

# Linux 的系统级性能剖析工具-perf

## (三)

承刚

TAOBAO Kernel Team  
chenggang.qin@gmail.com

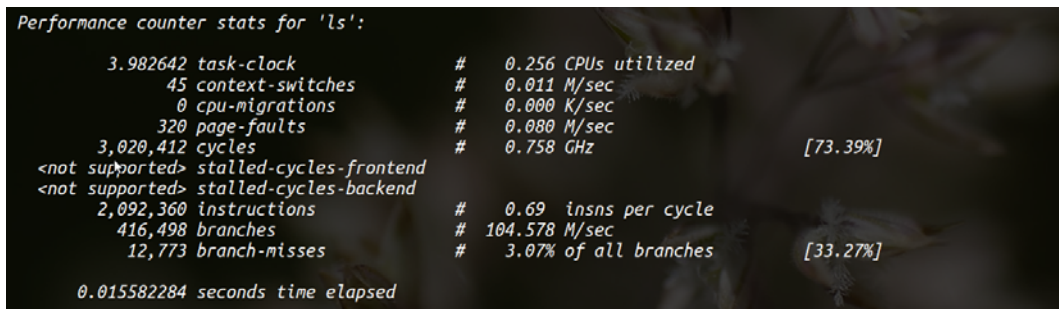
### 第四章 perf stat

#### 4.1 perf stat 的基本使用方法

perf stat 工具用来剖析一个应用程序的性能概况。使用方法非常简单，下面的命令能够得到'ls'程序的一些典型性能数据：

```
$perf stat ls
```

上述命令给出的性能概况如图 16 所示。



```
Performance counter stats for 'ls':
 3.982642 task-clock           #    0.256 CPUs utilized
   45 context-switches        #    0.011 M/sec
    0 cpu-migrations          #    0.000 K/sec
   320 page-faults            #    0.080 M/sec
3,020,412 cycles                #    0.758 GHz                [73.39%]
<not supported> stalled-cycles-frontend
<not supported> stalled-cycles-backend
 2,092,360 instructions        #    0.69 insns per cycle
 416,498 branches              # 104.578 M/sec
 12,773 branch-misses          #   3.07% of all branches    [33.27%]

0.015582284 seconds time elapsed
```

图 16. perf stat ls 的输出结果

从图上可以看到，perf stat 工具利用 10 个典型性能事件剖析了应用程序。task-clock 事件表示目标任务'ls'真正占用处理器的时间，单位是毫秒。我们将其称为任务执行时间。如图 16 所示，'ls'在处理器上执行了近 4 毫秒。“0.256 CPUs utilized”表示目标任务的处理器占用率。处理器占用率表示目标任务的执行时间与持续时间的比值。持续时间是指出从任务提交到执行结束之间的总时间。对操作系统有过了解的读者应该知道，Linux 这种多任务分时操作系统中，一个任务不太可能在执行期间始终占据处理器。操作系统会根据调度策略（linux 目前使用

CFS 调度算法) 合理安排各个任务轮流使用处理器, 每次调度会产生一次上下文切换。在此期间操作系统还需处理大量中断。因此, 一个任务的执行时间可能会很短, 但是它的持续时间会远高于此 (除非此任务是优先级最高的实时任务)。

以图 16 中的例子来说, 'ls' 的执行时间为 3.98 毫秒, 而持续为 15.58 毫秒, 处理器占用率为 0.256。在此期间, 系统共发生了 45 次上下文切换。平均每秒发生  $0.011 \times 10^6$  次。上下文切换次数的均值是上下文切换次数与任务执行时间的比值。

在多 (核) 处理器系统中, Linux 为了维持各个处理器的负载均衡, 会在特定条件下将某个任务从一个处理器迁往另外一个处理器。此时, 我们便说发生了一次处理器迁移。从图 16 上看到, ls 在执行期间没有被操作系统迁移过。

Linux 的内存管理子系统采用了分页机制。当应用程序请求的页面尚未建立、请求的页面不在内存中、或者请求的页面虽然在内存中, 但尚未建立物理地址与虚拟地址的映射关系时, 都会触发一次缺页异常 (page-fault)。内核在捕获缺页异常时, 根据异常种类进行相应的处理。另外, TLB 不命中, 页面访问权限不匹配等情况也会触发缺页异常。

内核中对 page faults (PERF\_COUNT\_SW\_PAGE\_FAULTS) 事件的精确定义是缺页异常的处理函数 do\_page\_fault() 被执行。程序 'ls' 在执行期间共触发了 320 次缺页异常。平均发生率为每秒  $0.08 \times 10^6$  次。

'cycles' 为 'ls' 程序消耗的处理器周期数。如果将被 'ls' 占据的那部分时间看作一个抽象处理器, 它的主频只需为 0.75GHz 便可以在 3.98 毫秒内完成 'ls' 命令的处理。

'instructions' 是指命令 'ls' 执行期间产生的处理器指令数。IPC (instructions per cycle) 为 0.69。IPC 是评价处理器与应用程序性能的重要指标。在 X86 这种 CSIC

处理器上，很多指令需要多个处理器周期才能执行完毕。另外，有些指令在流水线上未必能成功引退 (retired)，从而形成无效指令。长指令与无效执行越多，IPC 就越低，处理器的利用率与程序的执行效率也就越低。因此，IPC 在一定程度上，让我们对程序的执行效率有一个宏观认识。

'branches'是指程序在执行期间遇到的分支指令数。'branch-misses'则是预测错误的分支指令数。绝大多数现代处理器都具有分支预测与 OOO (Out-of-Order) 执行机制，以充分利用 CPU 内部的资源，减少流水线停顿周期。当处理器遇到分支指令时，正常来说，需要等待分支条件计算完毕才能知道后续指令流该往何处跳转。这就导致在等待分支条件计算期间，流水线上出现若干周期的停顿 (流水线 Hazard)。体系结构的经典著作《计算机体系结构：量化研究方法》上说，分支指令产生的性能影响为 10%~30%<sup>[2]</sup>，流水线越长，性能影响就越大。为了减少分支指令造成的流水线停顿，从 P5 处理器开始引入了分支预测机制。当处理器无法判断指令的跳转方向时，便通过分支预测单元选择一个最有可能的跳转方向。但是，既然是预测，就存在预测失败的可能。当分支预测失败时，会对处理器周期造成较大的浪费。在 5 发射 10 级流水线的处理器中，当分支预测的准确率为 90%时，处理器带宽会浪费 47%；而如果准确率提高到 96%，带宽浪费可降低至 26%<sup>[3]</sup>。Core i7 以及 Xeon 5500 等较新的处理器在分支预测失效时，已经无需刷新全部流水线，但错误指令加载与计算导致的无效开销依然不可小觑。这就要求我们在编写代码时，应尽量减少分支预测错误的次数。但在此之前，通过 perf stat, perf top, perf record 等工具查查分支预测失效率，以及导致分支预测失效过高的热点代码是非常有必要的。'branch misses'一行中的'\*\*\*% of all branches'即为目标程序执行期间的分支预测失效率。

## 4.2 perf stat 的参数介绍

perf stat 工具也提供了若干参数，其中一些与 perf top 类似，下面我们主要讲讲 perf stat 独有的参数。

'-e' or '--event' <event>

选择性能事件，参考 perf top 与 perf list 的相关章节。

'--filter' <filter>

配合 Tracepoints 使用，等同于 ftrace 中 filter 的概念，根据正则表达式追踪指定的函数。

'-i' or '--no-inherit'

禁止子任务继承父任务的性能计数器。类似于'perf top'中的'-i'参数，只是此处是禁止继承机制。

'-c' or '--scale'

要求底层驱动记录计数器的 run 与 enabled 时间。此选项默认打开，且不能关闭。

'-r' or '--repeat' <n>

重复执行 n 次目标程序，并给出性能指标在 n 次执行中的变化范围。命令：

```
$perf stat -r 10 ls > /dev/null
```

将重复执行 10 次"ls > /dev/null"，并给出如下结果：

```
Performance counter stats for 'ls' (10 runs):
 1.142186 task-clock           #    0.717 CPUs utilized    ( +- 13.50% )
      0 context-switches      #    0.088 K/sec             ( +-100.00% )
      0 cpu-migrations         #    0.088 K/sec             ( +-100.00% )
    268 page-faults           #    0.235 M/sec             ( +-   0.08% )
 2,001,841 cycles              #    1.753 GHz                ( +- 13.34% )
<not supported> stalled-cycles-frontend
<not supported> stalled-cycles-backend
 1,352,851 instructions        #    0.68 insns per cycle    ( +-   0.66% )
  315,470 branches             #   276.198 M/sec            ( +-   0.56% )
<not counted> branch-misses
 0.001593918 seconds time elapsed ( +- 14.09% )
```

图 17. perf stat 连续执行 10 次后给出的统计信息

与图 16 相比，图 17 上多了一系列统计信息。括号里的百分比为 n 个性能数据的标准差与数学期望的比值。这个值越大，表示各样本与平均值之间的偏差就越

大，也就是说样本的波动幅度就越大。

‘-v’ or ‘--verbose’

开启此选项后，perf stat 将显示更丰富的信息，比如打开计数器时系统报出的错误信息，各个计数器的计数值、运行时间与使能时间等信息。

‘-n’ or ‘--null’

开启这个选项后，perf stat 仅仅输出目标程序的执行时间，而不开启任何性能计数器。

执行命令：

```
$perf stat -n ls > /dev/null
```

perf stat 将给出如图 18 所示的信息。

```
root@chenggang-Latitude:perf# perf stat -n ls > /dev/null
Performance counter stats for 'ls':
0.001371681 seconds time elapsed
```

图 18. perf stat -n 的输出信息

perf stat 通过系统调用 clock\_gettime(CLOCK\_MONOTONIC, &ts)记录目标程序的执行时间，而没有利用任何性能计数器。

‘-d’ or ‘--detailed’

开启该选项后，perf stat 将给出更丰富的性能指标。

执行命令：

```
$perf stat -d ls > /dev/null
```

perf stat 将给出如图 19 所示的信息。

```
Performance counter stats for 'ls':
2.699710 task-clock # 0.730 CPUs utilized
1 context-switches # 0.000 M/sec
0 CPU-migrations # 0.000 M/sec
255 page-faults # 0.094 M/sec
492,342 cycles # 0.182 GHz
1,375,990 stalled-cycles-frontend # 279.48% frontend cycles idle
1,694,448 stalled-cycles-backend # 344.16% backend cycles idle
7,358,558 instructions # 14.95 insns per cycle
# 0.23 stalled cycles per insn
1,343,969 branches # 497.820 M/sec
27,552 branch-misses # 2.05% of all branches [91.77%]
2,019,798 L1-dcache-loads # 748.154 M/sec [54.98%]
30,831 L1-dcache-load-misses # 1.53% of all L1-dcache hits [18.12%]
<not counted> LLC-loads
<not counted> LLC-load-misses
0.003697957 seconds time elapsed
```

图 19. perf stat -d 的显示信息

与前图相比，开启'-d'选项后，perf stat 给出了'L1-dcache-loads'等 Cache 相关的性能指标。

#### '-S' or '--sync'

在执行目标程序前，先执行系统调用 sync()，将内存缓冲区中的数据写回磁盘。从而使得目标程序在执行时能够获得更干净的环境。

#### '-A' or '--no-aggr'

此选项必须与'-a'选项一起使用。开启此选项后，perf stat 将给出每个处理器上相应的信息。

执行命令：

```
$perf stat -a -A ls > /dev/null
```

perf stat 给出的信息如图 20 所示。

```
root@chenggang-Latitude:perf# perf stat -a -A ls > /dev/null
Performance counter stats for 'ls':
CPU0      5.619576 task-clock           #    1.162 CPUs utilized          (99.95%)
CPU1      5.590961 task-clock           #    1.156 CPUs utilized          (99.95%)
CPU0         7 context-switches          #    0.001 M/sec                  (99.97%)
CPU1         8 context-switches          #    0.001 M/sec                  (99.97%)
CPU0         1 cpu-migrations             #    0.178 K/sec                  (99.99%)
CPU1         1 cpu-migrations             #    0.178 K/sec                  (99.98%)
CPU0        274 page-faults              #    0.049 M/sec                  (99.99%)
CPU1        12 page-faults              #    0.049 M/sec                  (99.99%)
CPU0    4,038,957 cycles            #    0.721 GHz                    (46.98%)
CPU1    409,743 cycles            #    0.000 GHz                    (46.50%)
CPU0    <not supported> stalled-cycles-frontend
CPU1    <not supported> stalled-cycles-frontend
CPU0    <not supported> stalled-cycles-backend
CPU1    <not supported> stalled-cycles-backend
CPU0    2,744,471 instructions        #    1.23 insns per cycle          (99.80%)
CPU1    240,960 instructions        #    0.00 insns per cycle          (99.80%)
CPU0    627,569 branches             #   111.961 M/sec                 (99.88%)
CPU1    68,950 branches             #   111.961 M/sec                 (99.88%)
CPU0    18,935 branch-misses         #    5.44% of all branches        (55.70%)
CPU1    1,826 branch-misses         #    5.44% of all branches        (55.97%)

0.004835320 seconds time elapsed
```

图 20. perf stat -A 的显示信息

#### '-x' or '--field-separator'

如果希望将 perf stat 的信息导进数据库，或者希望利用某些文本分析工具对输出信息进行分析，就需要获得格式化的输出结果。'-x'参数能够满足这项需求。

如果我们希望各项输出信息之间通过';'分隔，可以采用如下命令：

```
$perf stat -x ';' ls > /dev/null
```

perf stat 的输出信息如图 21 所示。

```
root@chenggang-Latitude:perf# perf stat -x ';' ls > /dev/null
3.193227;task-clock
0;context-switches
0;cpu-migrations
274;page-faults
890086;cycles
<not supported>;stalled-cycles-frontend
<not supported>;stalled-cycles-backend
1967093;instructions
428786;branches
17183;branch-misses
```

图 21. perf stat -x 的输出信息

'-o' or '--output' <file>

可以通过此选项将 perf stat 的结果输出到指定文件。

'--append'

以追加模式，将输出信息写入输出文件。

'--pre' <command>

通过此参数，可以指定在目标程序之前执行的程序。

'--post' <command>

通过此参数，可以指定在目标程序之后执行的程序。