K sleep 86:40 0.30  0.30 swapper
0    p2 2204 evett     178 20   720K  312K run    0:00  0.92  0.27 top
1    ? 29494 oracle    154 20  8772K  660K sleep 0:28  0.23  0.23 oracleor
```

从此输出中，很明显可以看出PID为28586并且USERNAME为manish的用户是CPU密集型用户。注意，从USER、SYS与IDLE的值中，你可以看到系统显示USER的值为46%，SYS为1.2%，其余是空闲的。这意味着该系统主要服务于用户请求，这是一个好的迹象。尽管用户是顶端CPU用户，但是这不必惊慌，因为在一个负载较轻的系统中，如果有一个需要大量CPU的进程，那么它会独占CPU。

要查看的关键数据是系统的负载平均值。第一行显示系统的负载平均值为0.59、0.61和0.69。0.59是最后1分钟的负载平均值，0.61是最后5分钟的负载平均值，0.69是最后15分钟的负载平均值。负载平均值清楚地表明负载量系统内以及最后的15分钟内的情况，当CPU负载量突然变化并且系统性能变坏时，负载平均值是非常有用的信息。当负载平均值突然增加时，你可以推断出这是由于系统中某些资源密集型进程而导致的。系统装载量为2或3表明系统轻度负载，装载量为5或6表明系统中度负载。其他要查看的信息有%CPU列，它指示了由特定进程使用的CPU

时间的百分率。如果该数值较高，则很明显表明该进程是一个大量花费CPU时间的进程。

你还可以使用ps命令确定CPU密集型用户。ps命令包括了一个显示CPU在服务于特殊进程时所花费时间的列。你可以按照花费CPU时间由高到低的顺序对ps的输出进行排序，并再次找出顶端CPU资源用户。

清单22-12显示PID为28586和14171是顶端CPU资源。自进程开始以来，28586已经使用了192.21分钟的CPU时间，14171已经使用了152.29分钟的CPU时间。

清单22-12 使用ps命令得到CPU使用情况

```
$ ps -ael | sort -n -r -k 13
```

| F S             | UID | PID   | PPID  | C  | PRI | NI | ADDR    | SZ   | WCHAN    | TTY | TIME   | COMD     |
|-----------------|-----|-------|-------|----|-----|----|---------|------|----------|-----|--------|----------|
| 1 R             | 105 | 28586 | 28586 | 89 | 200 | 20 | 3903a00 | 1741 | -        | ?   | 192:21 | oracle   |
| 1 S             | 104 | 14171 | 1     | 0  | 154 | 20 | 3937a00 | 1711 | 36f8b38  | ?   | 152:29 | oracle   |
| 3 S             | 0   | 3     | 0     | 0  | 128 | 20 | 2529100 | 0    | 3c2050   | ?   | 122:09 |          |
| ↳statdaemon     |     |       |       |    |     |    |         |      |          |     |        |          |
| 3 S             | 0   | 0     | 0     | 0  | 127 | 20 | 40c8b0  | 0    | 3c2050   | ?   | 86:47  | swapper  |
| 3 S             | 0   | 2     | 0     | 0  | 128 | 20 | 2155e80 | 0    | 40e990   | ?   | 32:19  | vhand    |
| 1 S             | 0   | 459   | 458   | 0  | 127 | 20 | 2718180 | 2    | 7ffe6000 | ?   | 32:05  | netfmt   |
| 1 S             | 0   | 1016  | 1     | 0  | 154 | 20 | 2874180 | 9    | 40e0a8   | ?   | 22:52  | spserver |
| 1 S             | 0   | 787   | 1     | 0  | 154 | 20 | 280e880 | 180  | 40e0a8   | ?   | 21:01  | dced     |
| 1 S             | 0   | 681   | 1     | 0  | 154 | 20 | 27cef80 | 85   | 40e0a8   | ?   | 15:47  | snmpdm   |
| 1 S             | 104 | 1339  | 1     | 0  | 154 | 20 | 2845980 | 195  | 40e0a8   | ?   | 14:18  | tnslsnr  |
| 3 S             | 0   | 12    | 0     | 0  | 138 | 20 | 254de00 | 0    | 391cf40  | ?   | 12:00  |          |
| ↳vx_sched_threa |     |       |       |    |     |    |         |      |          |     |        |          |
| 3 S             | 0   | 54    | 0     | 0  | 100 | 20 | 254d980 | 0    | -        | ?   | 6:24   | nvsisr   |
| 1 S             | 125 | 13379 | 1     | 0  | 156 | 20 | 3330200 | 1535 | 461a84   | ?   | 5:21   | oracle   |
| 3 S             | 0   | 4     | 0     | 0  | 128 | 20 | 2529380 | 0    | 40e4dc   | ?   | 4:09   |          |
| ↳unhashdaemon   |     |       |       |    |     |    |         |      |          |     |        |          |
| 1 S             | 125 | 13383 | 1     | 0  | 156 | 20 | 3739d80 | 1529 | 461a94   | ?   | 4:41   | oracle   |
| 1 S             | 104 | 21303 | 1     | 0  | 154 | 20 | 2948d80 | 1846 | 3e55438  | ?   | 4:02   | oracle   |

注意 这些进程可能是非活动的，所以注意那些已经在一天的开始时消耗了大量CPU资源、目前处于空闲状态的会话。要确定进程是否是空闲的，再次运行相同的命令并检查CPU时间是否增加了。活动会话可以通过查看清单22-12中的第二列来得到识别。数值R表明它正在运行。即便在这里，一个具有S状态的进程当前也可以在睡眠，因为它正在等待某些事情，然后转入运行状态。为了识别一个当前处于活动状态的顶端CPU资源，请记住以上所有的这些东西。换句话说，显示于Oracle途径中的快照技术可以用于识别。最后，管理员可以把来自Oracle途径的值以及来自操作系统途径的值在资源密集型用户中调整归零。

在pid被识别之后，你可以找到sid进程并进一步使用Oracle的V\$SQLTEXT视图来查找当前正在执行的进程。这可以使用清单22-13中的脚本来完成。

清单22-13 用于从sid获得pid并获得会话正在执行的SQL语句的脚本

```
REM Find the sid of the Unix process using the process id
Select sid
From v$session a,
     v$process b
Where a.paddr = b.addr
And b.spid = <pid>;

REM Find the SQL the user is currently executing
Select sql_text
```

```
From v$sqltext a ,
      v$session b
Where   a.hash_value = b.sql_hash_value
And     b.sid         = <sid>
Order By piece;
```

在Windows NT操作系统中，你可以监控处理器对象并使用 %处理器时间计数器生成一张一定时期内的CPU使用情况图。你可以创建数据的折线图或条形图。

大量使用磁盘可能引起CPU性能问题。当磁盘正在执行大量的输入/输出时，CPU会因为处理输入/输出中断而陷入停顿。你可以在 PerfMon ( Windows NT操作系统 ) 中通过查看处理器：中断数/秒和处理器：%中断时间来对此进行监控。

假如CPU花费过多的时间处理中断，可能会引发严重的性能衰退。还有，你可以检查处理器应用时间（处理器：%处理器时间），如图22-2所示。假如该时间量的持续时间超过 90%，就说明有一个CPU瓶颈。

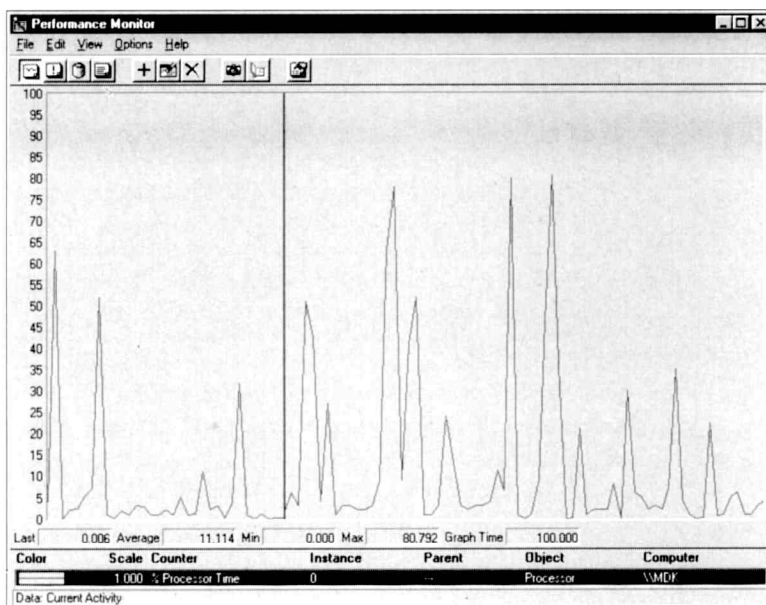


图22-2 使用Oracle 8性能监控程序监控处理器使用时间

## 22.3 文件输入/输出

当某个用户程序需要对数据进行存取时，操作系统首先查看该数据当前在高速缓存中是否可用，以及如果可用的话，它将如何被使用。假如在高速缓存中没有找到该数据，那么就不得不从磁盘中读取数据。与从高速缓存中读取数据相比，从磁盘中读取数据的过程代价很大，因为内存存取比磁盘存取更快。磁盘操作极大地阻碍了进程的吞吐量。

基于磁盘存取的进程性能依赖于诸如输入/输出控制卡的数量、软件、磁盘速度等因素。

当一个进程请求数据时，下面所列的是检索被请求的数据时发生的短暂事件：

- 1) 输入/输出控制器把数据地址定位在正确的磁盘上；这被称作队列延迟 ( queuing delay )。一般还存在其他一些在给定的时刻从磁盘中读取数据的未完成的请求。花费在队列中的时间依赖于操作系统与磁盘上装载的系统。对磁盘中未完成的请求进行排序以便磁盘头不必做太

多的来回移动。这种对请求的排序可能导致请求在队列中花费更多的时间，但搜索时间减少了。

2) 当数据的存放位置找到后，磁盘臂——或者更精确地说，磁盘头必须定位在正确磁盘的正确磁道上。发生在这里的时间延迟叫做搜索时间（seek time）。

3) 当磁盘在磁盘头下旋转时，磁盘头读取该磁盘的数据。在磁盘头被定位在正确的磁道后，它必须等待磁盘旋转到相关的数据扇区刚好位于磁盘头下，这样便能完成读 / 写操作。这个阶段通常称为旋转等待时间（rotational latency）。

4) 通过磁盘头读出的数据被传送到控制器。这最后一部分的输入 / 输出被称为数据传送时间（data transfer time）。最终，该数据在输入 / 输出板上传送并发送到发出数据请求的进程。

用于从磁盘中检索数据的总时间可以粗略地表达为如下形式：

排队时间 + 搜索时间 + 旋转等待时间 + 数据传送时间

可以清楚地看到，磁盘存取包含磁盘的机械和旋转运动，假如有多个用户并行地在同一张磁盘上频繁进行输入 / 输出调用时，磁盘存取或许会成为系统中的瓶颈。最终，磁盘的输入 / 输出率支配了该磁盘的总吞吐量，在该磁盘中形成用户队列。

### 22.3.1 输入/输出预览

在你开始查找输入 / 输出密集型用户以前，必须确定任何一个文件系统当前是否被频繁地存取。如果是这样的话，开始定位这些用户。

像以前一样，得到短时间段的系统快照，然后获得时间间隔输出。通过查看输出值，很容易估价是否正在进行大量的输入 / 输出。

运行清单 22-14 的脚本可以对系统有一个快速的了解。在该清单中，有 stat\$begin\_file 和 stat\$end\_file 两个表用于存储开始和结束文件输入 / 输出的统计信息。利用两个快照之间的差别与时间，你可以计算出文件存取率。运行清单 22-14，在该快照间隔的起始处存储文件输入 / 输出统计信息。

清单 22-14 用于在 stat\$begin\_file 表中创建临时表和插入值的脚本

```
REM
REM Start IO : Script to Capture File I/O Statistics at the beginning of the
interval.
REM
Drop View stats$file_view;
Create View stats$file_view As
  Select ts.name      ts,
         i.name       name,
         x.phyrds     pyr,
         x.phywrt     pyw,
         x.readtim     prt,
         x.writetim    pwt,
         x.phyblkrd    pbr,
         x.phyblkwrt   pbw,
         sysdate       st_time
  From v$filestat x, ts$ ts, v$datafile i, file$ f
 Where i.file# = f.file#
 And ts.ts# = f.ts#
 And x.file# = f.file#;

Drop Table stats$begin_file;
```

```
Create Table stats$begin_file Tablespace tools
Storage(Initial 10K Next 10K Pctincrease 10 ) As
Select *
From stats$file_view
Where 0 = 1;
```

```
Drop Table stats$end_file;
Create Table stats$end_file Tablespace tools
Storage(Initial 10K Next 10K Pctincrease 10 )As
Select *
From stats$begin_file;
```

```
Insert Into stats$begin_file
Select *
From stats$file_view;
```

在想要的间隔结束后，运行结束输入/输出脚本以便在间隔结束时捕获统计数字以对系统的性能进行分析。结束输入/输出脚本捕获统计数字并存储于 stat\$end\_file表中，然后产生输入/输出报告。清单 22-15 是用于结束输入/输出并获得输入/输出使用报表的脚本。

清单 22-15 用于填充 stat\$ file\_end 表并产生输入/输出使用报告的脚本

```
REM
REM End IO : Script to Capture File I/O Statistics at the end of the interval.
REM
Set Term Off
Insert Into stats$end_file
Select *
From stats$file_view;

Drop Table stats$files;
Create Table stats$files Tablespace tools
Storage(Initial 10K Next 10K Pctincrease 0)As
Select b.ts table_space,
b.name file_name,
e.pyr-b.pyr phys_reads,
e.pbr-b.pbr phys_blks_rd,
e.prt-b.prt phys_rd_time,
e.pyw-b.pyw phys_writes,
e.pbw-b.pbw phys_blks_wr,
e.pwt-b.pwt phys_wrt_tim,
(e.st_time -b.st_time)*24*60*60 tot_tim
From stats$begin_file b, stats$end_file e
Where b.name=e.name;

Drop Table stats$begin_file;
Drop Table stats$end_file;
Set Term On
Clear Screen
Set Pause On
Set Pagesize 24
Set Feedback off
Col table_space Format a17
Col file_name Format a55
Col phys_writes Format 9999999 Heading 'PHY_WRS'
Col phys_reads Format 9999999 Heading 'PHY_RDS'
Col phys_rd_time Format 999999 Heading 'PHY_RDT'
Col phys_wrt_tim Format 999999999 Heading 'PHY_WRT'
Col phys_blks_rd Format 999999999 Heading 'BLKS_RD'
Col phys_blks_wr Format 9999999 Heading 'BLKS_WR'
Col ios_sec Format 999 Heading 'IOS_SEC'
```

```

REM
REM Generate IO Overview Report
REM
Select table_space,
       file_name,
       phys_reads,
       phys_writes,
       phys_blks_rd,
       phys_blks_wr,
       phys_rd_time,
       phys_wrt_tim,
       (phys_reads + phys_writes)/tot_tim ios_sec
From stats$files
Order By ios_sec Desc

```

The following listing is the output of the above SQL statement.

| TABLE_SPACE      | FILE_NAME                                      |
|------------------|--|
| PHY_RDS PHY_WRS  | BLKS_RD BLKS_WR PHY_RDT PHY_WRT IOS_SEC        |
| ARMSLIVEDB04_TS  | /u03/oradata/ora7/armslive/armslivedbu04.dbf   |
| 23100 34         | 23100 34 17644 106 93                          |
| ARMSLIVEIDX02_TS | /da11/oradata/ora7/armslive/armsliveidxa07.dbf |
| 16886 0          | 16886 0 912 0 68                               |
| ARMSLIVEIDX11_TS | /u14/oradata/ora7/armslive/armsliveidx19.dbf   |
| 3461 542         | 3461 542 1288 2946 16                          |
| ARMSLIVEDB04_TS  | /u03/oradata/ora7/armslive/armslivedbu03.dbf   |
| 2521 38          | 2521 38 2131 255 10                            |
| RBS2             | /db07/oradata/ora7/rbs4ora7.dbf                |
| 1 982            | 1 982 3 7890 4                                 |
| ARMSLIVEIDX03_TS | /db04/oradata/ora7/armslive/armliveidxb01.dbf  |
| 870 51           | 870 51 323 124 4                               |
| ARMSLIVETMP01_TS | /db06/oradata/ora7/armslive/armslivetmpb01.dbf |
| 866 1            | 24729 1 1402 2 3                               |
| ARMSLIVEDB12_TS  | /u03/oradata/ora7/armslive/armslivedbu19.dbf   |
| 157 323          | 157 323 32 1065 2                              |
| ARMSLIVEDB11_TS  | /u09/oradata/ora7/armslive/armslivedbu16.dbf   |
| 87 331           | 273 331 30 1693 2                              |
| ARMSLIVEIDX10_TS | /u16/oradata/ora7/armslive/armsliveidx17.dbf   |
| 137 25           | 137 25 69 66 1                                 |
| SYSTEM           | /u01/oradata/ora7/syst1ora7.dbf                |
| 63 23            | 79 23 31 204 0                                 |
| SYSTEM           | /u01/oradata/ora7/syst2ora7.dbf                |
| 34 12            | 37 12 11 90 0                                  |
| ARMSLIVEDB02_TS  | /db06/oradata/ora7/armslive/armslivedb06.dbf   |
| 7 32             | 7 32 1 203 0                                   |

从此输入/输出总报告中可以看出，系统中很明显存在一些输入/输出活动并且受此影响最严重的表空间是 ARMSLIVEDB04\_TS。该表空间的输入/输出率大约是每秒完成 93 次读/写。根据磁盘的速度或磁盘能够支持的输入/输出率，可以确定获得的数值是否是高的。在发现系

统中有大量活动发生后，你现在可以鉴别引起此大量输入/输出活动的用户。

你还可以在UNIX操作系统上使用 sar 命令得到操作系统级的一个抽样的概况。清单 22-16 中的sar命令每5秒钟对磁盘活动进行3次采样。

清单22-16 用于监控磁盘输入/输出统计数字的sar命令

```
$ sar -d 5 3
```

---

```
HP-UX arms1 B.10.10 U 9000/819    08/30/97
```

|          | device | %busy | avque | r+w/s | blks/s | avwait | avserv |
|----------|--------|-------|-------|-------|--------|--------|--------|
| 19:43:00 | c0t5d0 | 76.05 | 1.94  | 169   | 2711   | 8.76   | 5.94   |
| 19:43:05 | c0t4d0 | 9.58  | 0.95  | 12    | 192    | 8.86   | 9.04   |
|          | c0t3d0 | 8.18  | 0.50  | 6     | 102    | 4.77   | 12.28  |
|          | c3t6d0 | 7.58  | 0.50  | 28    | 447    | 5.20   | 3.12   |
|          | c4t6d1 | 15.77 | 1.30  | 28    | 450    | 8.44   | 14.35  |
|          | c4t5d0 | 6.19  | 0.50  | 5     | 77     | 5.02   | 12.64  |
|          | c3t4d0 | 7.19  | 0.50  | 9     | 150    | 5.27   | 7.68   |
|          | c3t5d0 | 0.40  | 0.50  | 0     | 6      | 3.80   | 9.32   |
| 19:43:10 | c0t6d0 | 0.20  | 0.50  | 0     | 1      | 6.42   | 11.56  |
|          | c0t5d0 | 72.60 | 1.79  | 186   | 2980   | 8.59   | 5.09   |
|          | c0t4d0 | 6.20  | 0.50  | 7     | 115    | 5.48   | 8.48   |
|          | c0t3d0 | 8.60  | 0.50  | 7     | 112    | 5.19   | 13.32  |
|          | c3t6d0 | 6.20  | 0.50  | 27    | 438    | 5.12   | 2.42   |
|          | c4t6d1 | 15.40 | 1.52  | 25    | 394    | 9.70   | 16.09  |
|          | c4t5d0 | 3.60  | 0.50  | 3     | 42     | 4.40   | 14.15  |
|          | c3t4d0 | 5.00  | 0.50  | 6     | 99     | 5.63   | 7.49   |
| 19:43:15 | c0t5d0 | 73.20 | 1.86  | 184   | 2947   | 9.44   | 5.38   |
|          | c0t4d0 | 12.00 | 0.50  | 14    | 221    | 5.05   | 9.26   |
|          | c0t3d0 | 8.60  | 0.50  | 6     | 99     | 5.39   | 12.99  |
|          | c3t6d0 | 6.40  | 0.50  | 27    | 435    | 5.31   | 2.33   |
|          | c4t6d1 | 14.40 | 1.33  | 23    | 365    | 10.21  | 14.72  |
|          | c4t5d0 | 4.20  | 0.50  | 3     | 51     | 5.38   | 13.08  |
|          | c3t4d0 | 9.60  | 0.50  | 13    | 202    | 4.76   | 7.91   |
| Average  | c0t5d0 | 73.95 | 1.86  | 180   | 2879   | 8.93   | 5.46   |
| Average  | c0t4d0 | 9.26  | 0.66  | 11    | 176    | 6.53   | 9.01   |
| Average  | c0t3d0 | 8.46  | 0.50  | 7     | 104    | 5.11   | 12.88  |
| Average  | c3t6d0 | 6.73  | 0.50  | 28    | 440    | 5.21   | 2.63   |
| Average  | c4t6d1 | 15.19 | 1.38  | 25    | 403    | 9.38   | 15.03  |
| Average  | c4t5d0 | 4.66  | 0.50  | 4     | 56     | 4.97   | 13.14  |
| Average  | c3t4d0 | 7.26  | 0.50  | 9     | 150    | 5.12   | 7.74   |
| Average  | c3t5d0 | 0.13  | 0.50  | 0     | 2      | 3.80   | 9.32   |
| Average  | c0t6d0 | 0.07  | 0.50  | 0     | 0      | 6.42   | 11.56  |

---

在输入/输出概况快照脚本被执行的同时，这个 sar 命令被激发。在该命令中，磁盘 c0t5d0 的读/写率为180，这清楚地表明该磁盘上存在大量的输入/输出。sar 命令的输出被详细列出；其中，avwait 是花费在队列中的时间，avserv 为搜索时间与旋转等待时间、磁盘传送时间三者之和。这条信息相当有用。使用这两项技术，你可以鉴别频繁存取的磁盘和文件。

现在寻找使用大量输入/输出资源的用户。在操作系统级，寻找执行大量输入/输出操作的用户很困难；因此，可使用 Oracle 系统表取得你所要的信息。

### 22.3.2 寻找输入/输出密集型用户

你可以使用 V\$SESS\_IO 视图来鉴别资源密集型用户。该视图存储有关会话的输入/输出统计数字的信息。对于每一个会话，在该视图中有一行与之对应。对 V\$SESS\_IO 视图中的列的

解释在表22-4中给出。

表22-4 V\$SESS\_IO视图的列描述

| 列 名                | 描 述                   |
|--------------------|-----------------------|
| SID                | 会话标识符（数据库名）           |
| BLOCK_GETS         | 会话获得的数据块的数量           |
| CONSISTENT_GETS    | 使用一致性获取机制的会话的一致性获取的数量 |
| PHYSICAL_READS     | 此会话的物理读取的数量           |
| BLOCK_CHANGES      | 此会话改变的数据块的数量          |
| CONSISTENT_CHANGES | 此会话的一致性改变的数量          |

像以前的小节中所讲述的那样，你可以在该视图中使用快照技术，将在一个监控期中捕获的开始和结束快照值存储在一个临时表中，然后利用差别来查找执行大量文件存取的用户。使用在PHYSICAL READS列中的差别来查找从磁盘中执行大量数据存取的用户。BLOCK\_GETS列中的差别可以用来鉴别执行大量缓冲区读取的用户。可以开发与22.2节中类似的脚本。

在数据库中引起大量磁盘存取的最常见的事情是磁盘排序（会在临时表空间中引起瓶颈）、全表扫描和频繁地索引读取。

可以使用清单22-17中的脚本鉴别当前正在执行一个排序的会话。

清单22-17 用于鉴别当前正在执行一个排序的会话脚本

```
Select sid
From v$session_wait
Where event = 'db file scattered read'
And p1 = <file id>;
```

在V\$session\_wait中的p1列中包含文件标识符（file id），事件将在该标识符上执行。现在，可以把属于临时表空间的文件的file id用数据库数据文件中的file#值替换。从用于数据库中的临时表空间的dba\_data\_files中选取file#列，然后替换清单22-17中的file id变量。

在发现sid值以后，可以使用V\$SQLTEXT视图和清单22-17中的脚本发现由此会话执行的当前SQL语句。如果在数据库中执行了过多的磁盘排序，考虑增加init.ora的参数SORT\_AREA\_SIZE的值。

你可以使用清单22-18中的脚本鉴别一个当前正在执行全表扫描且正在等待多块读取调用的会话或最后一个等待用于一个多块读取的会话。清单22-18立刻便能识别所有执行全表扫描的会话。

清单22-18 用于查找当前正在执行全表扫描的会话脚本

```
col sql_text format a40

select sid,sql_text
From v$session a, v$sqlarea b
Where a.sql_hash_value = b.hash_value
And sid in
  (Select sid
   From v$session_wait
   Where event like 'db file scattered read')
/
```

The following listing is the output of the above SQL statement.

```

SID          SQL_TEXT
-----
100 select ord_id from cust_ord where crea_u
      ser_id like 'BSMJG%'
102 select emp_id,name from employee where
      salary > 1003;

```

你可以使用V\$SQLAREA视图鉴别正在执行大量物理读取和访问大量缓冲区的 SQL 语句，如清单22-19所示。你可以用一个你认为合适的值替换清单 22-19中的buffer\_gets和disk\_reads。

清单22-19 用于鉴别正在执行过多物理读取或存取大量缓冲区的会话的脚本

```

set pagesize 100
col sql_text format a50

select substr(sql_text,1,200) sql_text, buffer_gets, disk_reads
from v$sqlarea
where buffer_gets > 10000
or disk_reads > 10000000
order by disk_reads desc;

```

The following listing is the output of the above SQL Script.

| SQL_TEXT   | BUFFER_GETS | DISK_READS |
|--|-------------|------------|
| Select RTrim ( cust_ord.brand_id ),RTrim ( ord_id<br>) ,Nvl ( cust_ord.pmt<br>_mthd_id , '12' ) ,Nvl ( card<br>_num , '0' ) ,To_Char ( Nvl ( card_exp_dt ,sysdate<br>) , 'mmyy' ) ,Nvl ( pp_amt ,0.00 ) ,Nvl ( ord_amt , | 20488669    | 13025204   |
| Select count ( * ) From cust_ord ,cmpn_pmt_type Whe<br>re auth_stat = 'Y' and cust_ord.pmt_mthd_id In ( Se<br>lect pmt_mthd_id From pmt_mthd Where pmt_mthd_typ<br>in ( 'R' , 'D' ) ) and cust_ord.cmpn_id =cmpn_pmt_t   | 13136493    | 12952619   |
| SELECT TITLE_ID,CNTRY_ID FROM CUST_ORD_ADR WHER<br>E RTRIM(UPPER(ORD_ID)) = RTRIM(UPPER(:b1)) AND RT<br>RIM(UPPER(ADR_ID)) = RTRIM(UPPER(:b2))   | 10469659    | 10453418   |
| update cust_ord set ord_stat = '07', chng_user_id<br>= 'stat19', chng_dt = sysdate where ord_id in<br>(select c.ord_id from cust_ord a, cust<br>_ord_line b, cust_ord_itm c where a.ord_id                               | 24942696    | 3454225    |

这些SQL语句中的大部分很有可能正在执行全表扫描，从而引发了大量读操作。通过添加辅助性能索引或使用其他的调整技术对 SQL 语句进行优化。

在Windows NT操作系统中，你必须从Windows NT命令提示符执行如下的DISKPERF命令以使磁盘计数器能够收集到有关磁盘读写的统计数字：

```
DISKPERF -YE
```

这会影响到系统开销；这也就是为什么在缺省情况下不使用该计数器的原因。你必须重新启动该系统以便得到逻辑磁盘计数器的正确读数。

DB\_BLOCK\_SIZE参数对磁盘输入/输出有影响。DB\_BLOCK\_SIZE应当等于或成倍于NT簇容量。DB\_BLOCK\_SIZE决不能小于NT簇容量。簇容量是Windows NT一次所能够读出或写入的最小的输入/输出基本单元。例如，如果DB\_BLOCK\_SIZE是4K并且NT簇容量是1K的话，那么每当Oracle需要从磁盘中读取数据时，Windows NT就需要执行4次输入/输出操作。

这是不可接受的。

### 22.3.3 磁盘队列

一个磁盘驱动器一次仅能支持一个请求。当有其他进程等待该磁盘时便会形成磁盘队列。假如该磁盘队列过载的话，你会发现性能出现问题。PerfMon计数器的物理磁盘：磁盘传送量/秒项目让你了解到一定时间中的磁盘传送率。通过把物理磁盘：平均值制成图还能够查看磁盘队列（包括读和写队列）。磁盘队列的长度显示于图 22-3 中。

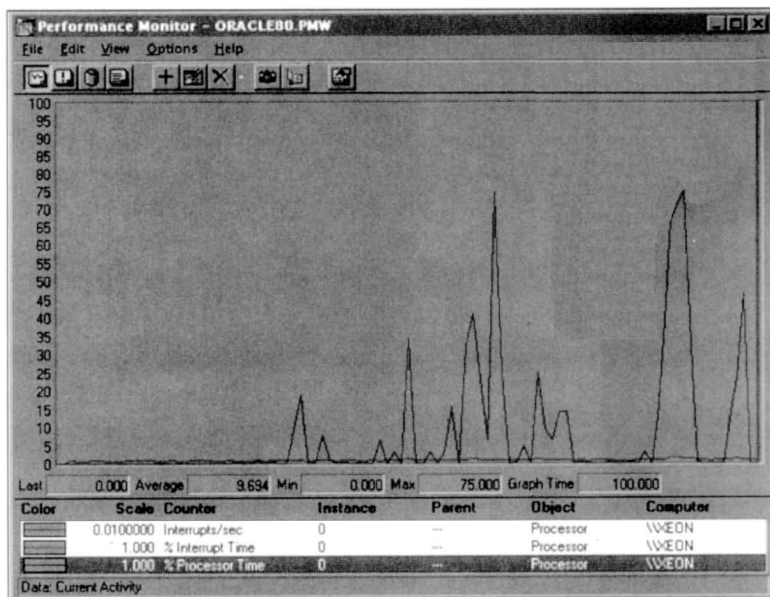


图22-3 监控物理磁盘性能

采用如下预防措施以使输入/输出问题减到最小：

通过使用多个磁盘驱动器和磁盘控制器分散输入/输出装载值。

将分离的、无数据条的磁盘上的 Oracle 联机重做日志与数据文件分开存放。

用更快的磁盘、控制器和输入/输出总线对硬件进行升级，例如，从 IDE 磁盘到 SCSI 磁盘或从 SCSI 磁盘到更快的 SCSI。

## 22.4 内存

UNIX 系统把虚拟内存用于内存管理。使用虚拟内存进程，系统中所有进程的总容量可能超过系统中可用的物理内存。可以使用分页与交换技术——UNIX 系统中的两种内存管理策略来实现虚拟内存。虚拟内存技术使进程的容量大于可用的物理内存数量成为可能。

主存（RAM）存储执行程序所需要的数据。当数据不再被需要用于执行时，它被存储在磁盘上的辅助内存中以便为其他的活动进程腾出空间。整个进程从主存移动到磁盘上的辅助内存区，这个区域被称为交换区（swap space）。

在内存分页中，单个内存页面被传送到交换区，也就是说，整个进程的一部分被传送到交换区。在交换中，整个进程被传送到交换区。内存分页只释放少量的内存，而交换释放大

量的内存。这些内存管理策略所用到的原理是将进程最需要的内存页面保留在物理内存中，其余的内存页面存放在辅助存储区，即交换区中。

当一个进程需要更多的内存时，物理内存中的非活动页面可以被分页出，物理内存中的空间被释放。当空间在物理内存区中被释放时，驻留在辅助内存中的页面便可以被带入物理内存中（RAM）。当UNIX系统把单个内存页面移动到交换区时发生内存分页。因此，仅有一部分进程移动到交换区。另一方面，交换把整个进程移动到交换区中。这样，交换释放了许多物理内存，但是它比内存分页代价更高。一个理想内存状况的关键指标是在完全负载的情况下，系统轻微分页并且根本不发生交换。

在大多数的系统中可以看到如下的情况：当系统有轻微的负载时，没有分页或交换；当系统适度负载时，开始进行分页；当系统负载较重时，由于繁重的内存请求与物理内存的低可用性，分页停止，系统中的交换开始。交换是最昂贵的内存操作。

在一个虚拟内存系统中，系统中的进程所使用的内存地址不是物理内存地址，而是虚拟地址（virtual address）。此虚拟地址由内存管理单元（MMU）转化为物理内存地址。

当进程需要一张内存页面时，它使用虚拟地址引用那一页。此虚拟地址由内存管理单元转化，用来定位实际物理地址。此转化的地址可以映射到物理内存中的一个区域中或者映射到辅助内存（交换区）地址中。

假如物理页面被定位在主存中，该进程可以立即使用它。如果它被定位在磁盘上，内存管理单元会产生页面出错（page fault），然后操作系统把此页面读到物理内存中的一个空闲页面中；这就叫做调入页面（page in）操作。假如内存严重短缺，此页面的读取会引发物理内存中的其他页面被写入到磁盘中。这通常被称为调出页面（page out）。在页面被操作系统读入后，新的页面地址被传送到内存管理单元，然后操作系统重新启动指令。分页仅把物理内存页面移动到磁盘中；交换会把整个进程移动到磁盘中。假如系统中内存严重短缺，那么会非常频繁地发生交换。

无论是调出页面还是调入页面对于系统性能来说都是很昂贵的，因为与可直接在高速缓存中使用的数据相比，磁盘读/写非常昂贵。因此，假如进程由于在物理内存缺乏的系统中频繁申请新的数据页而产生大量的页面错误的话，进程的运行速度会明显慢下来。

在使用虚拟内存的系统中，只有由进程使用的那部分内存驻留在物理内存中。这部分内存被叫做该进程的工作组（working set）。因为在任何给定的时刻上，进程不会使用它的全部页面，因此把当前使用的页面保存在物理内存中是更为经济的做法。在进程运行期间，该进程有可能需要当前不在物理内存中的页面。这会引发页面出错，但是在进程运行期间出现一些页面出错问题是极为普通的，因为把进程需要的所有内存页面都放入主存中明显是不切合实际的。操作系统总是试图把工作组的全部页面保存在主存中。使用内存的这项技术可以使一个系统的内存吞吐量达到最大值，记住，系统中有许多并行进程正在运行，这些并行进程主动生成对内存的并行请求。

进程的实际物理内存使用决不是恒定不变的，由于它的易变性，要确定物理内存的使用非常困难。另一方面，进程对虚拟内存的需求可以很容易地被确定。一个进程所消耗的物理内存的数量基于许多因素，例如系统负载、进程正在做的工作等等。例如，假如一个进程正在一个较重负载的系统中运行，因为有许多需要物理内存页面的其他进程，所以物理内存中实际分配给该进程的页面的数量将非常接近于该进程的工作组的容量。换句话说，如果同一

个进程在一个较轻负载的系统中运行，该系统中正在运行的并行活动进程并不多，那么实际分配给该进程的物理页面的数量会大大超过该进程需要的工作组容量。类似地，在一个进程的运行过程中，如果该进程的工作组改变了，那么分配的物理内存的数量也会随之改变。所有这些使得很难预测消耗大量物理内存的程序，因为内存分配完全是随形势的发展而变化的。要模拟进程对物理内存的需求，你必须还要模拟系统中的负载，因为一个孤立运行在系统中的进程对物理内存的使用要大于运行在峰值负载上的进程对内存的使用。记住如下这些要点：

要监控系统中的进程对内存的使用，你必须把系统负载考虑在内。一个进程在较轻负载的系统中独占大量的物理内存而在一个较重负载的系统中占用少量内存是极为普通的事情。

要检查物理内存是否足够用于系统中运行的所有进程，最好的方法是做如下测试：

1) 将你的系统加载到峰值负载。

2) 检查整个内存工作情况（在下一节中讨论）以查看是否存在任何分页或交换的迹象。忽略少量的分页，因为当进程的工作组改变时，进程分页是很自然的事情。假如有过多的分页或交换发生，或许表明系统中可用的物理内存数量较少，你必须重新安排你的某些内存密集型进程的执行时间。

在你对监控内存的使用进行研究以前，分析一下单个进程的内存使用的分类。

#### 22.4.1 进程内存分类

UNIX系统中的每一个进程被分为两个大的领域：文本页与数据页。文本（text）段包含进程执行的机器指令。数据（data）段包含进程在执行期间所需要的所有其他信息。文本页被标记为只读，因为它们包含机器指令并且从文件系统中调出。执行同一个程序的进程将共享相同的文本页，这样便优化了内存使用。例如，假如有 10 个用户工作在 Oracle Form 上，所有这些人共享 Oracle Form 的可执行文件或文本页。

另一方面，数据页被标记为读/写。它们专用于某进程并且从分页区（交换区）中调出；也就是说，每一个进程有它自己专用的数据页，当进程数量增加时，对专用页的需求也随之增加——这与共享文本页的情况不一样。主要的区别是假如一个文本页由于内存短缺而被调出，该文本页当前占用的物理内存页能够直接被新的信息改写；但是如果一个数据页被改写或调出，那么该页必须在物理页开始改写以前写入或拷贝到磁盘中。这其中的理由很简单，因为文本页决不会被修改，文本页的映象总能够从文件系统中获得。

一个UNIX进程实际上被划分为 6 个区域：文本、堆栈、堆、BSS、初始化数据和共享内存。

如前所述的一样，文本在进程之间共享并且从文件系统中调出。它包含机器指令并且被标记为只读。

堆栈（stack）对进程是专用的，它从交换区中调出，包含运行期执行堆栈。除了函数和过程参数以外，运行期堆栈还包括函数和过程活动记录。堆栈页被标记为读/写。堆栈的容量可以增长。

当堆（heap）运行时，它包含了分配给进程的数据页。堆是一个专用的进程区并且它的容量可以在程序运行期间增长得非常大。堆页被标记为读/写，这些页从交换区中调出。

BSS段对进程也是专用的。它被标记为读/写，它的容量在进程执行期间保持不变，BSS

段的页面被标记为读/写。该段的容量在进程执行期间保持固定。该段用于存储静态地分配的未初始化数据，这些页从交换区中调出。

初始化数据（initialized data）段对进程是专用的。它被标记为读/写，从交换区中调出，在进程执行期间其容量保持不变。

共享内存（shared memory）段是在进程之间共享的页面。它们被标记为读/写。使用系统调用shmget（）分配共享页面。进程通过使用系统调用shmat（）共享页面。共享页面从交换区中调出，共享内存段的容量在分配完成后便保持固定。

## 22.4.2 内存预览

在你深入研究使用内存的个别进程以前，必须首先对系统有一个大概的了解。当对系统中的内存进行预览时，要寻找的关键东西是系统中过多的分页与交换。假如系统中的内存比较缺乏，那么当进程被分配了CPU的一个时间片后，该进程把这段CPU时间用于交换与分页，不做急剧影响系统性能的其他事情。系统中频繁地分页与交换活动严重地妨碍了系统性能，必须通过增加额外的物理内存或重新安排进程的方式来尽可能地减小。

在UNIX系统中，vmstat与sar是用来对系统内存的使用进行监控的工具。这两个工具的缺陷是它们仅能在总体水平上实现对内存的监控，不能给出每一个进程的内存使用细节。清单22-20所显示的是vmstat命令的输出。

清单22-20 用于监控内存使用的vmstat命令的输出

```
vmstat -S 2 10
procs      memory      page      faults      cpu
r  b  w    avm free    si so    pi po    fr de    sr  in  sy    cs us sy id
1 111 0    5152 2885    0  0    0  0    0  0    0 1028 3068 1121 19  8 73
1 111 0    5152 2885    0  0    0  0    0  0    0 1004 2778 1074 17  5 78
2 117 0    4986 2885    0  0    0  0    0  0    0 1022 2666 1097 18 10 72
2 117 0    4986 2885    0  0    0  0    0  0    0 1008 2618 1129 22  7 71
0 111 0    5644 2842    0  0    0  0    0  0    0  994 2742 1079 27 10 64
0 111 0    5644 2841    0  0    0  0    0  0    0 1016 2974 1210 24 14 63
0 111 0    5644 2927    0  0    0  0    0  0    0 1029 4388 1438 21  9 70
0 109 0    5816 2927    0  0    0  0    0  0    0  997 3656 1359 12  8 81
0 109 0    5816 2927    0  0    0  0    0  0    0  953 3135 1255 16  5 79
```

表22-5中列举了Vmstat命令输出中的主要的列。

正如你看到的那样，vmstat命令非常全面，从内存使用、进程状态到CPU使用百分数等各方面几乎给了系统一个完整的概述。

在此输出中，就内存使用而言，要查找的关键项目是po、so、b与w列。

假如po（调出页面）与so（换出）值很高，表明系统中内存严重短缺。在清单22-20中，系统中没有页面调出。系统中大约有 $2885 \times 4\text{KB}$ （4KB是内存页面的容量），即11.26MB的可用空闲内存。

注意 你可以使用由vmstat命令产生的po与so输出确定分页与交换活动。应在一天中定期运行vmstat命令以便检查系统是否经常分页或交换。

假如你查看前面的输出，会发现b列显示了大量的阻塞进程。我通过使用sar-d命令做了进一步的研究，发现大量的阻塞进程是因为少量磁盘上正在进行繁忙的输入/输出。由于输入/输出正在等待被执行，所以造成进程被阻塞。假如vmstat输出显示了大量的分页或交换，你必

须鉴别内存密集型用户。

表22-5 vmstat输出中的主要列

| 列 名    | 描 述               |
|--------|-------------------|
| procs  | 在如下显示的各种状态中的进程数   |
| r      | 运行模式下的进程数         |
| b      | 阻塞模式下的进程数         |
| w      | 已交换的和正在等待进程资源的进程数 |
| memory | 实内存与虚拟内存的报告       |
| avm    | 平均活动内存            |
| free   | 页面中自由表的容量         |
| page   | 在时间间隔中的页面出错报告     |
| si     | 被换入的页面数           |
| so     | 被换出的页面数           |
| pi     | 调进页面的千字节数（检验）     |
| po     | 调出页面的千字节数         |
| cpu    | CPU使用情况的报告        |
| us     | 服务于用户请求所用的时间百分数   |
| sy     | 服务于系统调用所用的时间百分数   |
| id     | 空闲时间              |

### 22.4.3 寻找内存密集型用户

需要技巧才能在UNIX系统中监控内存的个别使用。大部分工具（例如 vmstat和sar）报告系统内存使用情况，但不报告单个进程的内存使用情况。实际物理内存的使用不容易获取的另一个原因是每一个进程被划分为文本与数据，文本页在进程之间共享，而数据页专用于进程。假如进程被依附于一个共享内存段，那么此偏差量必须被减去以便获得实际的专用内存（由进程使用的内存）。当文本页被共享时，会在所有进程的虚拟内存使用中报告。因此，假如你增加使用系统中所有进程的虚拟内存，报告中获得的数字将超过所有进程实际使用的虚拟内存量。

尽管如此，你可以假设当对Oracle系统中的内存进行监控时，所有的进程共享相同的文本页（也就是Oracle可执行文件）。假如所有的进程正在使用相同的可执行文件，那么所有进程的文本内存用量是相同的。剩余的内存是由进程使用的专用内存。在UNIX系统中，带有-l选项的ps命令可以用来确定每一个单独进程的内存使用情况。-l选项在ps输出的sz列中报告了内存使用情况。清单22-21中的ps命令可以用来监控系统Oracle进程（前台和后台）的内存使用情况。sz列报告了进程核心映像的物理页面的容量。它包括文本、数据和堆栈空间。清单22-21中的输出按照sz列值由高到低的顺序显示排序后的ps命令，其中sz列是输出中的第7列。此输出对所有Oracle进程给出了使用的文本+数据+堆栈的容量。

**注意** 以前提到的sz列代表文本加数据再加堆栈的内存使用。Oracle文本页的容量可以在Oracle可执行文件中使用测量容量的命令来确定。在获得文本页的容量之后，理论上讲，假如你从虚拟内存容量中减去使用ps -ael命令获得的数值，你将得到每个进程的实际虚拟内存使用情况。然而，有时ps -el命令中的sz列不能报告实际的文本页容量；它仅报告进程需要的页面。这是由于有页面请求算法，在该算法中仅仅最需要的文本页才被进程使用，剩余的页面仅在需要时才从文件系统中取出。

你可以使用在top命令中的容量列，因为当top命令在SIZE列中就虚拟内存提出报告时，该命令报告的是总的文本页容量，而不是实际使用的页容量。在清单 22-21中，由进程25623号使用的虚拟内存的容量是8640K，这包括了整个文本页。

清单22-21 使用top命令输出获得虚拟内存容量

```
System: arms1                               Tue Sep  9 19:41:51 1997
Load averages: 4.87, 4.74, 4.44
211 processes: 204 sleeping, 7 running
Cpu states:
CPU  LOAD   USER    NICE    SYS    IDLE   BLOCK  SWAIT   INTR   SSYS
0    4.98  88.1%   0.0%   11.9%   0.0%   0.0%   0.0%   0.0%   0.0%
1    4.75  79.2%   0.0%   20.8%   0.0%   0.0%   0.0%   0.0%   0.0%
-----
avg  4.87  83.2%   0.0%   16.8%   0.0%   0.0%   0.0%   0.0%   0.0%
Memory: 53592K (15892K) real, 72336K (25548K) virtual, 11848K free Page# 1/20
CPU  TTY  PID USERNAME  PRI NI  SIZE  RES  STATE  TIME %WCPU %CPU COMMAND
0    ? 25623 arms      240 20 8640K 548K run   7:34 43.47 43.39 oracleor
1    ? 18128 oracle    241 20 9184K 1024K run  177:07 31.29 31.23 oracleor
0    ? 19523 oracle    240 20 9644K 1552K run   60:48 26.19 26.15
oracleor
```

注意 如果试图使用ps命令寻找容量值，会得到下述结果：

```
$ ps -ael | grep 256,23
1 R    105 25623 25622 254 241 20 3909b00 1524
    9:20 oracle
```

这个ps命令显示虚拟内存的容量是1524 × 4K=6096K，小于top命令的8640K，这是因为实际使用的文本页容量低于总文本页容量。

当计算单个进程的实际内存使用情况时，请记住上面所说的这一点。

清单22-22中的输出确定了Oracle进程的pid。当在以前的小节中使用SQL语句获得pid时可以发现进程的sid，并且在会话层可以完成进一步的调查。这将在下面的输出中进行说明。

清单22-22 用来查找进程内存使用情况的ps命令

```
$ ps -ael | grep oracle | sort -n -r -k 10
1 R    129 28060 28058 248 240 20 3932f00 2400 - ? 15:24 oracle
1 R    104 28049 1 246 239 20 3941080 2001 - ? 10:55 oracle
1 S    104 25751 1 0 156 20 38e1f80 1939 461b34 ? 19:02 oracle
1 S    104 15018 1 0 154 20 37c2980 1915 3d90e38 ? 198:25 oracle
1 S    104 25861 1 59 148 20 3807d00 1711 2194bac ? 32:42 oracle
1 S    104 25743 1 0 156 20 3e38380 1658 461b1c ? 23:52 oracle
1 S    104 14103 1 0 154 20 2829900 1653 3d4e338 ? 109:12 oracle
1 R    104 25739 1 255 241 20 37c2d80 1623 - ? 142:21 oracle
1 S    104 27772 1 0 156 20 372ba80 1603 461b04 ? 9:11 oracle
1 S    105 22744 22743 0 154 20 3941e80 1578 3d61da2 ? 38:59 oracle
1 S    129 25731 25727 0 154 20 335f700 1538 38c47a2 ? 0:00 oracle
1 S    104 19722 1 0 154 20 2845380 1525 3d6cb38 ? 0:05 oracle
1 S    129 28055 28053 0 154 20 3780f00 1522 3dda0a2 ? 0:00 oracle
1 S    125 25629 1 0 156 20 3738680 1511 461aa4 ? 0:01 oracle
1 S    104 7258 1 0 154 20 38e6100 1498 391e838 ? 0:00 oracle
1 S    125 25623 1 0 156 20 294f600 1493 461a8c ? 0:38 oracle
1 S    125 25621 1 0 156 20 2576080 1490 461a84 ? 1:15 oracle
1 S    125 25627 1 0 156 20 3807300 1483 461a9c ? 0:03 oracle
1 S    125 25625 1 3 156 20 3937000 1483 461a94 ? 2:00 oracle
1 S    125 25649 1 0 156 20 2958980 1478 461af4 ? 0:10 oracle
1 S    125 25647 1 0 156 20 3903b00 1478 461aec ? 0:10 oracle
1 S    125 25645 1 0 156 20 2845a80 1478 461ae4 ? 0:11 oracle
```

|     |     |       |   |   |     |    |         |      |          |      |        |
|-----|-----|-------|---|---|-----|----|---------|------|----------|------|--------|
| 1 S | 125 | 25643 | 1 | 0 | 156 | 20 | 3354400 | 1478 | 461adc ? | 0:13 | oracle |
| 1 S | 125 | 25641 | 1 | 0 | 156 | 20 | 391d700 | 1478 | 461ad4 ? | 0:13 | oracle |
| 1 S | 125 | 25639 | 1 | 0 | 156 | 20 | 3739b80 | 1478 | 461acc ? | 0:13 | oracle |
| 1 S | 125 | 25637 | 1 | 0 | 156 | 20 | 2852780 | 1478 | 461ac4 ? | 0:14 | oracle |
| 1 S | 125 | 25635 | 1 | 0 | 156 | 20 | 2888580 | 1478 | 461abc ? | 0:15 | oracle |
| 1 S | 125 | 25633 | 1 | 0 | 156 | 20 | 3663e80 | 1478 | 461ab4 ? | 0:15 | oracle |
| 1 S | 125 | 25631 | 1 | 0 | 156 | 20 | 3e40c80 | 1478 | 461aac ? | 0:15 | oracle |
| 1 S | 125 | 25619 | 1 | 0 | 156 | 20 | 27fb500 | 1477 | 461a7c ? | 0:01 | oracle |

单个进程对内存的使用情况也可以通过使用 V\$SESSTAT 系统视图得到监控。要了解你正在查看的那部分内存，需要研究内存如何在 Oracle SGA 中组织起来。SGA 是一个包括共享池、块缓冲区和重做日志缓冲区的共享段。共享池是 SGA 中的一个区域，它包含了诸如共享 SQL 区与数据字典缓存的构造。

对于正在服务器上执行的每条 SQL 语句来说，有一个共享部分（共享 SQL 区）和一个专用部分（专用 SQL 区）。执行同一段 SQL 代码的两个用户使用相同的共享 SQL 区，但是每个用户有他们各自的专用 SQL 区。

共享 SQL 区包含语法分析树和每条 SQL 语句的执行计划。该区域的大小取决于语句的复杂性。每一个这样的 SQL 区的容量可以用 V\$SQLAREA 视图的 sharable\_memory 列来确定，V\$SQLAREA 视图包含了共享池中的所有 SQL 区。每一个执行相同 SQL 语句的会话都共用相同的共享 SQL 区。

专用 SQL 区是一个内存区域，它包含了诸如绑定信息与运行期缓冲区的数据。每一个执行 SQL 语句的会话都有它自己的专用 SQL 区，该区域用于它所执行的每条 SQL 语句。概括地说，对于正在执行同一条 SQL 语句的多个会话来说，有一个共享的 SQL 区并且每个会话各自拥有的多个专用 SQL 区。专用 SQL 区进一步划分为一个持久区和一个运行时间区。

你需要监控内存使用的那部分内存区称为程序全局区（PGA）。程序全局区包含单个进程的数据和控制信息。程序全局区还被称为进程全局区（Process Global Area）。程序全局区的内容取决于连接类型是专用的还是多线程的。

在两种体系结构中，程序全局区总是包含堆栈空间（stack space），它是用来存储会话变量、数组和其他信息的内存。

在专用服务器选项（dedicated server option）中，程序全局区存储与用户的会话有关的辅助信息；也就是说，在这里存储专用 SQL 区和其他的会话信息。在多线程选项（multithreaded option）中，该信息存放在 SGA 中。

你能够监控程序全局区的使用情况并估计系统中每个用户对内存的需求。可以使用 V\$SESSTAT 视图了解对 PGA 内存的使用。statistic# value=20 对应于程序全局区内存使用。使用清单 22-23 中的脚本对内存密集型用户进行监控。

清单 22-23 使用 V\$SESSTAT 查看程序全局区内存的使用情况

```
col value format 99999 heading "Memory |Kb"
```

```
Select sid,
       value/1024 value
  From v$sesstat
 Where statistic# =20
 Order By value desc
```

The following is the output of the above SQL statement.

| Memory<br>SID | Kb   |
|---------------|------|
| 31            | 1165 |
| 153           | 754  |
| 142           | 733  |
| 101           | 686  |
| 70            | 362  |
| 73            | 336  |
| 26            | 306  |
| 60            | 298  |
| 82            | 297  |
| 53            | 297  |
| 123           | 275  |
| 44            | 272  |
| 110           | 270  |

在确定了消耗大量程序全局区的会话后，你可以继续前面小节中提到的步骤以便发现每个会话正在做的工作。

Windows NT对内存的寻址能力最大可以达到 4GB；其中2GB保留用于操作系统，剩余的2GB用于其他的应用程序。

为了在你的 Windows NT服务器中查找可用的内存，打开 Windows NT的资源管理器并选择关于 Windows NT的帮助。图22-4所示的窗口出现在你的屏幕上，其中以千字节为单位显示了可用的内存容量。

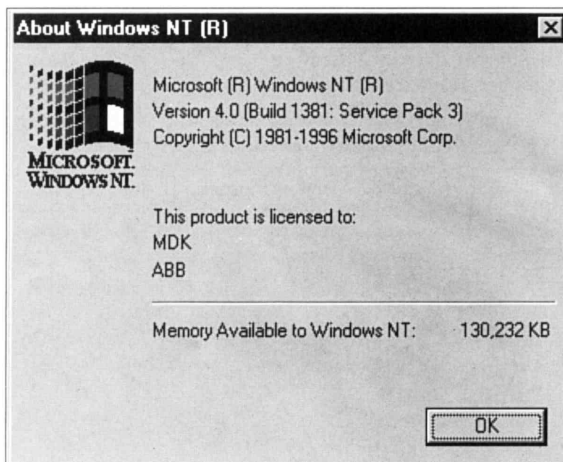


图22-4 本系统中可用的内存容量

#### 22.4.4 缓冲区高速缓存

最重要的内存（SGA）因素是调整数据库缓冲区高速缓存以使用户能够尽可能地从内存中读信息，而不是从磁盘中读信息。假如有效地调整缓冲区高速缓存，那么就可以保证数据库不会受到输入/输出的限制并避免由输入/输出限制所导致的速度变慢。你可以查询V\$SYSSTAT表或使用Oracle 8性能监控程序得到有关缓冲区高速缓存使用的实时性能信息。Oracle 8缓冲区高速缓存的物理读/获取计数器显示了缓存遗漏率，它显示了涉及物理磁盘活动的数据库读取率。

图22-5中显示的当前遗漏率是1.9%。它不应超过20%，因为你应总是使缓存命中率保持

在80%以上。

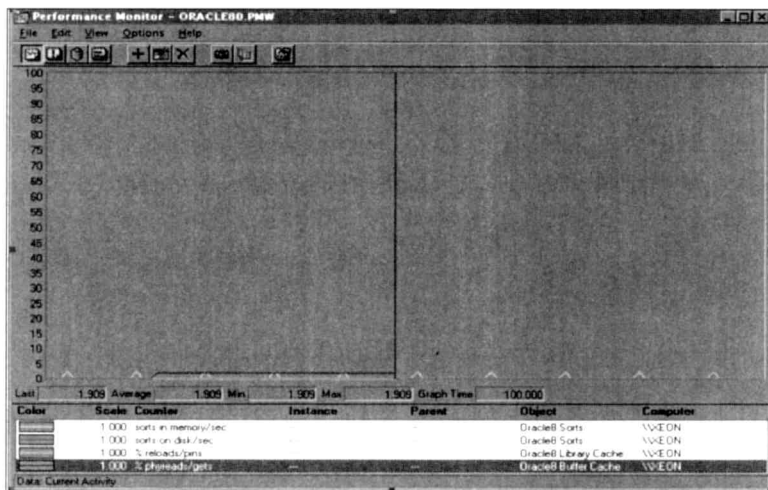


图22-5 监控缓存遗漏率

你可以执行如下查询以得到有关缓冲区高速缓存使用的实时性能信息：

```
select trunc((1-(sum (decode (name, 'physical reads',value,0))/
(sum(decode(name,'db block gets',value,0)) +
(sum(decode(name,'consistent gets',value,0)))))
)*100) "Buffer Hit Ratio"
from v$sysstat;
```

#### 22.4.5 共享池

共享池由库缓存和数据字典缓存组成。

##### 1. 库缓存

库缓存包含所有新近执行过的 SQL 语句，例如存储的过程、函数、包、触发器以及 PL/SQL 块。

执行如下查询，得到命中率的价值来确定库缓存的性能：

```
select sum ( pinhits - reloads )/sum ( pins ) "HIT RATIO",
       sum (reloads) / sum (pins) "RELOAD PERCENT"
from v$librarycache;
```

以上 SQL 语句的输出如下：

```
HIT RATIO  RELOAD PERCENT
-----
.99729584  .00005962
```

前面例子中 99.729% 的命中率和 0.0059% 的重新加载率（遗漏）是可以接受的。你应设法使命中率保持在 85% 以上，而重新加载率保持在 2% 或 2% 以下。库缓存重新加载率可以在 Oracle 8 PerfMon 中使用计数器 Oracle 8 库缓存：%重新加载/针查看。

##### 2. 数据字典缓存

数据字典缓存是有效数据库操作的中心。V\$ROWCACHE 表包含有关数据字典缓存的信息。如果没有数据字典缓存，每次对查询进行语法分析时，Oracle 就必须从磁盘中得到元数据信息。

执行下列查询以确定缓存遗漏率：

```
select sum(gets) "Dict. Gets",  
       sum(getmisses) "Get Misses",  
       sum(getmisses)/sum(gets)*100 "Ratio (Ideal <15%)"  
from v$rowcache;
```

以上SQL语句的输出如下：

```
Dict. Gets Get Misses Ratio (Ideal <15%)  
-----  
204324      960      .46984202
```

对于优化性能而言，字典缓存遗漏率应当低于 15%。你还可以使用 Oracle 8 PerfMon 计数器 Oracle 8 数据字典缓存：% 遗漏/获取得到相同信息。

#### 22.4.6 递归调用

只要表、索引或回滚段需要更多的数据库空间时就会出现递归调用。递归调用影响数据字典缓存的性能。当数据库对象没有更多的可用空间并且引发动态扩展时，Oracle 必须暗中执行递归调用以使用新区间的信息更新数据字典。

执行如下查询以确定递归调用的数量：

```
select * from v$sysstat  
where name = 'recursive calls'  
/
```

你还可以使用 Oracle 8 PerfMon 并将计数器 Oracle 8 动态扩展：递归调用/秒用图表示，如图 22-6 所示。

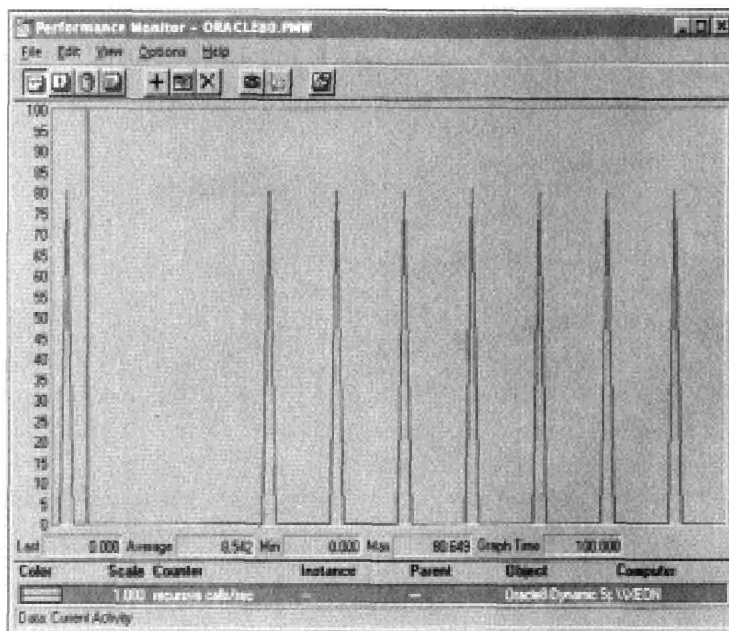


图22-6 监控数据库的递归调用

#### 22.4.7 分页

在内存中，页面是 Windows NT 一次可以使用的输入/输出的最小值。当一个应用程序需要更多的内存时，Windows NT 在内存中寻找当前没有被使用的页面。然后执行页面调出，将非

活动页面移动到磁盘上一个特别指定的文件中，以便腾出空间给更多的活动页面。假如再次需要页面文件中的一个或多个页面，Windows NT执行页面调入，将这些页面从页面文件中移回内存中。基于Intel的NT系统的缺省页面容量是4K，基于Alpha的NT系统的缺省页面容量是8K。

你可以通过在Windows NT PerfMon中查看计数器内存：页面输入/秒和内存：页面输出/秒来观察页面调入和页面调出的情况，如图22-7所示。

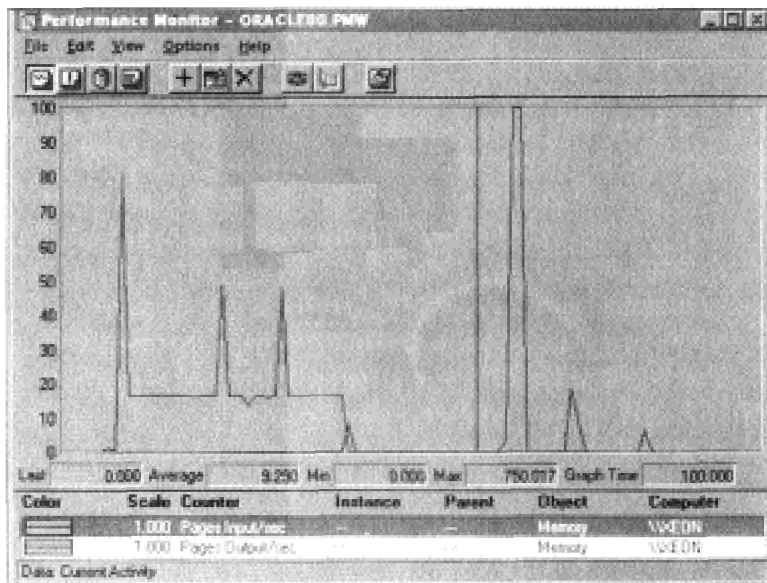


图22-7 监控Windows NT分页活动

Oracle 8初始化参数PRE\_PAGE\_SGA可以尽可能地把SGA保留在内存中。当PRE\_PAGE\_SGA被设置为YES，Windows NT尽最大努力把构成SGA的页面保存在物理内存中，并避免把SGA页面写入分页文件中。Windows NT分页文件PAGEFILE.SYS缺省情况下驻留在C：驱动器上。页面文件的总容量至少同物理内存一样大。

#### 22.4.8 分页错误

当一个程序运行时，它所使用的物理内存中的页面总数被称为该进程的工作组（working set）。假如一个进程需要对当前不在它的工作组中的页面进行存取时，就会发生页面错误。这可以是一个软件错误（所需的页面在内存中的其他地方发现）或硬件错误（所需的页面在磁盘上并且必须被读入内存中）。

使用Windows NT的任务管理器可以快速地鉴别产生大量页面错误的进程，如图22-8所示。

下面的一些措施用于解决过多的分页问题：

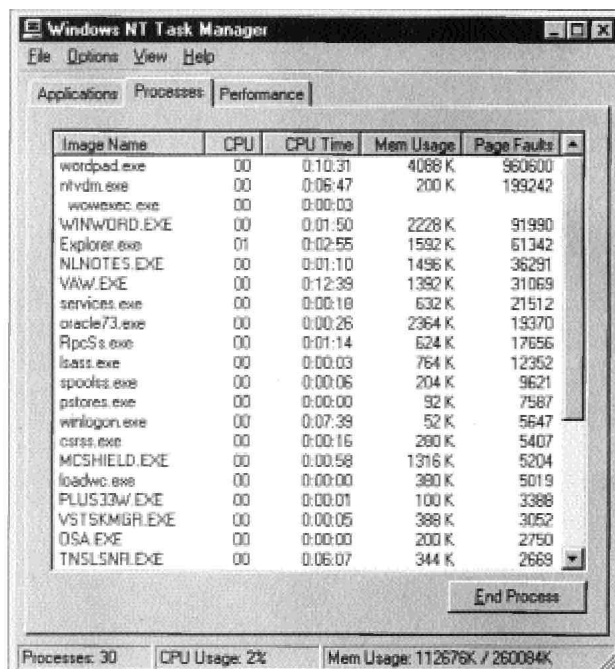
- 使Windows NT分页文件更大。

- 停止不必要的Windows NT服务。

- 确认整个SGA能够装入物理内存。

- 删除不再使用的网络协议。

- 增加内存容量。



The screenshot shows the Windows NT Task Manager window with the 'Performance' tab selected. The 'Processes' list is displayed, showing various running processes with their CPU usage, CPU time, memory usage, and page faults. The status bar at the bottom indicates 30 processes, 2% CPU usage, and 112676K / 260084K memory usage.

| Image Name   | CPU | CPU Time | Mem Usage | Page Faults |
|--------------|-----|----------|-----------|-------------|
| wordpad.exe  | 00  | 0:10:31  | 4088 K    | 960600      |
| ntvdm.exe    | 00  | 0:06:47  | 200 K     | 199242      |
| wwwexec.exe  | 00  | 0:00:03  |           |             |
| WINWORD.EXE  | 00  | 0:01:50  | 2228 K    | 91990       |
| Explorer.exe | 01  | 0:02:55  | 1592 K    | 61342       |
| NLNOTES.EXE  | 00  | 0:01:10  | 1496 K    | 36291       |
| VAWV.EXE     | 00  | 0:12:39  | 1392 K    | 31069       |
| services.exe | 00  | 0:00:10  | 632 K     | 21512       |
| oracle73.exe | 00  | 0:00:26  | 2364 K    | 19370       |
| RpcSs.exe    | 00  | 0:01:14  | 624 K     | 17656       |
| lsass.exe    | 00  | 0:00:03  | 764 K     | 12352       |
| spoolss.exe  | 00  | 0:00:06  | 204 K     | 9621        |
| psstores.exe | 00  | 0:00:00  | 92 K      | 7587        |
| winlogon.exe | 00  | 0:07:39  | 52 K      | 5647        |
| csrss.exe    | 00  | 0:00:16  | 280 K     | 5407        |
| MCSHIELD.EXE | 00  | 0:00:58  | 1316 K    | 5204        |
| loadwc.exe   | 00  | 0:00:00  | 380 K     | 5019        |
| PLUS32WV.EXE | 00  | 0:00:01  | 100 K     | 3388        |
| VSTSKMGR.EXE | 00  | 0:00:05  | 388 K     | 3052        |
| OSA.EXE      | 00  | 0:00:00  | 200 K     | 2750        |
| TNSLSNR.EXE  | 00  | 0:06:07  | 344 K     | 2669        |

Processes: 30    CPU Usage: 2%    Mem Usage: 112676K / 260084K

图22-8 使用任务管理器识别页面错误