《Linux 系统和网络性能监测》

读书笔记

1.0 性能监测简介

性能调优是找出系统瓶颈并消除这些瓶颈的过程。很多系统管理员认为性能调优仅仅是调整一下内核的参数即可解决问题，事实上情况并不是这样。性能调优是实现操作系统的各个子系统之间的平衡性，这些子系统包括：

*** CPU**
*** Memory**
*** IO**
*** Network**

子系统之间相互依存，任何一个子系统的负载过度都能导致其他子系统出现问题，例如：

*** 大量的 page-in IO 请求可能导致内存队列被塞满**
*** 网卡的巨量吞吐可能导致 CPU 资源耗尽**
*** 系统尝试保持释放内存队列时可能耗尽 CPU 资源**
*** 来自内存的大量磁盘写入请求可能导致 CPU 资源和 IO 通道耗尽**

性能调优的前提是找出系统瓶颈之所在，尽管问题看似由某个子系统所导致，然而这很可能是另外一个子系统的过载所引起的。

1.1 判定应用的类型

为了明白从何处开始着手调整性能瓶颈，弄清被分析系统的性能表现是首要任务。任何系统的应用常可分为以下两类：

*** IO 限制型——一个 IO 限制型的应用需要大量的内存和基础存储设备占用。因其需要大量的数据读写请求，此类应用对 CPU 和网络需求不高（除非存储系统在网络上）。IO 限制型应用使用 CPU 资源来进行 IO 操作且常进入睡眠状态。数据库应用常被认为属于此类。**

*** CPU 限制型——一个 CPU 限制型应用需要大量的 CPU 资源，来进行批量的处理或大量的计算。大容量 web 服务，mail 服务，以及任何类型的渲染服务都被归到此类。**

1.2 判定基准信息

系统的利用率因管理员的期望值和系统的参数值而异，判断一个系统是否有性能问题的唯一途径是弄清楚对系统的期望是神马，需求的性能是神马，应该得到的数据是神马？而为了建立这些信息的唯一途径是为系统建立一个基准。在性能可接受的状态下必须为系统建立统计信息，这样就可以在性能不可接受时进行对比。

在下面的例子中，将对两种状态下的统计信息进行对比：

```
# vmstat 1
procs                  memory      swap      io     system         cpu
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy wa id
 1  0 138592  17932 126272 214244    0    0     1    18  109   19  2  1  1 96
 0  0 138592  17932 126272 214244    0    0     0     0  105   46  0  1  0 99
 0  0 138592  17932 126272 214244    0    0     0     0  198   62 40 14  0 45
```

```
 0    0 138592   17932 126272 214244    0    0     0     0  117   49  0  0  0 100
 0    0 138592   17924 126272 214244    0    0     0   176  220  938  3  4 13 80
 0    0 138592   17924 126272 214244    0    0     0     0  358 1522  8 17  0 75
 1    0 138592   17924 126272 214244    0    0     0     0  368 1447  4 24  0 72
 0    0 138592   17924 126272 214244    0    0     0     0  352 1277  9 12  0 79
```

```
# vmstat 1
procs                  memory      swap      io     system      cpu
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy wa id
 2  0 145940  17752 118600 215592    0    1     1    18  109   19  2  1  1 96
 2  0 145940  15856 118604 215652    0    0     0   468  789  108 86 14  0  0
 3  0 146208  13884 118600 214640    0  360     0   360  498   71 91  9  0  0
 2  0 146388  13764 118600 213788    0  340     0   340  672   41 87 13  0  0
 2  0 147092  13788 118600 212452    0  740     0  1324  620   61 92  8  0  0
 2  0 147360  13848 118600 211580    0  720     0   720  690   41 96  4  0  0
 2  0 147912  13744 118192 210592    0  720     0   720  605   44 95  5  0  0
 2  0 148452  13900 118192 209260    0  372     0   372  639   45 81 19  0  0
 2  0 149132  13692 117824 208412    0  372     0   372  457   47 90 10  0  0
```

只需要看代表 idle 时间的最后一列（id）就能发现问题，在基准数据中 CPU 空闲时间在 79%-100%之间，在高负荷状态下系统利用率 100%且无空闲。需要考虑的是 CPU 这块的问题。

2.0 安装监测工具

大多*nix 系统均自带了很多标准监测命令，也有的是第三方工具：

工具名称	描述	基础安装包	可选安装包
vmstat	多功能	Y	Y
mpstat	CPU 性能	N	Y
sar	多功能	N	Y
iostat	磁盘性能	N	Y
netstat	网络性能	Y	Y
dstat	多功能	N	大多数
iptraf	流量监测	N	Y
netperf	网络带宽	N	大多数
ethtool	网卡监测	Y	Y
iperf	网络带宽	N	Y
tcptrace	数据包监测	N	Y
iotop	IO 监测	N	Y

3.0 CPU 介绍

CPU 利用率很大部分取决于试图访问它的资源，内核拥有一个管理两种资源的调度器：线程（单或多）和中断。调度器给予不同资源以不同的优先级，以下由优先级从高到低：

*** 中断——设备通知内核它们处理完成。例如网卡发送一个数据包或硬盘驱动器提供一次 IO 请求**

*** 内核（系统）进程——所有的内核进程都在此级别的优先级进行处理**

*** 用户进程——通常被称为“用户空间”，所有应用软件运行在用户空间，拥有最低的优先级**

为了弄明白内核是如何管理不同的资源的，几个关键概念需要提及一下：context switches，run queues，utilization。

3.1 Context Switches(上下文切换)

大多数处理器在同一时间只能处理一个进程或线程，多线程处理器可同时处理 n 个线程。然而，linux 内核把多核处理器的各个核心当做独立核心。例如，内核把一个双核的处理当做两个独立处理器。

一个标准的内核可同时处理 50 到 50000 个进程，在只有一颗 CPU 的情况下，内核必须调度和平衡这些进程和线程。每个线程在处理器上都拥有一个时间分配单元，当一个线程超过自己的时间单元或被更高优先级的程序抢占时，此线程及被传回队列而此时更高优先级的程序将在处理器上执行。这种线程间的切换操作即是上下文切换。

3.2 运行队列

每个 CPU 维持着一个线程的运行队列，理论上，调度器应该是不断地运行和执行线程。线程要么处于睡眠状态，要么处于可运行状态。假如 CPU 子系统处于高负载状态，那么内核调度器罢工是有可能的，其结果将导致可运行状态的进程开始阻塞运行队列。运行队列越大，执行进程所花费的时间也越长。

一个很流行的术语叫“load（负载）”经常被用来描述运行队列的状态，系统负载是由正在执行的进程和 CPU 运行队列中的进程的结合，如果有 2 个线程正在一个双核系统中执行且 4 个正在运行队列中，那么负载数即是 6，像 top 等工具可查看过去 1,5,15 分钟的负载均值。

3.3 CPU 利用率

CPU 利用率被定义为 CPU 使用的百分比，CPU 如何被利用是衡量一个系统的重要标准。多数性能监测工具把 CPU 利用分为以下几个类型：

*** 用户时间——CPU 花在执行用户空间进程的时间百分比**

*** 系统时间——CPU 花在执行内核进程和中断的时间百分比**

*** IO 等待——CPU 花在等待 IO 请求完成的时间百分比**

*** IDLE——CPU 的空闲时间百分比**

4.0 CPU 性能监测

理解 CPU 的性能状态即是理解中断，运行队列和上下文切换的状态。之前有提到过性能与基准信息有密切关系，但是有些常规的性能预期：

*** 运行队列——每个处理器上的运行队列不应该有超过 1-3 个排队的线程。例如，一个双核系统不应该有超过 6 个进行在运行队列里。**

*** CPU 利用率——假如一个 CPU 满状态负荷，那么以下的平衡关系需要达到：**

65%--70%的用户时间

30%--35%的系统时间

0%--5%的空闲时间

*** 上下文切换——上下文切换的数量与 CPU 的利用率有直接关系。如果 CPU 处于高负荷状态下那么大量的上下文切换是正常的。**

linux 系统里有很多可以监测以上数据的工具，如 vmstat 和 top。

4.1 vmstat 工具的使用

vmstat 工具的低开销使得它可以在一个高负载的系统上持续运行，它有两种工作模式：均值模式和采样模式。采样模式如下：

```
# vmstat 1
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b    swpd    free    buff  cache   si   so    bi    bo   in    cs us sy id wa
 0  0 104300   16800   95328  72200    0    0     5    26    7    14  4  1 95  0
 0  0 104300   16800   95328  72200    0    0     0    24 1021    64  1  1 98  0
 0  0 104300   16800   95328  72200    0    0     0     0 1009    59  1  1 98  0
```

每个区域的含义：

区域	描述
r	run queue 运行队列中的进程数
b	blocked 等待 IO 请求完成的阻塞进程数
in	interrupts 正在处理的中断数
cs	context switches 正在发生的上下文切换数
us	user 用户 CPU 时间百分比
sys	system 内核 CPU 时间百分比
wa	wait 可运行进程等待 IO 百分比
id	idle CPU 空闲时间百分比

4.2 案例分析：CPU 持续性利用

```
# vmstat 1
procs                      memory      swap      io     system         cpu
 r  b    swpd    free    buff  cache   si   so    bi    bo   in    cs us sy wa id
 3  0 206564   15092   80336 176080    0    0     0     0  718    26 81 19  0  0
 2  0 206564   14772   80336 176120    0    0     0     0  758    23 96  4  0  0
```

1	0	206564	14208	80336	176136	0	0	0	0	820	20	96	4	0	0
1	0	206956	13884	79180	175964	0	412	0	2680	1008	80	93	7	0	0
2	0	207348	14448	78800	175576	0	412	0	412	763	70	84	16	0	0
2	0	207348	15756	78800	175424	0	0	0	0	874	25	89	11	0	0
1	0	207348	16368	78800	175596	0	0	0	0	940	24	86	14	0	0
1	0	207348	16600	78800	175604	0	0	0	0	929	27	95	3	0	2
3	0	207348	16976	78548	175876	0	0	0	2508	969	35	93	7	0	0
4	0	207348	16216	78548	175704	0	0	0	0	874	36	93	6	0	1
4	0	207348	16424	78548	175776	0	0	0	0	850	26	77	23	0	0
2	0	207348	17496	78556	175840	0	0	0	0	736	23	83	17	0	0
0	0	207348	17680	78556	175868	0	0	0	0	861	21	91	8	0	1

可从数据得到如下观察结果：

*** 其拥有很高的中断数（in）和很低的上下文切换数，这说明可能有单个进程在进行大量的硬件资源请求。**

*** 用户时间平均在 85%以上，说明此进程一直停留在处理器中。**

*** 运行队列数刚好达到可接受的上限值，且出现超过上限值的情况。**

4.3 案例分析：超载调度

以下示例中内核调度器的上下文切换达到饱和：

```
# vmstat 1
```

procs					memory		swap		io		system				cpu
r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	wa	id
2	1	207740	98476	81344	180972	0	0	2496	0	900	2883	4	12	57	27
0	1	207740	96448	83304	180984	0	0	1968	328	810	2559	8	9	83	0
0	1	207740	94404	85348	180984	0	0	2044	0	829	2879	9	6	78	7
0	1	207740	92576	87176	180984	0	0	1828	0	689	2088	3	9	78	10
2	0	207740	91300	88452	180984	0	0	1276	0	565	2182	7	6	83	4
3	1	207740	90124	89628	180984	0	0	1176	0	551	2219	2	7	91	0
4	2	207740	89240	90512	180984	0	0	880	520	443	907	22	10	67	0
5	3	207740	88056	91680	180984	0	0	1168	0	628	1248	12	11	77	0
4	2	207740	86852	92880	180984	0	0	1200	0	654	1505	6	7	87	0
6	1	207740	85736	93996	180984	0	0	1116	0	526	1512	5	10	85	0
0	1	207740	84844	94888	180984	0	0	892	0	438	1556	6	4	90	0

可从数据得到如下观察结果：

*** 上下文切换数高于中断数，这表明内核必须花费相当数量的时间来处理上下文切换进程。**

*** 大量的上下文切换引起 CPU 利用的不平衡，明显的事实是大量的 IO 等待和用户时间的不足。**

*** 由于 CPU 受 IO 等待的限制，运行队列开始阻塞。**

4.4 mpstat 工具的使用

如果你的系统有多个处理器核心，你就可以使用 mpstat 工具来监测每个核心。linux 内

核把一个双核处理器当做 2 个 CPU，所以一个拥有 2 颗双核心的系统将被视为 4CPU。

```
# mpstat  - P ALL 1
Linux 2.4.21-20.ELsmp (localhost.localdomain)     05/23/2006

05:17:31 PM   CPU     %user     %nice %system     %idle       intr/s
05:17:32 PM   all      0.00      0.00      3.19     96.53      13.27
05:17:32 PM     0      0.00      0.00      0.00    100.00       0.00
05:17:32 PM     1      1.12      0.00     12.73     86.15      13.27
05:17:32 PM     2      0.00      0.00      0.00    100.00       0.00
05:17:32 PM     3      0.00      0.00      0.00    100.00       0.00
```

4.5 案例分析：未过载状态

下面的案例中有 4 个 CPU：

```
# top -d 1
top - 23:08:53 up   8:34,   3 users,   load average: 0.91, 0.37, 0.13
Tasks: 190 total,     4 running, 186 sleeping,     0 stopped,     0 zombie
Cpu(s): 75.2% us,   0.2% sy,   0.0% ni, 24.5% id,   0.0% wa,   0.0% hi,   0.0%
si
Mem:     2074736k total,     448684k used,   1626052k free,       73756k buffers
Swap:    4192956k total,          0k used,    4192956k free,     259044k cached

  PID USER         PR   NI   VIRT   RES   SHR S %CPU %MEM       TIME+   COMMAND
15957 nobody       25    0   2776   280   224 R   100   20.5    0:25.48 php
15959 mysql        25    0   2256   280   224 R   100   38.2    0:17.78 mysqld
15960 apache       25    0   2416   280   224 R   100   15.7    0:11.20 httpd
15901 root         16    0   2780 1092   800 R     1    0.1     0:01.59 top
    1 root         16    0   1780   660   572 S     0    0.0     0:00.64 init

    # mpstat  - P ALL 1
    Linux 2.4.21-20.ELsmp (localhost.localdomain)     05/23/2006

    05:17:31 PM   CPU     %user     %nice %system     %idle       intr/s
    05:17:32 PM   all     81.52      0.00     18.48     21.17      130.58
    05:17:32 PM     0     83.67      0.00     17.35      0.00      115.31
    05:17:32 PM     1     80.61      0.00     19.39      0.00       13.27
    05:17:32 PM     2      0.00      0.00     16.33     84.66        2.01
    05:17:32 PM     3     79.59      0.00     21.43      0.00        0.00

    05:17:32 PM   CPU     %user     %nice %system     %idle       intr/s
    05:17:33 PM   all     85.86      0.00     14.14     25.00      116.49
    05:17:33 PM     0     88.66      0.00     12.37      0.00      116.49
    05:17:33 PM     1     80.41      0.00     19.59      0.00        0.00
    05:17:33 PM     2      0.00      0.00      0.00    100.00        0.00
```

```
05:17:33 PM       3     83.51      0.00     16.49      0.00          0.00

05:17:33 PM    CPU     %user     %nice %system     %idle      intr/s
05:17:34 PM    all     82.74      0.00     17.26     25.00      115.31
05:17:34 PM      0     85.71      0.00     13.27      0.00      115.31
05:17:34 PM      1     78.57      0.00     21.43      0.00        0.00
05:17:34 PM      2      0.00      0.00      0.00    100.00        0.00
05:17:34 PM      3     92.86      0.00      9.18      0.00        0.00

05:17:34 PM    CPU     %user     %nice %system     %idle      intr/s
05:17:35 PM    all     87.50      0.00     12.50     25.00      115.31
05:17:35 PM      0     91.84      0.00      8.16      0.00      114.29
05:17:35 PM      1     90.82      0.00     10.20      0.00        1.02
05:17:35 PM      2      0.00      0.00      0.00    100.00        0.00
05:17:35 PM      3     81.63      0.00     15.31      0.00        0.00
```

你可以使用 ps 命令判定哪个进程运行在哪个 CPU 上：

```
# while :; do    ps -eo pid,ni,pri,pcpu,psr,comm | grep 'mysqld'; sleep 1;
done
  PID  NI PRI %CPU PSR COMMAND
15775   0  15 86.0   3 mysqld
  PID  NI PRI %CPU PSR COMMAND
15775   0  14 94.0   3 mysqld
  PID  NI PRI %CPU PSR COMMAND
15775   0  14 96.6   3 mysqld
  PID  NI PRI %CPU PSR COMMAND
15775   0  14 98.0   3 mysqld
  PID  NI PRI %CPU PSR COMMAND
15775   0  14 98.8   3 mysqld
  PID  NI PRI %CPU PSR COMMAND
15775   0  14 99.3   3 mysqld
```

4.6 小结

CPU 的性能监测包含以下部分：

*** 检查系统运行队列并确保每个核心上不超过 3 个可运行进程**

*** 确保 CPU 利用率的用户时间和系统时间在 70/30 之间**

*** 当 CPU 花费更多的时间在 system mode 上时，更有可能是因过载而试图重新调度优先级**

*** 运行 CPU 限制型应用比 IO 限制型应用更易出现性能瓶颈**

5.0 虚拟内存介绍

虚拟内存是使用磁盘作为 RAM 的扩充使得可用内存的有效大小得到相应增加。内核会将当前内存中未被使用的块的内容写入硬盘以此来腾出内存空间。当上面的内容再次需要被用到时，它们将被重新读入内存。这些对用户完全透明；在 linux 下运行的程序只会看到有大量内存空间可用而不会去管它们的一部分时不时的从硬盘读取。当然，硬盘的读写操作速度比内存慢上千倍，所以这个程序的运行速度也不会很快。这个被用作虚拟内存的硬盘空间呗称作交换空间（swap space）。

5.1 虚拟内存页

虚拟内存被分成页，在 X86 架构下的每个虚拟内存页大小为 4KB，当内核由内存向磁盘读写数据时，是以页为单位。内核会把内存页写入 swap 空间和文件系统。

5.2 内核内存分页

内存分页是一个常见的动作，不能和内存交换相混淆。内存分页是在正常间隔下同步内存中信息至磁盘的过程。随着时间的增长，应用会不断耗尽内存，有时候内核必须扫描内存并回收被分配给其他应用的空闲页面。

5.3 页帧回收算法（PFRA）

PFRA 负责回收内存。PFRA 根据页类型来判断是否被回收，内存页有以下类型：

* 不可被回收——locked，kernel，reserved 被锁、内核、保留页
* 可交换——无名内存页
* 可同步——被磁盘同步的页面
* 可丢弃——静态页，废弃页

除了不可被回收类型外其他均可被 PFRA 回收。PFRA 具有两个主要功能，一个是 kswapd 内核进程，一个是“Low On Memory Reclaiming”功能。

5.4 kswapd

kswapd 守护进程负责确保内存保持可用空闲空间。它监测内核中的 pages_high 和 pages_low 标记，如果空闲内存空间值小于 pages_low 值，kswwapd 进程开始扫描并尝试每次回收 32 个页面，如此重复直至空闲内存空间大于 pages_high 值。

kswapd 进程履行以下操作：

* **假如页面未改变，它将该页面放入 free list。**
* **假如页面发生改变且被文件系统回写，它将页面内容写入磁盘。**
* **假如页面发生改变且未被文件系统回写（无名页），它将页面内容写入 swap 设备。**

5.5 pdflush 内核分页

pdflush 守护进程负责同步所有与文件系统相关的页面至磁盘，换句话说，就是当一个文件在内存中发生改变，pdflush 守护进程就将其回写进磁盘。

当内存中的脏页面数量超过 10%时 pdflush 守护进程开始回写，这个动作与内核的 vm.dirty_background_ratio 参数值有关。

```
# sysctl -n vm.dirty_background_ratio
10
```

多数情况下 pdflush 守护进程与 PFRA 之间是相互独立的。除了常规的回收动作之外，当内核调用 LMR 算法时，LMR 将强制 pdflush 进行脏页面回收。

5.6 案例分析：大规模写入 I/O

vmstat 命令除了报告 CPU 的情况外还能查看虚拟内存的使用情况，vmstat 输出的以下区域与虚拟内存有关：

区域	描述
swapd	当前使用的虚拟内存 KB 数
free	当前空闲的物理内存 KB 数
buff	供 read()和 write()使用的物理内存缓冲大小
cache	映射到进程地址空间的物理内存数
so	swap out 从内存到 swap 写入的大小
si	swap in 从 swap 到内存写入的大小
bo	block out 从内存写入磁盘的大小
bi	block in 从磁盘写入内存你的大小

以下信息显示一个 I/O 限制型应用运行时的状态：

```
# vmstat 3
 procs                  memory                  swap                io        system                cpu
 r  b     swpd     free     buff    cache     si     so       bi      bo     in       cs us sy id wa
 3  2 809192 261556    79760   886880    416      0    8244     751    426     863 17  3  6 75
 0  3 809188 194916    79820   952900    307      0 21745    1005 1189    2590 34  6 12 48
 0  3 809188 162212    79840   988920     95      0 12107       0 1801    2633  2  2  3 94
 1  3 809268  88756    79924 1061424    260     28 18377     113 1142    1694  3  5  3 88
 1  2 826284  17608    71240 1144180    100 6140 25839 16380 1528    1179 19  9 12 61
 2  1 854780  17688    34140 1208980      1 9535 25557 30967 1764    2238 43 13 16 28
 0  8 867528  17588    32332 1226392     31 4384 16524 27808 1490    1634 41 10  7 43
 4  2 877372  17596    32372 1227532    213 3281 10912  3337  678     932 33  7  3 57
 1  2 885980  17800    32408 1239160    204 2892 12347 12681 1033     982 40 12  2 46
 5  2 900472  17980    32440 1253884     24 4851 17521  4856  934    1730 48 12 13 26
 1  1 904404  17620    32492 1258928     15 1316  7647 15804  919     978 49  9 17 25
 4  1 911192  17944    32540 1266724     37 2263 12907  3547  834    1421 47 14 20 20
 1  1 919292  17876    31824 1275832      1 2745 16327  2747  617    1421 52 11 23 14
 5  0 925216  17812    25008 1289320     12 1975 12760  3181  772    1254 50 10 21 19
 0  5 932860  17736    21760 1300280      8 2556 15469  3873  825    1258 49 13 24 15
```

通过以上数据可得出以下结果：

* **大量的数据从磁盘读入内存（bi），因 cache 值在不断增长**
* **尽管数据正不断消耗内存，空闲空间仍保持在 17M 左右**

*** buff 值正不断减少，说明 kswapd 正不断回收内存**

*** swpd 值不断增大，说明 kswapd 正将脏页面内容写入交换空间（so）**

5.7 小结

虚拟内存的性能监测包括以下步骤：

*** 当系统利用内存缓存超过磁盘缓存，系统反应速度更快**

*** 除在有大量持续的交换空间和磁盘读入动作情况下外，空闲内存空间很少说明 cache 得到了有效的利用**

*** 如果系统报告有持续的交换空间使用，说明内存不足**

6.0 I/O 监测介绍

磁盘 IO 子系统是 linux 系统里最慢的部分，这是由于其与 CPU 相比相去甚远，且依赖于物理式工作（转动和检索）。如果将读取磁盘和读取内存所花费的时间转换为分秒，那么他们之间的差距是 7 天和 7 分钟，所以 linux 内核尽量少的进行磁盘操作是至关重要的。以下部分将描述下内核处理数据从磁盘到内存和从内存到磁盘的不同方式。

6.1 数据读写——内存页面

linux 内核将磁盘 IO 分为页面进行操作，大多数 linux 系统中默认页面大小为 4K，即以 4K 为单位进行磁盘和内存间的读写操作。我们可以使用 time 命令来查找页面大小：

```
# /usr/bin/time -v date
.....
Page size (bytes): 4096
.....
```

6.2 主要页错误（Major Page Faults）和次要页错误（Minor Page Faults）

linux 和大多数 UNIX 系统一样，使用虚拟内存层来映射物理地址空间，这种映射在某种意义上是说当一个进程开始运行，内核仅仅映射其需要的那部分，内核首先会搜索 CPU 缓存和物理内存，如果没有找到内核则开始一次 MPF，一次 MPF 即是一次对磁盘子系统的请求，它将数据页从磁盘和缓存读入 RAM。

一旦内存页被映射到高速缓冲区，内核便会试图使用这些页，被称作 MnPF,MnPF 通过重复使用内存页而缩短了内核时间。

在下面示例中，time 命令显示了当一个程序启动的时候产生了多少 MPF 和 MnPF，在第一次启动的时候产生了很多 MPF：

```
# /usr/bin/time -v evolution
.....
Major (requiring I/O) page faults: 163
Minor (reclaiming a frame) page faults: 5918
.....
```

第二次启动的时候 MPF 消失因为程序已经在内存中：

```
# /usr/bin/time -v evolution
.....
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 5581
.....
```

6.3 文件缓冲区

文件缓冲区可使内核减少对 MPFs 和 MnPFs 的使用，随着系统不断地 IO 操作，缓冲区会随之增大，直至内存空闲空间不足并开始回收，最终结果是系统管理员们开始关心这个事

实，但这只是系统正在很好的使用缓冲空间而已。
下面的输入来自/proc/meminfo 文件：

```
# cat /proc/meminfo
MemTotal: 2075672 kB
MemFree: 52528 kB
Buffers: 24596 kB
Cached: 1766844 kB
.....
```

以上显示系统拥有 2G 内存，当前有 52MB 空闲空间，24MB 的 buffer 供应磁盘写操作，1.7GB 的 cache 由磁盘读入内存。
内核通过 MnPF 机制来使用这些东东，光这些数据还不足以说明系统出现瓶颈。

6.4 内存页面分类

在 linux 内核中有 3 种内核页：

*** 读取页面——从磁盘读入（MPF）的只读页面，这些页面存在于缓冲区中包括不可改变的静态文件，二进制文件和库文件。内核会因需求而不断地将他们读入内存，如果内存变得不够用，内核会将他们“窃取”至空闲列表，这将导致某个应用通过 MPF 将它们重新加载至内存。**

*** 脏页面——在内存中被内核修改的页面，它们将被 pdflush 守护进程回写至磁盘，当内存不够用时，kswapd 进程也会和 pdflush 一道进行回写以释放更多内存空间。**

*** 无名页面——它们属于某个进程，但是没有任何文件或后端存储与之关联，它们不能被回写进磁盘，当内存不够用时 kswapd 守护进程会将它们写入交换空间直至 RAM 释放出来。**

6.5 数据页面磁盘回写

应用可能直接调用 fsync()或 sync()系统调用将脏页面回写入磁盘，这些系统调用会直接请求至 I/O 调度器。如果一个应用不调用它们，则 pdflush 守护进程会时不时地进行页面回写操作。

7.0 监测磁盘 I/O

在一定条件下系统会出现 I/O 瓶颈，可由很多监测工具监测到，如 top，vmstat，iostat，sar 等。这些工具输入的信息大致一样，也有不同之处，下面将讨论出现 I/O 瓶颈的情况。

7.1 计算每秒 IO 量

每一次向磁盘的 IO 请求都会花费一定时间，这主要是因为磁盘必须旋转，磁头必须检索。磁盘的旋转通常被称为“旋转延迟(rotational delay)”（RD），磁头的移动杯称为“磁盘检索(disk seek)”（DS）。因此每次 IO 请求的时间由 DS 和 RD 计算得来，磁盘的 RD 由驱动气的转速所固定，一 RD 被为磁盘一转的一半，计算一块 10000 转磁盘的 RD 如下：

1. 算出每转的秒数：60 秒/10000 转 = 0.006 秒/转
2. 转换为毫秒： 0.006*1000 毫秒 = 6 毫秒
3. 算出 RD 值： 6/2 = 3 毫秒
4. 加上平均检索时间：3+3 = 6 毫秒
5. 加上内部转移等待时间： 6+2 = 8 毫秒
6. 算出一秒的 IO 数：1000 毫秒/8 毫秒 = 125 次/秒（IOPS）

在一块万转硬盘上应用每请求一次 IO 需要 8 毫秒，每秒可提供 120 到 150 次操作。

7.2 随机 IO 和有序 IO

每次 IO 的数据量与系统的工作负荷相关。系统的负荷分两类：有序和随机。

7.2.1 有序 IO

iostat 命令可提供 IOPS 值和每次 IO 的数据量，有序负荷需要一次有序地读取大量数据，因此每次 IO 的数据量很大，其性能取决于短时间大数据量的执行能力。

```
# iostat -x 1

avg-cpu: %user   %nice   %sys    %idle
         0.00    0.00    57.1 4  42.86

Device:  rrqm/s wrqm/s  r/s  w/s   rsec/s   wsec/s   rkB/s   wkB/s   avgrq-sz avgqu-sz await svctm %util
/dev/sda  0.00  12891.43 0.00 105.71 0.00 106080.00 0.00 53040.00 1003.46 1099.43 3442.43 26.49 280.00
/dev/sda1 0.00  0.00   0.00  0.00  0.00      0.00     0.00     0.00    0.00     0.00    0.00  0.00   0.00
/dev/sda2 0.00  12857.14 0.00 5.71 0.00  105782.86   0.00 52891.43 18512.00  559.14 780.00 490.00 280.00
/dev/sda3 0.00  34.29  0.00  100.00 0.00  297.14     0.00 148.57       2.97  540.29 3594.57 24.00 240.00

avg-cpu: %user %nice %sys %idle
0.00 0.00 23.53 76.47

Device:  rrqm/s wrqm/s  r/s   w/s  rsec/s  wsec/s   rkB/s   wkB/s   avgrq-sz avgqu-sz await svctm %util
/dev/sda  0.00 17320.59 0.00 102.94 0.00 142305.88 0.00  71152.94    1382.40 6975.29 952.29 28.57 294.12
/dev/sda1 0.00   0.00   0.00   0.00 0.00      0.00 0.00   0.00          0.00    0.00   0.00   0.00  0.00
/dev/sda2 0.00 16844.12 0.00 102.94 0.00 138352.94 0.00 69176.47   1344.00  6809.71  952.29 28.57 294.12
/dev/sda3 0.00 476.47   0.00 0.00   0.00    952.94 0.00  1976.47      0.00    165.59   0.00  0.00 276.47
```

计算 IOPS 能力的方式是用每秒读写数据量除以每秒读写次数：
53040/105 = 505 KB/IO
71152/102 = 697 KB/IO

7.2 随机 IO

随机负荷与数据大小关系不大，与磁盘的 IOPS 值有关，web 服务，mail 服务一般属于此类，IO 请求相对较少，其依赖于一次可执行多少次请求，因此磁盘的 IOPS 值非常关键。

```
avg-cpu: %user %nice %sys %idle
          2.04 0.00 97.96 0.00

Device:  rrqm/s  wrqm/s r/s w/s rsec/s wsec/s rkB/s wkB/s avgrq-sz avgqu-sz await svctm %util
/dev/sda 0.00  633.67 3.06 102.31 24.49 5281.63 12.24 2640.82 288.89 73.67 113.89 27.22 50.00
/dev/sda1 0.00   5.10  0.00 2.04  0.00  57.14   0.00   28.57   28.00  1.12  55.00 55.00 11.22
/dev/sda2 0.00 628.57 3.06 100.27 24.49 5224.49 12.24 2612.24 321.50 72.55 121.25 30.63 50.00
/dev/sda3 0.00   0.00  0.00  0.00 0.00   0.00  0.00  0.00       0.00  0.00   0.00  0.00  0.00

avg-cpu: %user %nice %sys %idle
          2.15 0.00 97.85 0.00

Device: rrqm/s wrqm/s r/s w/s  rsec/s wsec/s rkB/s wkB/s avgrq-sz avgqu-sz await svctm %util
/dev/sda 0.00  41.94  6.45 130.98 51.61 352.69 25.81 3176.34 19.79  2.90    286.32 7.37 15.05
/dev/sda1 0.00 0.00   0.00   0.00  0.00   0.00  0.00    0.00  0.00  0.00      0.00 0.00  0.00
/dev/sda2 0.00 41.94  4.30 130.98 34.41 352.69 17.20 3176.34 21.18  2.90    320.00 8.24 15.05
/dev/sda3 0.00 0.00   2.15   0.00 17.20   0.00  8.60    0.00  8.00  0.00      0.00 0.00  0.00
```

此数据与刚才的数据相比每次 IO 写入的数据量差异很大：
2640/102 = 23 KB/IO
3176/130 = 24 KB/IO

7.3 当虚拟内存终结 I/O 时

假如系统没有足够的内存供给所有的请求时，并开始使用交换空间，和文件系统 IO 一样 swap 的写入也很慢，假如内存严重的不足，那么这可能导致大量的页面涌向 swap，假如这个 swap 空间与要写入的文件处于同一文件系统下，此系统将进入 IO 争抢，这将导致系统性能直线下降。假如页面无法读或写入磁盘，那么它们将一直停留于内存中，内核会开始尝试释放内存空间，问题是 IO 通道已经严重阻塞且无法做任何事情，这必将导致系统内核出错并崩溃。
下面的 vmstat 输出展示了这种情况：

```
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
 r  b      swpd    free   buff  cache   si   so    bi    bo   in    cs us sy id wa
17  0      1250    3248 45820 1488472   30  132   992     0 2437  7657 23 50  0 23
11  0      1376    3256 45820 1488888   57  245   416     0 2391  7173 10 90  0  0
12  0      1582    1688 45828 1490228   63  131  1348    76 2432  7315 10 90  0 10
12  2      3981    1848 45468 1489824  185   56  2300    68 2478  9149 15 12  0 73
14  2     10385    2400 44484 1489732    0   87  1112    20 2515 11620  0 12  0 88
```

14 2 12671 2280 43644 1488816 76 51 1812 204 2546 11407 20 45 0 35

以上数据显示大量的读入内存请求（bi），内存不够，导致数据块不停写入（so）交换空间，交换空间大小不断增长（swpd），同样也出现了大量的 IO 等待时间（wa），这表明 CPU 因为 IO 请求的原因而开始变慢。

```
# iostat -x 1

avg-cpu: %user %nice %sys %idle
         0.00  0.00 100.00 0.00

Device: rrqm/s wrqm/s   r/s w/s rsec/s wsec/s    rkB/s    wkB/s   avgrq-sz  avgqu-sz   await svctm %util
/dev/sda  0.00  1766.67 4866.67 1700.00 38933.33 31200.00 19466.67 15600.00 10.68    6526.67 100.56 5.08
3333.33
/dev/sda1 0.00 933.33   0.00    0.00     0.00  7733.33     0.00  3866.67  0.00 20.00 2145.07 7.37 200.00
/dev/sda2 0.00 0.00  4833.33    0.00 38666.67   533.33  19333.33  266.67  8.11 373.33 8.07   6.90  87.00
/dev/sda3 0.00 833.33  33.33 1700.00   266.67 22933.33    133.33 11466.67   13.38   6133.33 358.46 11.35
1966.67
```

在上面的例子中，swap 设备（/dev/sda1）和文件系统（/dev/sda3）在争抢 IO。

7.4 判定应用 IO 使用

iotop 命令显示每个进程的 IO 使用情况，其输出结果与 top 命令相似，可与 iostat 命令结合来判断产生 IO 瓶颈的应用。

在下面的示例中，在同一磁盘上（sda2）既有读操作（r/s）又有写操作(w/s)：

```
# iostat  - x 1

avg-cpu:   %user     %nice %system %iowait   %steal     %idle
             7.14      0.00     35.71     57.14      0.00      0.00

Device:         rrqm/s   wrqm/s     r/s     w/s    rsec/s    wsec/s avgrq-sz avgqu-sz   await  svctm  %util
sda               0.00     0.00  123.47   25.51    987.76  21951.02   153.97    27.76  224.29   6.85 102.04
sda1              0.00     0.00    0.00    0.00      0.00      0.00     0.00     0.00    0.00   0.00   0.00
sda2              0.00     0.00  123.47   25.51    987.76  21951.02   153.97    27.76  224.29   6.85 102.04

#iotop -d 5 -P
Total DISK READ: 981.23 K/s | Total DISK WRITE: 21.43 M/s
  PID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 2574 be/4 root        967.01 K/s    0.00 B/s  0.00 % 39.05 % find /
   64 be/3 root          0.00 B/s   19.94 M/s  0.00 % 13.09 % smbd -D
 2533 be/4 dhoch         3.63 K/s    8.72 K/s  0.00 %  1.82 % [kjournald]
 2442 be/4 root          0.00 B/s    2.91 K/s  0.00 %  0.46 % iostat -x 1
 2217 be/4 dhoch         0.00 B/s 1488.57 B/s  0.00 %  0.00 % mono /usr~-ior-fd=25
 1985 be/4 dhoch         0.00 B/s  255.12 K/s  0.00 %  0.00 % smbd -D
```

iotop 的 DISK READ 和 DISK WRITE 与 iostat 的 rsec/s 和 wsec/s 相对应。

7.5 小结

I/O 性能监测可总结如下：

*** 任何时间出现 CPU 等待 IO，说明磁盘超载。**

*** 计算出你的磁盘可维持的 IOPS 值。**

*** 判定你的应用时属于随机磁盘访问型还是有序型。**

*** 通过对比等待时间和服务时间即可判断磁盘是否缓慢。**

*** 监测交换空间和文件系统坐在分区并确保他们之间不存在争抢 IO。**

8.0 网络监测介绍

网络是所有子系统中最难监测的一个，因为网络比较抽象，在监测时有很多在系统可控制之外的因素如延迟，冲突，拥塞和丢包等对监测产生影响。下面将讨论的是以太网、IP、TCP 的性能监测。

8.1 以太网配置设定

除非有明确的设定，所有以太网的速度都是自动协商的，这很大程度上是由于历史原因造成的，早些时候一个网络里经常有不同网速和双工模式的网络设备。

大多数企业以太网是 100BaseTX 或 1000BaseTX，可以使用 ethtool 工具来判断一个系统的网速。

下面的示例中一个拥有 100BaseTX 网卡的机器工作在 10BaseTX 下：

```
# ethtool eth0
Settings for eth0:
        Supported ports: [ TP MII ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
        Advertised auto-negotiation: Yes
        Speed: 10Mb/s
        Duplex: Half
        Port: MII
        PHYAD: 32
        Transceiver: internal
        Auto-negotiation: on
        Supports Wake-on: pumbg
        Wake-on: d
        Current message level: 0x00000007 (7)
        Link detected: yes
```

下面将其强制设定为 100BaseTX 模式：

```
# ethtool -s eth0 speed 100 duplex full autoneg off
# ethtool eth0
Settings for eth0:
        Supported ports: [ TP MII ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
        Advertised auto-negotiation: No
        Speed: 100Mb/s
```

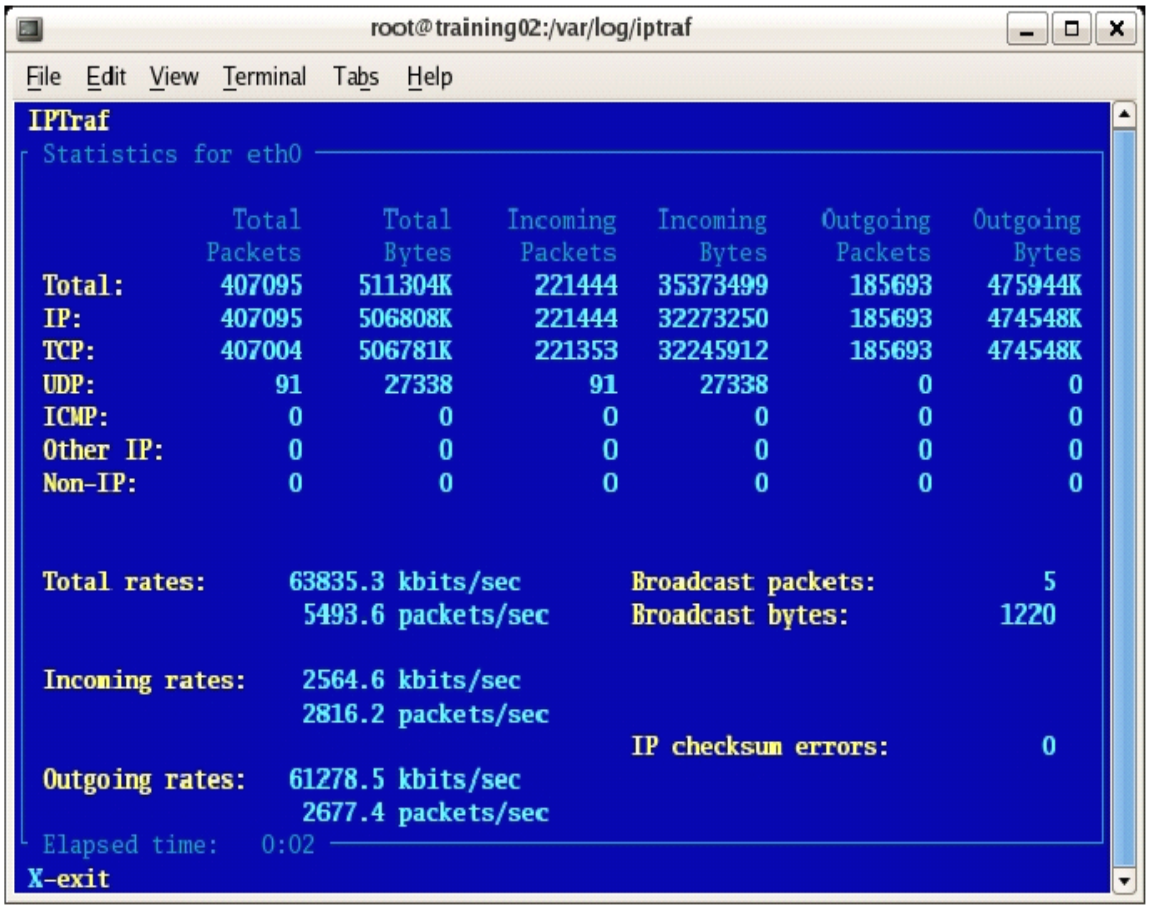
Duplex: Full
Port: MII
PHYAD: 32
Transceiver: internal
Auto-negotiation: off
Supports Wake-on: pumbg
Wake-on: d
Current message level: 0x00000007 (7)
Link detected: yes

8.2 网络吞吐量监测

监测网络吞吐量最好的办法是在两个系统之间发送流量并统计其延迟和速度。

8.2.0 使用 iptraf 监测本地吞吐量

iptraf 工具可提供以太网卡的吞吐量情况：
iptraf -d eth0



上面的数据显示被测试系统正以 61mbps（7.65M）频率发送数据，相比于 100mbps 网络这有点低。

8.2.1 使用 netperf 监测远端吞吐量

与 iptraf 的动态监测不一样的是 netperf 使用可控方式测试网络，这一点对测试一个客户端到一个高负载服务器之间的吞吐量很有帮助，netperf 工具是以 C/S 模式运行。
首先需要在服务器上运行 netperf 服务端：

```
server# netserver
Starting netserver at port 12865
Starting netserver at hostname 0.0.0.0 port 12865 and family AF_UNSPEC
```

netperf 可以执行多种测试，最基本的是标准测试：

```
client# netperf -H 192.168.1.215 -l 30
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
192.168.1.230 (192.168.1.230) port 0 AF_INET
Recv     Send      Send
Socket Socket   Message   Elapsed
Size     Size      Size       Time        Throughput
bytes   bytes     bytes      secs.       10^6bits/sec

 87380   16384   16384      30.02          89.46
```

输出显示吞吐量在 89mbps 左右，服务器和客户端在同一网段。
从一个 10 跳的 54G 无线网进行测试只能达到 14mbps 左右：

```
client# netperf -H 192.168.1.215 -l 30
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
192.168.1.215 (192.168.1.215) port 0 AF_INET
Recv     Send      Send
Socket Socket   Message   Elapsed
Size     Size      Size       Time        Throughput
bytes   bytes     bytes      secs.       10^6bits/sec

 87380   16384   16384      30.10          14.09
```

从 50 跳距离局域网测试：

```
# netperf -H 192.168.1.215 -l 30
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
192.168.1.215 (192.168.1.215) port 0 AF_INET
Recv     Send      Send
Socket Socket   Message   Elapsed
Size     Size      Size       Time        Throughput
bytes   bytes     bytes      secs.       10^6bits/sec

 87380   16384   16384      30.64           5.05
```

从外网测试：

```
# netperf -H litemail.org -p 1500 -l 30
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
```

litemail.org (72.249.104.148) port 0 AF_INET
Recv Send Send
Socket Socket Message Elapsed
Size Size Size Time Throughput
bytes bytes bytes secs. 10^6bits/sec

87380 16384 16384 31.58 0.93

通过 VPN 测试：
netperf -H 10.0.1.129 -l 30
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
10.0.1.129 (10.0.1.129) port 0 AF_INET
Recv Send Send
Socket Socket Message Elapsed
Size Size Size Time Throughput
bytes bytes bytes secs. 10^6bits/sec

87380 16384 16384 31.99 0.51

另外一个有用的测试模式是测试 TCP 请求应答速率，其原理是建立一个 TCP 连接并发送多个请求：
client# netperf -t TCP_RR -H 192.168.1.230 -l 30
TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET
to 192.168.1.230 (192.168.1.230) port 0 AF_INET
Local /Remote
Socket Size Request Resp. Elapsed Trans.
Send Recv Size Size Time Rate
bytes Bytes bytes bytes secs. per sec

16384 87380 1 1 30.00 4453.80
16384 87380

数据显示网络可支持 4453 左右的 psh/ack 每秒，因为发送包的大小只有 1k。下面使用 2k 的请求和 32k 的应答：
client# netperf -t TCP_RR -H 192.168.1.230 -l 30 -- -r 2048,32768
TCP REQUEST/RESPONSE TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to
192.168.1.230 (192.168.1.230) port 0 AF_INET
Local /Remote
Socket Size Request Resp. Elapsed Trans.
Send Recv Size Size Time Rate
bytes Bytes bytes bytes secs. per sec

16384 87380 2048 32768 30.00 222.37
16384 87380

可以看到速率已经降到 222 左右。

8.2.3 使用 iperf 测试网络性能

iperf 与 netperf 相似在两台机器间进行测试，不同之处在于 iperf 测试更深入，它只有一个可执行文件既可用于服务端又可用于客户端，默认通信端口是 5001.

启动服务端（192.168.1.215）：

```
server# iperf -s -D
Running Iperf Server as a daemon
The Iperf daemon process ID : 3655
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

一个满负荷无线网中的客户端进行测试：

```
client# iperf -c 192.168.1.215 -t 60 -i 5
------------------------------------------------------------
Client connecting to 192.168.1.215, TCP port 5001
TCP window size: 25.6 KByte (default)
------------------------------------------------------------
[  3] local 192.168.224.150 port 51978 connected with
192.168.1.215 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 5.0 sec  6.22 MBytes  10.4 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  5.0-10.0 sec  6.05 MBytes  10.1 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3] 10.0-15.0 sec  5.55 MBytes  9.32 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3] 15.0-20.0 sec  5.19 MBytes  8.70 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3] 20.0-25.0 sec  4.95 MBytes  8.30 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3] 25.0-30.0 sec  5.21 MBytes  8.74 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3] 30.0-35.0 sec  2.55 MBytes  4.29 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3] 35.0-40.0 sec  5.87 MBytes  9.84 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3] 40.0-45.0 sec  5.69 MBytes  9.54 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3] 45.0-50.0 sec  5.64 MBytes  9.46 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
```

```
[  3] 50.0-55.0 sec  4.55 MBytes  7.64 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3] 55.0-60.0 sec  4.47 MBytes  7.50 Mbits/sec
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-60.0 sec  61.9 MBytes  8.66 Mbits/sec
```

下面是进行 UDP 丢包率测试：

```
# iperf -c 192.168.1.215 -b 10M
WARNING: option -b implies udp testing
------------------------------------------------------------
Client connecting to 192.168.1.215, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:     107 KByte (default)
------------------------------------------------------------
[  3] local 192.168.224.150 port 33589 connected with 192.168.1.215 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  11.8 MBytes  9.90 Mbits/sec
[  3] Sent 8420 datagrams
[  3] Server Report:
[ ID] Interval       Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[  3]  0.0-10.0 sec  6.50 MBytes  5.45 Mbits/sec  0.480 ms 3784/ 8419 (45%)
[  3]  0.0-10.0 sec  1 datagrams received out-of-order
```

10M 的数据只传输了一部分，丢包率在 45%左右。

8.3 用 tcptrace 进行特定连接测试

tcptrace 工具提供了基于 TCP 的详细测试信息，基于 libpcap 文件，它可以监测出有时不太容易抓到的 TCP 流：

*** TCP 中继——需要被重新发送的包数量和数据大小**

*** TCP 窗口大小——使用很小的 window sizes 识别低速度连接**

*** 连接的总吞吐量**

*** 连接持续性**

8.3.1 案例分析

tcptrace 把 libpcap 文件作为输入，不加任何选项执行此工具可显示出包里所有信息，下面的示例中 tcptrace 处理一个叫 bigstuff 的包文件：

```
# tcptrace bigstuff
1 arg remaining, starting with 'bigstuff'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004

146108 packets seen, 145992 TCP packets traced
elapsed wallclock time: 0:00:01.634065, 89413 pkts/sec analyzed
trace file elapsed time: 0:09:20.358860
```

```
TCP connection info:
  1: 192.168.1.60:pcanywherestat - 192.168.1.102:2571 (a2b)          404>   450<
  2: 192.168.1.60:3356 - ftp.strongmail.net:21 (c2d)                  35>    21<
  3: 192.168.1.60:3825 - ftp.strongmail.net:65023 (e2f)               5>     4<
(complete)
  4: 192.168.1.102:1339 - 205.188.8.194:5190 (g2h)                      6>     6<
5: 192.168.1.102:1490 - cs127.msg.mud.yahoo.com:5050 (i2j)          5>     5<
  6: py-in-f111.google.com:993 - 192.168.1.102:3785 (k2l)           13>    14<
。。。  。。。
```

在上面输出的数据中可以看到每条信息前面都有一个编号，tcptrace 最常用的选项是-l 和-o，用来指定某个具体的连接。下面的示例：

```
# tcptrace -l -o1 bigstuff
1 arg remaining, starting with 'bigstuff'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004

146108 packets seen, 145992 TCP packets traced
elapsed wallclock time: 0:00:00.529361, 276008 pkts/sec analyzed
trace file elapsed time: 0:09:20.358860
TCP connection info:
32 TCP connections traced:
TCP connection 1:
        host a:           192.168.1.60:pcanywherestat
        host b:           192.168.1.102:2571
        complete conn: no          (SYNs: 0)  (FINs: 0)
        first packet:  Sun Jul 20 15:58:05.472983 2008
        last packet:   Sun Jul 20 16:00:04.564716 2008
        elapsed time:  0:01:59.091733
        total packets: 854
        filename:      bigstuff
   a->b:                              b->a:
     total packets:            404     total packets:            450
     ack pkts sent:            404     ack pkts sent:            450
     pure acks sent:            13     pure acks sent:           320
     sack pkts sent:             0     sack pkts sent:             0
     dsack pkts sent:            0     dsack pkts sent:            0
     max sack blks/ack:          0     max sack blks/ack:          0
     unique bytes sent:      52608     unique bytes sent:      10624
     actual data pkts:         391     actual data pkts:         130
     actual data bytes:      52608     actual data bytes:      10624
     rexmt data pkts:            0     rexmt data pkts:            0
     rexmt data bytes:           0     rexmt data bytes:           0
     zwnd probe pkts:            0     zwnd probe pkts:            0
     zwnd probe bytes:           0     zwnd probe bytes:           0
```

outoforder pkts:	0	outoforder pkts:	0
pushed data pkts:	391	pushed data pkts:	130
SYN/FIN pkts sent:	0/0	SYN/FIN pkts sent:	0/0
urgent data pkts:	0 pkts	urgent data pkts:	0 pkts
urgent data bytes:	0 bytes	urgent data bytes:	0 bytes
mss requested:	0 bytes	mss requested:	0 bytes
max segm size:	560 bytes	max segm size:	176 bytes
min segm size:	48 bytes	min segm size:	80 bytes
avg segm size:	134 bytes	avg segm size:	81 bytes
max win adv:	19584 bytes	max win adv:	65535 bytes
min win adv:	19584 bytes	min win adv:	64287 bytes
zero win adv:	0 times	zero win adv:	0 times
avg win adv:	19584 bytes	avg win adv:	64949 bytes
initial window:	160 bytes	initial window:	0 bytes
initial window:	2 pkts	initial window:	0 pkts
ttl stream length:	NA	ttl stream length:	NA
missed data:	NA	missed data:	NA
truncated data:	36186 bytes	truncated data:	5164 bytes
truncated packets:	391 pkts	truncated packets:	130 pkts
data xmit time:	119.092 secs	data xmit time:	116.954 secs
idletime max:	441267.1 ms	idletime max:	441506.3 ms
throughput:	442 Bps	throughput:	89 Bps

8.3.2 案例分析——计算转发百分比

```
# tcptrace -f'rexmit_segs>100' bigstuff
Output filter: ((c_rexmit_segs>100)OR(s_rexmit_segs>100))
1 arg remaining, starting with 'bigstuff'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004

146108 packets seen, 145992 TCP packets traced
elapsed wallclock time: 0:00:00.687788, 212431 pkts/sec analyzed
trace file elapsed time: 0:09:20.358860
TCP connection info:
  16: ftp.strongmail.net:65014 - 192.168.1.60:2158 (ae2af) 18695> 9817<

# tcptrace -l -o16 bigstuff
  arg remaining, starting with 'bigstuff'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov  4, 2004

146108 packets seen, 145992 TCP packets traced
elapsed wallclock time: 0:00:01.355964, 107752 pkts/sec analyzed
trace file elapsed time: 0:09:20.358860
TCP connection info:
```

32 TCP connections traced:
==========

TCP connection 16:
host ae: ftp.strongmail.net:65014
host af: 192.168.1.60:2158
complete conn: no (SYNs: 0) (FINs: 1)
first packet: Sun Jul 20 16:04:33.257606 2008
last packet: Sun Jul 20 16:07:22.317987 2008
elapsed time: 0:02:49.060381
total packets: 28512
filename: bigstuff

ae->af:		af->ae:	
unique bytes sent:	25534744	unique bytes sent:	0
actual data pkts:	18695	actual data pkts:	0
actual data bytes:	25556632	actual data bytes:	0
rexmt data pkts:	1605	rexmt data pkts:	0
rexmt data bytes:	2188780	rexmt data bytes:	0

计算方式：
1605/18695* 100 = 8.5%

8.2.3 案例分析——通过时间计算转发

tcptrace 工具提供了可以根据不同标尺（协议，端口，时间等）显示数据的模块，其中 slice 模块可以根据运行时间查看 TCP 性能。你可以准确地测量出何时发生了大量的包转发并与其他性能数据相结合来判断性能瓶颈之所在。
下面的示例展示了如何使用 slice 模块：
tcptrace –xslice bigfile
命令会在当前目录生成一个叫 slice.dat 的文件，这个文件包含了 15 秒为间隔的转发信息：
ls -l slice.dat
-rw-r--r-- 1 root root 3430 Jul 10 22:50 slice.dat
more slice.dat

date	segs	bytes	rexsegs	rexbytes	new	active
22:19:41.913288	46	5672	0	0	1	1
22:19:56.913288	131	25688	0	0	0	1
22:20:11.913288	0	0	0	0	0	0
22:20:26.913288	5975	4871128	0	0	0	1
22:20:41.913288	31049	25307256	0	0	0	1
22:20:56.913288	23077	19123956	40	59452	0	1
22:21:11.913288	26357	21624373	5	7500	0	1
22:21:26.913288	20975	17248491	3	4500	12	13
22:21:41.913288	24234	19849503	10	15000	3	5
22:21:56.913288	27090	22269230	36	53999	0	2

22:22:11.913288	22295	18315923	9	12856	0	2
22:22:26.913288	8858	7304603	3	4500	0	1

8.4 小结

网络性能监测有如下几点：

* 检查并确保所有网卡运行在适当的速率下。
* 检查每个网卡的总吞吐量并确保其符合网速。
* 监测网络流量类型以确保适当的流量有适当的优先级。

附：渐进式性能监测案例

下面的案例中一个终端用户报告了一个 web 用户接口需要 20 分钟才能处理完本该 15 分钟就能搞定的问题。

系统配置：

* RHEL3.7
* Dell 1850 双核处理器，2G 内存，75G 15K 硬盘
* 标准 LAMP 架构

性能分析流程：

1.vmstat 判断基本信息

vmstat 1 10

procs			memory			swap			io	system		cpu		
r	b	swpd	free	buff	cache	si	so	bi	bo	in	cs	us	sy	id	wa
1	0	249844	19144	18532	1221212	0	0	7	3	22	17	25	8	17	18
0	1	249844	17828	18528	1222696	0	0	40448	8	1384	1138	13	7	65	14
0	1	249844	18004	18528	1222756	0	0	13568	4	623	534	3	4	56	37
2	0	249844	17840	18528	1223200	0	0	35200	0	1285	1017	17	7	56	20
1	0	249844	22488	18528	1218608	0	0	38656	0	1294	1034	17	7	58	18
0	1	249844	21228	18544	1219908	0	0	13696	484	609	559	5	3	54	38
0	1	249844	17752	18544	1223376	0	0	36224	4	1469	1035	10	6	67	17
1	1	249844	17856	18544	1208520	0	0	28724	0	950	941	33	12	49	7
1	0	249844	17748	18544	1222468	0	0	40968	8	1266	1164	17	9	59	16
1	0	249844	17912	18544	1222572	0	0	41344	12	1237	1080	13	8	65	13

关键点：

* si，so 为 0，swpd，free 稳定，说明内存够用；
* idle 保持在 50%左右，说明 CPU 也没问题；
* cs，bo 值都很高；
* wa 保持在 20%左右。

2.使用 iostat 判断数据读取请求来自哪里

```
# iostat -x 1
Linux 2.4.21-40.ELsmp (mail.example.com)  03/26/2007

avg-cpu:  %user   %nice    %sys   %idle
          30.00    0.00    9.33   60.67

Device:    rrqm/s wrqm/s   r/s   w/s  rsec/s  wsec/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await  svctm  %util
/dev/sda  7929.01  30.34 1180.91 14.23 7929.01  357.84  3964.50   178.92     6.93     0.39    0.03   0.06   6.69
/dev/sda1    2.67   5.46  0.40  1.76   24.62   57.77    12.31    28.88    38.11     0.06    2.78   1.77   0.38
/dev/sda2    0.00   0.30  0.07  0.02    0.57    2.57     0.29     1.28    32.86     0.00    3.81   2.64   0.03
/dev/sda3 7929.01  24.58 1180.44 12.45 7929.01  297.50  3964.50   148.75     6.90     0.32    0.03   0.06   6.68

avg-cpu:  %user   %nice    %sys   %idle
          9.50    0.00   10.68   79.82

Device:    rrqm/s wrqm/s   r/s   w/s  rsec/s  wsec/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await  svctm  %util
/dev/sda     0.00   0.00 1195.24  0.00    0.00    0.00     0.00     0.00     0.00    43.69    3.60   0.99 117.86
/dev/sda1    0.00   0.00  0.00  0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00
/dev/sda2    0.00   0.00  0.00  0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00
/dev/sda3    0.00   0.00 1195.24  0.00    0.00    0.00     0.00     0.00     0.00    43.69    3.60   0.99 117.86

avg-cpu:  %user   %nice    %sys   %idle
          9.23    0.00   10.55   79.22

Device:    rrqm/s wrqm/s   r/s   w/s  rsec/s  wsec/s    rkB/s    wkB/s avgrq-sz avgqu-sz   await  svctm  %util
/dev/sda     0.00   0.00 1200.37  0.00    0.00    0.00     0.00     0.00     0.00    41.65    2.12   0.99 112.51
/dev/sda1    0.00   0.00  0.00  0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00
/dev/sda2    0.00   0.00  0.00  0.00    0.00    0.00     0.00     0.00     0.00     0.00    0.00   0.00   0.00
/dev/sda3    0.00   0.00 1200.37  0.00    0.00    0.00     0.00     0.00     0.00    41.65    2.12   0.99 112.51
```

关键点：

* 唯一活动的分区时/dev/sda3；
* sda3 上有 1200 左右的 IOPS（r/s）；
* rkB/s 为 0，这点与 vmstat 观察到的高 IO 等待相符；
* 高 IOPS 与 vmstat 观察到的高 cs 值相符。

3.使用 top 命令判定活跃度最高的应用

```
# top -d 1
 11:46:11   up 3 days, 19:13,   1 user,   load average: 1.72, 1.87, 1.80
176 processes: 174 sleeping, 2 running, 0 zombie, 0 stopped
CPU states:   cpu      user      nice   system      irq   softirq   iowait      idle
              total    12.8%     0.0%     4.6%     0.2%      0.2%    18.7%    63.2%
              cpu00    23.3%     0.0%     7.7%     0.0%      0.0%    36.8%    32.0%
              cpu01    28.4%     0.0%    10.7%     0.0%      0.0%    38.2%    22.5%
              cpu02     0.0%     0.0%     0.0%     0.9%      0.9%     0.0%    98.0%
              cpu03     0.0%     0.0%     0.0%     0.0%      0.0%     0.0%   100.0%
Mem:   2055244k av, 2032692k used,    22552k free,   0k shrd,    18256k buff
                    1216212k actv,   513216k in_d,    25520k in_c
Swap: 4192956k av,   249844k used, 3943112k free                  1218304k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
14939 mysql     25   0  379M 224M 1117 R  38.2 25.7% 15:17.78 mysqld
 4023 root      15   0  2120  972  784 R   2.0  0.3   0:00.06 top
    1 root      15   0  2008  688  592 S   0.0  0.2   0:01.30 init
    2 root      34  19     0    0    0 S   0.0  0.0   0:22.59 ksoftirqd/0
    3 root      RT   0     0    0    0 S   0.0  0.0   0:00.00 watchdog/0
    4 root      10  -5     0    0    0 S   0.0  0.0   0:00.05 events/0
```

关键点：

* mysql 进程使用系统资源最多；
* mysql 的大量请求导致 IO 等待。

4.使用 strace 命令观察读取请求的来路

strace -p 14939

Process 14939 attached - interrupt to quit
read(29, "\3\1\237\1\366\337\1\222%\4\2\0\0\0\0012P/d", 20) = 20
read(29, "ata1/strongmail/log/strongmail-d"..., 399) = 399
_llseek(29, 2877621036, [2877621036], SEEK_SET) = 0
read(29, "\1\1\241\366\337\1\223%\4\2\0\0\0\0012P/da", 20) = 20
read(29, "ta1/strongmail/log/strongmail-de"..., 400) = 400
_llseek(29, 2877621456, [2877621456], SEEK_SET) = 0
read(29, "\1\1\235\366\337\1\224%\4\2\0\0\0\0012P/da", 20) = 20
read(29, "ta1/strongmail/log/strongmail-de"..., 396) = 396
_llseek(29, 2877621872, [2877621872], SEEK_SET) = 0
read(29, "\1\1\245\366\337\1\225%\4\2\0\0\0\0012P/da", 20) = 20
read(29, "ta1/strongmail/log/strongmail-de"..., 404) = 404
_llseek(29, 2877622296, [2877622296], SEEK_SET) = 0
read(29, "\3\1\236\2\366\337\1\226%\4\2\0\0\0\0012P/d", 20) = 20

关键点：

* 数据表明 mysql 在执行大量的随机 IO；
* 大量的读取请求来自一个特别的查询。

5.使用 mysqladmin 命令检查查询耗时最长的语句

./mysqladmin -pstrongmail processlist

+----+------+-----------+------------+---------+------+----------+-----------------------------------
| Id | User | Host | db | Command | Time | State | Info
+----+------+-----------+------------+---------+------+----------+-----------------------------------
| 1 | root | localhost | strongmail | Sleep | 10 | | |
| 2 | root | localhost | strongmail | Sleep | 8 | | |
| 3 | root | localhost | root | Query | 94 | Updating | update `failures` set ùpdate_datasource`='Y' where database_id='32' and update_datasource='N' and |
| 14 | root | localhost | | Query | 0 | | show processlist

关键点：

* mysql 不断地执行 update 查询至 failures 表；
* 为了实施 update，数据库必须对该表进行索引。
* 就是 mysql 的这个动作导致了系统性能的下降。

后续处理

通过开发人员对 php 代码的检查，在代码里找到了问题所在，这条查询语句本来假定表里只有 10000 条记录，但是现在表里有 4000000 条，这就导致了 update 出现问题，查询变慢。