

# Linux 的系统级性能剖析工具-perf

## (一)

承刚

TAOBAO Kernel Team  
chenggang.qin@gmail.com

### 前言

在软件规模庞大与硬件成本相对廉价的年代，系统级性能分析和优化常被开发与测试人员忽视。但是随着虚拟化和云计算时代的到来，对计算资源按需付费的盈利模式要求系统软件与应用软件的开发者们必须对硬件资源的使用效率精益求精。在嵌入式领域更是如此。而且在互联网企业中，应用程序的性能提升对节约服务器成本来说至关重要。由于服务器集群异常庞大，软件性能的些许提升都有可能带来成本的大幅降低<sup>[1]</sup>。系统级性能优化是提升程序性能的重要步骤。但是，目前针对系统级性能优化的参考资料很稀缺，开发与测试人员往往不了解系统级性能优化的意义和方法。

系统级性能优化是指为了提高应用程序对操作系统资源与硬件资源的使用效率，或者为了提高操作系统对硬件资源的使用效率而进行的代码优化。通过提高对操作系统资源与硬件资源的利用率，使得应用程序与基础软硬件平台具有更好的交互性，往往可以显著提升应用程序的执行速度和稳定性。系统级性能优化通常包括 2 个阶段，分别是性能剖析（performance profiling）和代码优化。性能剖析阶段的目标是寻找性能瓶颈，查找引发性能问题的原因及热点代码。代码优化阶段的目标是针对具体的性能问题而优化代码与编译选项，以改善软件性能。在代码优化阶段往往需要凭借开发者的经验，编写简洁高效的代码，甚至在汇编语言级别合理使用各种指令，合理安排各种指令的执行顺序。而在性能剖析阶段，

则需要借助于现有的 profiling 工具，如 perf, VTune, Oprofile 等。其中 perf 相较于其它工具而言，具有特殊的优势。它不仅开源，而且内置于内核源码树。在如今 upstream kernel 的开发中，perf 是发展较快的领域之一。淘宝内核组也介入了 perf 的开发工作。

本文是系统级性能分析系列文章的第一篇，主要介绍性能分析工具 perf 的使用方法。后续还会有介绍系统级性能分析方法以及代码优化方法的文章。

## 第一章 Perf 简介

### 1.1 perf 的基本原理

Perf 是内置于 Linux 内核源码树中的性能剖析 (profiling) 工具。它基于事件采样原理，以性能事件为基础，支持针对处理器相关性能指标与操作系统相关性指标的性能剖析。可用于性能瓶颈的查找与热点代码的定位。

我们先通过一个例子来看看 perf 究竟能干什么。程序[code1]是一个简单的计算 pi 的计算密集型程序。很显然，[code1]的热点在函数 do\_pi()中。

```
[code1]
#include <stdio.h>
#include <math.h>
#include <sys/types.h>
#include <linux/unistd.h>

int do_pi(){
    double    mypi,h,sum,x;
    long long n,i;

    n = 5000000;
    h = 1.0/n;
    sum=0.0;

    for (i = 1; i <= n; i+=1) {
        x = h * (i - 0.5);
        sum += 4.0 / (1.0 + pow(x,2));
    }
    mypi = h * sum;

    return 0;
}

int main()
{
    printf("pid: %d\n", getpid());
    sleep(8);

    do_pi();

    return 0;
}
```

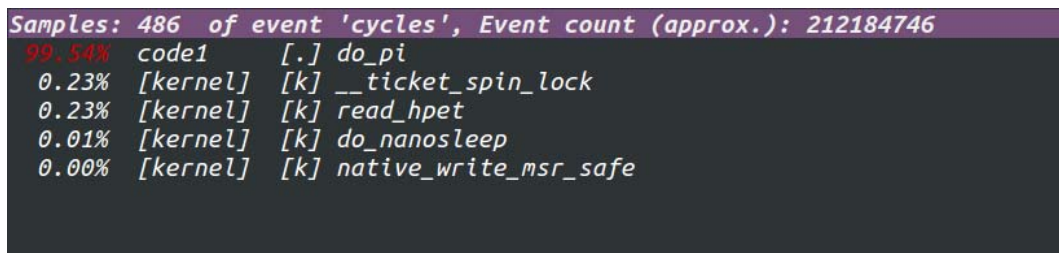
```
}
```

运行[code1]后，根据程序显示的 pid 在命令行中执行：

```
$perf top -p $pid
```

该命令利用默认的性能事件“cycles”对[code1]进行热点分析。“cycles”是处理器周期事件。这条命令能够分析出消耗处理器周期最多的代码，在处理器频率稳定的前提下，我们可以认为 perf 给出热点代码的就是消耗时间最多的代码段。

执行上述命令后，Perf 会给出如下结果：



```
Samples: 486 of event 'cycles', Event count (approx.): 212184746
99.54% code1      [.] do_pi
 0.23% [kernel]    [k] __ticket_spin_lock
 0.23% [kernel]    [k] read_hpet
 0.01% [kernel]    [k] do_nanosleep
 0.00% [kernel]    [k] native_write_msr_safe
```

图 1. Perf 对程序[code1]的分析结果

从图 1 上可以看到，在[code1]执行期间，函数 do\_pi()消耗了 99.54%的 CPU 周期，是消耗处理器周期最多的热点代码。这跟我们预想的一样。

那么 perf 是怎么做到的呢？首先，perf 会通过系统调用 sys\_perf\_event\_open 在内核中注册一个监测“cycles”事件的性能计数器。内核根据 perf 提供的信息在 PMU 上初始化一个硬件性能计数器(PMC: Performance Monitoring Counter)。PMC 随着 CPU 周期的增加而自动累加。在 PMC 溢出时，PMU 触发一个 PMI

(Performance Monitoring Interrupt) 中断。内核在 PMI 中断的处理函数中保存 PMC 的计数值，触发中断时的指令地址 (Register IP: Instruction Pointer)，当前时间戳以及当前进程的 PID, TID, comm 等信息。我们把这些信息统称为一个采样 (sample)。内核会将收集到的 sample 放入用于跟用户空间通信的 Ring Buffer。用户空间里的 perf 分析程序采用 mmap 机制从 ring buffer 中读入采样，并对其解析。perf 根据 pid, comm 等信息可以找到对应的进程。根据 IP 与 ELF 文件中的

符号表可以查到触发 PMI 中断的指令所在的函数。为了能够使 perf 读到函数名，我们的目标程序必须具备符号表。如果读者在 perf 的分析结果中只看到一串地址，而没有对应的函数名时，通常是由于在编译时利用 strip 删除了 ELF 文件中的符号表。建议读者在性能分析阶段，保留程序中的 symbol table, debug info 等信息。

根据上述的 perf 采样原理可以得知，perf 假设两次采样之间，即两次相邻的 PMI 中断之间系统执行的是同一个进程的同一个函数。这种假设会带来一定的误差，当读者感觉 perf 给出的结果不准时，不妨提高采样频率，perf 会给出更加精确的结果。

## 1.2 perf 功能概述

Perf 是一个包含 22 种子工具的工具集，功能很全面。表 1 给出了各个子工具的功能描述。

表 1. Perf 中的子工具

1	annotate	根据数据文件，注解被采样到的函数，显示指令级别的热点。
2	archive	根据数据文件中记录的 build-id，将所有被采样到的 ELF 文件打成压缩包。利用此压缩包，可以在任何机器上分析数据文件中记录的采样数据。
3	bench	Perf 中内置的 benchmark，目前包括两套针对调度器和内存管理子系统的 benchmark。
4	buildid-cache	管理 perf 的 buildid 缓存。每个 ELF 文件都有一个独一无二的 buildid。Buildid 被 perf 用来关联性能数据与 ELF

		文件。
5	buildid-list	列出数据文件中记录的所有 buildid。
6	diff	对比两个数据文件的差异。能够给出每个符号（函数）在热点分析上的具体差异。
7	evlist	列出数据文件中的所有性能事件。
8	inject	该工具读取 perf record 工具记录的事件流，并将其定向到标准输出。在被分析代码中的任何一点，都可以向事件流中注入其它事件。
9	kmem	针对内存子系统的分析工具。
10	kvm	此工具可以用来追踪、测试运行于 KVM 虚拟机上的 Guest OS。
11	list	列出当前系统支持的所有性能事件。包括硬件性能事件、软件性能事件以及检查点。
12	lock	分析内核中的加锁信息。包括锁的争用情况，等待延迟等。
13	record	收集采样信息，并将其记录在数据文件中。随后可通过其它工具对数据文件进行分析。
14	report	读取 perf record 创建的数据文件，并给出热点分析结果。
15	sched	针对调度器子系统的分析工具。
16	script	执行 perl 或 python 写的功能扩展脚本、生成脚本框架、读取数据文件中的数据信息等。

17	stat	剖析某个特定进程的性能概况，包括 CPI、Cache 丢失率等。
18	test	Perf 对当前硬件平台的测试工具。可以用此工具测试当前的硬件平台（主要是处理器型号和内部版本）是否能支持 perf 的所有功能。
19	timechart	生成一幅描述处理器与各进程状态变化的矢量图。
20	top	类似于 Linux 的 top 命令，对系统性能进行实时分析。
21	trace	strace inspired tool.
22	probe	用于定义动态检查点。

本文介绍了这 22 个工具中的绝大部分，由于精力与时间所限，忽略了几个不常用的工具，或许会在下一版中补充完全。

## 第二章 perf list 工具和性能事件

### 2.1 perf list 简介

利用 perf 剖析程序性能时，需要指定当前测试的性能事件。性能事件是指在处理器或操作系统中发生的，可能影响到程序性能的硬件事件或软件事件。比如 Cache 丢失，流水线停顿，页面交换等。这些事件会对程序的执行时间造成较大的负面影响。在优化代码时，应尽可能减少此类事件发生。因此，必须先利用 perf 等性能剖析工具查找到引发这些性能事件的热点代码以及热点指令。perf 定义的性能事件分为 3 类，分别是硬件性能事件、软件性能事件与 Tracepoint Events。

不同型号的 CPU 支持的硬件性能事件不尽相同。不同版本的内核提供的软件性能事件与 Tracepoint events 也不尽相同。因此，perf 提供了 list 工具用以查看当前软硬件平台支持的性能事件列表。list 工具的使用方法如下：

```
$perf list
```

执行该命令后，perf 将给出当前软硬件平台上支持的所有性能事件。输出结果如图 2 所示。每行后面括弧里的信息表示该事件是硬件事件、软件事件还是 Tracepoint events。

### 2.2 性能事件的属性

硬件性能事件由处理器中的 PMU 提供支持。如前文所述，perf 会对 PMI 中断发生时的 PC 寄存器进行采样。由于现代处理器的主频非常高，再加上深度流水线机制，从性能事件被触发，到处理器响应 PMI 中断，流水线上可能已处理过数百条指令。那么 PMI 中断采到的指令地址就不再是触发性能事件的那条指令的地址了，而且可能具有非常严重的偏差。为了解决这个问题，Intel 处理器通过 PEBS 机制实现了高精度事件采样。PEBS 通过硬件在计数器溢出时将处理器现场直接保存到内存（而不是在响应中断时才保存寄存器现场），从而使得 perf 能

```

List of pre-defined events (to be used in -e):
cpu-cycles OR cycles [Hardware event]
instructions [Hardware event]
cache-references [Hardware event]
cache-misses [Hardware event]
branch-instructions OR branches [Hardware event]
branch-misses [Hardware event]
bus-cycles [Hardware event]
stalled-cycles-frontend OR idle-cycles-frontend [Hardware event]
stalled-cycles-backend OR idle-cycles-backend [Hardware event]
ref-cycles [Hardware event]

cpu-clock [Software event]
task-clock [Software event]
page-faults OR faults [Software event]
context-switches OR cs [Software event]
cpu-migrations OR migrations [Software event]
minor-faults [Software event]
major-faults [Software event]
alignment-faults [Software event]
emulation-faults [Software event]

L1-dcache-loads [Hardware cache event]
L1-dcache-load-misses [Hardware cache event]
L1-dcache-stores [Hardware cache event]
L1-dcache-store-misses [Hardware cache event]
L1-dcache-prefetches [Hardware cache event]
L1-dcache-prefetch-misses [Hardware cache event]
L1-icache-loads [Hardware cache event]
L1-icache-load-misses [Hardware cache event]
L1-icache-prefetches [Hardware cache event]
L1-icache-prefetch-misses [Hardware cache event]
LLC-loads [Hardware cache event]
LLC-load-misses [Hardware cache event]
LLC-stores [Hardware cache event]
LLC-store-misses [Hardware cache event]
LLC-prefetches [Hardware cache event]
LLC-prefetch-misses [Hardware cache event]

```

图 2. Perf list 工具的输出结果

够采到真正触发性能事件的那条指令的地址，提高了采样精度。在默认条件下，perf 不使用 PEBS 机制。用户如果想要使用高精度采样，需要在指定性能事件时，在事件名后添加后缀“:p”或“:pp”。

如：

```
$perf top -e cycles:pp
```

Perf 在采样精度上定义了 4 个级别，如表 2 所示。

表 2. 性能事件的精度级别

0	无精度保证
1	采样指令与触发性能事件的指令之间的偏差为常数 (:p)
2	需要尽量保证采样指令与触发性能事件的指令之间的偏差为 0 (:pp)
3	保证采样指令与触发性能事件的指令之间的偏差必须为 0 (:ppp)



目前的 X86 处理器，包括 Intel 处理器与 AMD 处理器均仅能实现前 3 个精度级别。

除了精度级别以外，性能事件还具有其它几个属性，均可以通过“event:X”的方式予以指定。

表 3. 性能事件的属性

u	仅统计用户空间程序触发的性能事件。
k	仅统计内核触发的性能事件。
h	仅统计 Hypervisor 触发的性能事件。
G	在 KVM 虚拟机中，仅统计 Guest 系统触发的性能事件。
H	仅统计 Host 系统触发的性能事件。
p	精度级别

另外需要补充的是，perf list 工具仅列出了具有字符描述的硬件性能事件。而那些没有预定义字符描述的性能事件，也可以通过特殊方式使用。这时，就需要我们根据 CPU 的手册，通过性能事件的标号配置 PMU 的性能计数器。可以采用如下方式：

```
$perf top -e r[UMask:EventSelect]
```

举个例子，我们现在想统计所有从内存中读过数据的指令的个数，perf list 中并未预定义此事件的字符描述。通过查找 Intel 的处理器手册，我们找到了此事件的编码：

	UMask	Event
MEM_INST_RETIRED.LOADS	01	0B

便可以通过以下方式使用此事件：

```
$perf stat -e r010b ls
```