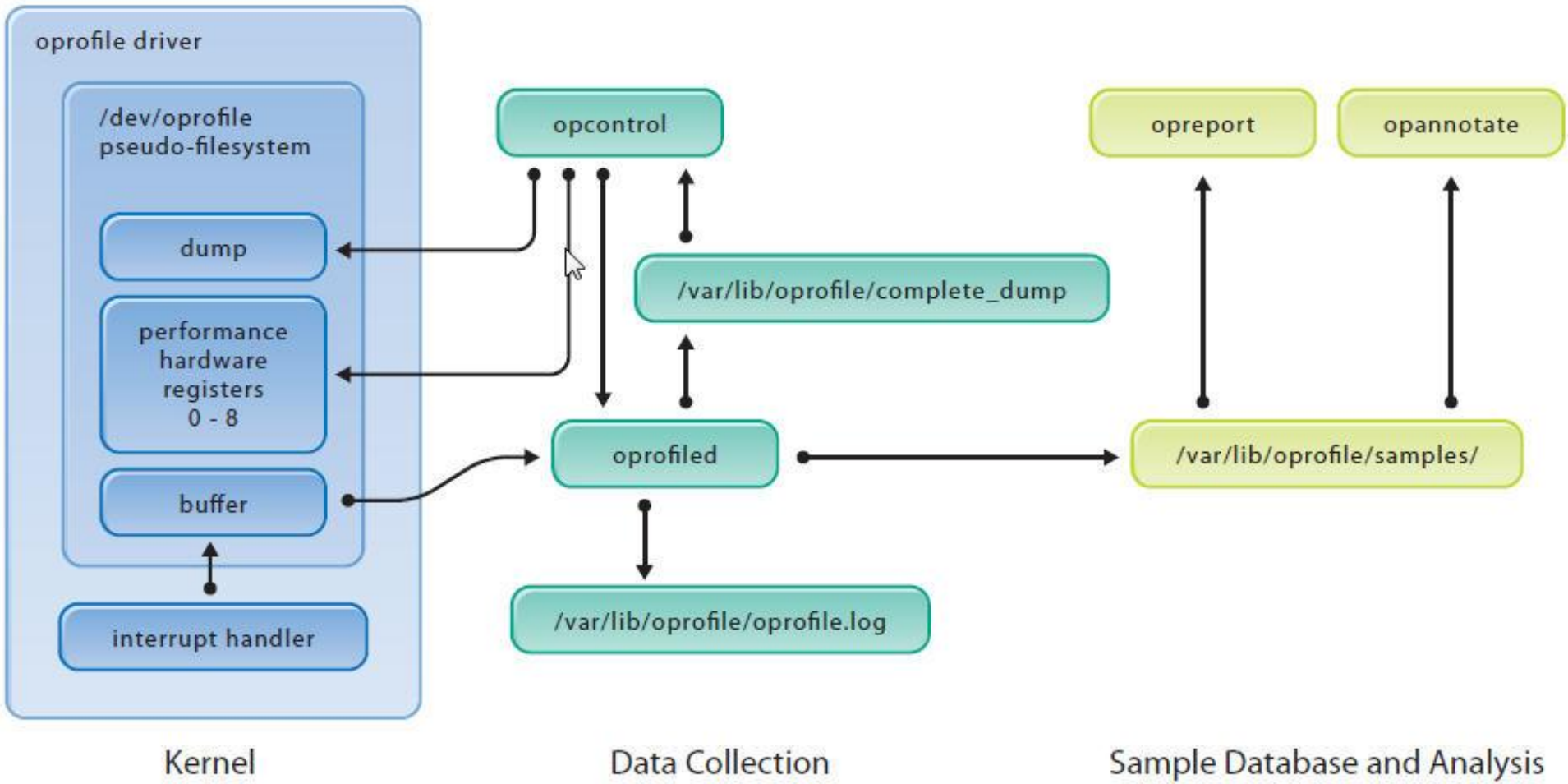
Oprofile
系统层面的性能微调工具

褚霸
核心系统数据库组
chuba@taobao.com
http://yufeng.info
2010/11/15

用途及关键特性

OProfile is a system-wide profiler for Linux systems, capable of profiling all running code at low overhead.

- 非侵入式,无需重新编译系统.
- 整个系统层面的Profile, All code is profiled.
- 利用硬件计数器.
- 低overhead.



Oprofile系统工作流图

安装(RHEL5U4)

首先需要安装内核符号信息
rpm -i kernel-debuginfo-common-2.6.18-164.el5.x86_64.rpm
rpm -i kernel-debuginfo-2.6.18-164.el5.x86_64.rpm
安装好的vmlinux在这里:
/usr/lib/debug/lib/modules/2.6.18-164.el5/vmlinux
Oprofile发行版内置的,无需安装,最新版本0.9.6
文档: man oprofile
官方网站: http://oprofile.sourceforge.net/news/

确定观察什么

Oprofile在Nehalem CPU上可以观测的事件列表:
opcontrol --list-events

设置事件:
opcontrol --setup --event=name:count:unitmask:kernel:user --event=xxxx

常用的事件有以下几种:

- CPU_CLK_UNHALTED: CPU执行时间
- LLC_MISSES: 末级Cache miss
- DTLB_MISSES: 数据TLB miss

准备我们的程序

我们的程序,包括内核驱动都需要有符号信息:

应用程序这样编译:
gcc -g foo.c ...

查看内核的导出的符号信息:
cat /proc/kallsyms

初始化Oprofile

加载oprofile内核模块
opcontrol --init
#我们对内核的取样没兴趣
opcontrol --setup --no-vmlinux

#我们需要内核的取样
opcontrol --setup --vmlinux=/usr/lib/debug/lib/modules/2.6.18-164.el5/vmlinux

采样数据

#在开始收集采样数据前回顾下我们的设置
opcontrol --status
#清除上一次采样到的数据
opcontrol --reset
#启动oprofiled守护程序,从内核中拉出采样数据
opcontrol --start

#运行我们的程序
./foo

#中途收集采样数据
opcontrol --dump
#关闭守护程序, 同时准备好采样的数据
opcontrol --shutdown

采样报告

#系统级别的
opreport --long-filenames

#模块级别的
opreport image:foo -l

#源码级别的
opannotate image:foo -s

最常用的命令

opcontrol --init
opcontrol --setup --no-vmlinux
opcontrol --status
opcontrol --start
opcontrol --dump
opcontrol --shutdown
opcontrol --reset
opreport --long-filenames
opreport image:filename -l
opannotate image:filename -s

实例演示-代码(ex1.c)

```
#include <string.h>
const char* find_str(const char* s, int l){
  const char* e = s+l;
  while(s <e) {
    if(*s == '<') return s;
    s++;
  }
}
int   main(int argc, char* argv[]) {
  char*s = argv[1];
  int   i, l;
  if(argc ==1) return -1;
  l=strlen(s);
  for(i   =   0;   i <   100000000;   i++) find_str(s, l);
  return   0;
}
```

一步步来看下实际效果

```
opcontrol --setup -e CPU_CLK_UNHALTED:6000:0:0:1
opcontrol --status
opcontrol --reset && opcontrol --start

time ./ex1 helloworld

opcontrol --shutdown
opannotate image:/home/wentong/ex1 -s
```

opreport --long-filenames

```
[root@my031019 ~]# opreport --long-filenames
CPU: Core 2, speed 2261.06 MHz (estimated)
Counted CPU_CLK_UNHALTED events (Clock cycles when not halted) with a unit mask
CPU_CLK_UNHALT...|
  samples|      %|
------------------
  1135883 96.5664 /home/wentong/ex1
    20420  1.7360 /usr/bin/oprofiled
     9576  0.8141 /lib64/libc-2.5.so
     4722  0.4014 /bin/bash
     3558  0.3025 /usr/lib/debug/lib/modules/2.6.18-164.el5/vmlinux
      976  0.0830 /lib64/ld-2.5.so
```

opannotate image:/home/wentong/ex1 -s

```
 * CPU: Core 2, speed 2261.06 MHz (estimated)
 * Counted CPU_CLK_UNHALTED events (Clock cycles when not halted) with a unit mask of 0x00 (Unhalt
 */
/*
 * Total samples for file : "/home/wentong/ex1.c"
 *
 * 1135883 100.000
 */


              :#include <string.h>
 83917  7.3878 :const char* find_str(const char* s, int l){ /* find_str total: 1009236 88.8503 */
  9568  0.8423 :  const char* e = s+l;
 49895  4.3926 :  while(s <e) {
729911 64.2593 :    if(*s == '<') return s;
135017 11.8865 :    s++;
              :  }
   928  0.0817 :}
              :int   main(int argc, char* argv[]) {  /* main total: 126647 11.1497 */
              :  char*s = argv[1];
              :  int   i, l;
              :  if(argc ==1) return -1;
              :  l=strlen(s);
126647 11.1497 :  for(i   =   0;   i <   100000000;   i++) find_str(s, l);
              :  return   0;
              :}
```

参考

http://oprofile.sourceforge.net/examples/
http://people.redhat.com/wcohen/Oprofile.pdf

谢谢大家！

Any question?