6.3 内核cpu利用率过高的问题排查

对于内核线程的CPU使用率高,比如ksoftirqd的CPU使用率高,这种内核线程,直接使用常见的性能分析工具,不太方便

那么这次就说下,如何观察这类内核线程

在说明其之前,先了解一下常见的Linux内核线程,Linux中,用户态进程的祖先,是systemd进程,PID为1,而内核线程呢? 则是2号进程kthreadd进程管理的

其中的关系为 0号为idle进程,系统创建的第一个进程,初始化1号和2号进程后,就变为了空闲任务,方便系统在空闲的时候空转

1号进程负责管理用户态线程

2号进程负责管理内核态线程

对于内核线程,只需要查看2号进程及其子进程即可

ps -f –ppid 2 -p 2

```
$ ps -f –ppid 2 -p 2
UID        PID  PPID  C STIME TTY          TIME CMD
root         2     0  0 12:02 ?        00:00:01 [kthreadd]
root         9     2  0 12:02 ?        00:00:21 [ksoftirqd/0]
root        10     2  0 12:02 ?        00:11:47 [rcu_sched]
root        11     2  0 12:02 ?        00:00:18 [migration/0]
...
root     11094     2  0 14:20 ?        00:00:00 [kworker/1:0-eve]
root     11647     2  0 14:27 ?        00:00:00 [kworker/0:2-cgr]
```

这其中有我们的主角 ksoftirqd,负责处理软中断,每个CPU都有一个

除了kthreadd 和 ksoftirqd之外,还有几个内核线程

kswapd0:用于内存回收

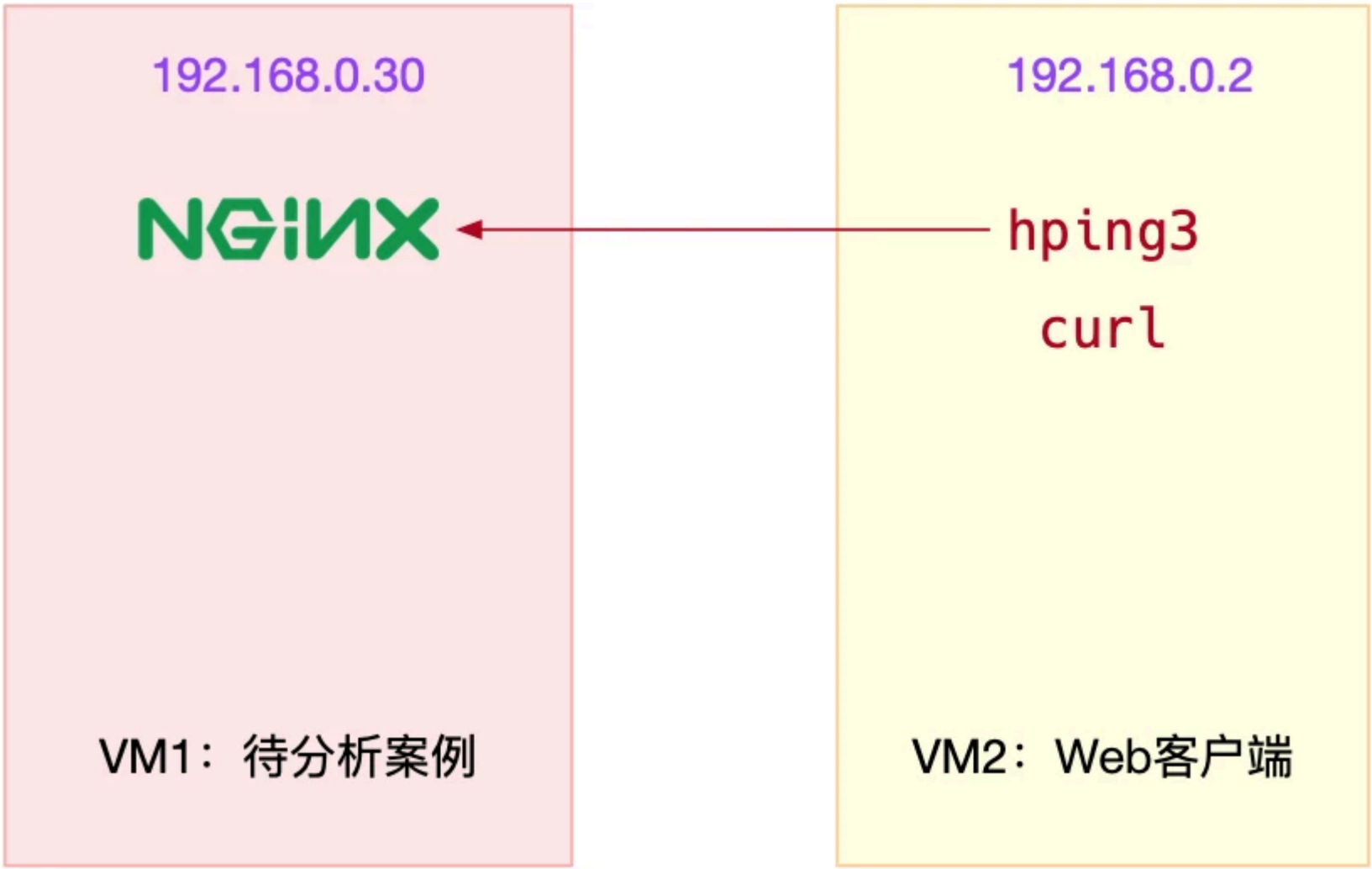
kworker: 用于执行内核工作队列

migration: 负载均衡过程中,讲进程迁移到CPU上

jbd2/sda-18 为文件系统提供日志功能,保证数据的完整性

pdflush 内存中的脏页,写入磁盘

今天的案例是利用Nginx,模拟相关的请求



我们分别运行两个Docker应用

然后进行访问Nginx的端口,确认Nginx的启动

```
$ curl http://192.168.0.30/
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
...
```

接下来,运行hPing3命令,模拟Nginx客户端

```
# -S参数表示设置TCP协议的SYN（同步序列号），-p表示目的端口为80
# -i u10表示每隔10微秒发送一个网络帧
# 注：如果你在实践过程中现象不明显，可以尝试把10调小，比如调成5甚至1
$ hping3 -S -p 80 -i u10 192.168.0.30
```

在ping命令开启后,我们查看对应的终端top

top命令获取到CPU的使用情况

```
$ top

top – 08:31:43 up 17 min,  1 user,  load average: 0.00, 0.00, 0.02

Tasks: 128 total,   1 running,  69 sleeping,   0 stopped,   0 zombie

%Cpu0  :  0.3 us,  0.3 sy,  0.0 ni, 66.8 id,  0.3 wa,  0.0 hi, 32.4 si,  0.0 st

%Cpu1  :  0.0 us,  0.3 sy,  0.0 ni, 65.2 id,  0.0 wa,  0.0 hi, 34.5 si,  0.0 st

KiB Mem :  8167040 total,  7234236 free,   358976 used,   573828 buff/cache

KiB Swap:        0 total,        0 free,        0 used.  7560460 avail Mem

PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND

9 root      20   0       0      0      0 S   7.0  0.0   0:00.48 ksoftirqd/0

18 root      20   0       0      0      0 S   6.9  0.0   0:00.56 ksoftirqd/1

2489 root      20   0  876896  38408  21520 S   0.3  0.5   0:01.50 docker-containe

3008 root      20   0   44536   3936   3304 R   0.3  0.0   0:00.09 top

1 root      20   0   78116   9000   6432 S   0.0  0.1   0:11.77 systemd

...
```

top中输出显示,两个CPU的软终端已经超过30%,而CPU使用率最高的进程,就是ksoftirqd进程

由此可以得出是网络收发导致的CPU使用率升高

对于这个内核线程,如何观察其行为呢?

如果使用pstack,或者通过/proc系统进行观测的话,效果都不是很好

那么如何观察内核线程ksoftirqd行为呢?

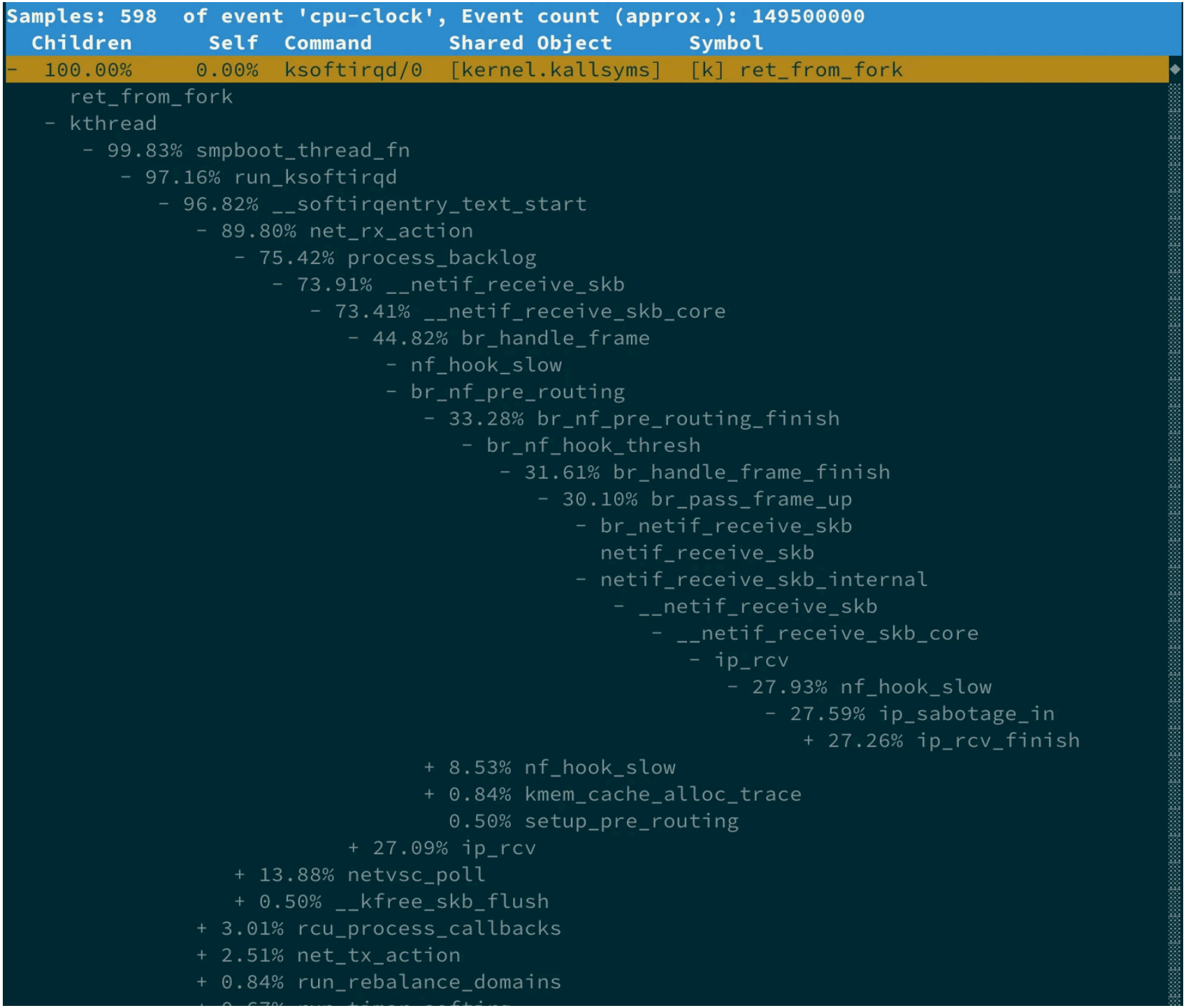
对于内核线程,可以考虑使用内核相关的性能工具,比如perf,试着分析一下进程号为9的ksoftirqd

在终端一中,执行如下的perf record命令,制定进程号为9,方便记录ksoftirqd的行为

采样30s后退出

$ perf record -a -g -p 9 — sleep 30

在之后,就可以执行perf report进行查看汇总


Samples: 598 of event 'cpu-clock', Event count (approx.): 149500000
Children Self Command Shared Object Symbol
- 100.00% 0.00% ksoftirqd/0 [kernel.kallsyms] [k] ret_from_fork
ret_from_fork
- kthread
- 99.83% smpboot_thread_fn
- 97.16% run_ksoftirqd
- 96.82% __softirqentry_text_start
- 89.80% net_rx_action
- 75.42% process_backlog
- 73.91% __netif_receive_skb
- 73.41% __netif_receive_skb_core
- 44.82% br_handle_frame
- nf_hook_slow
- br_nf_pre_routing
- 33.28% br_nf_pre_routing_finish
- br_nf_hook_thresh
- 31.61% br_handle_frame_finish
- 30.10% br_pass_frame_up
- br_netif_receive_skb
netif_receive_skb
- netif_receive_skb_internal
- __netif_receive_skb
- __netif_receive_skb_core
- ip_rcv
- 27.93% nf_hook_slow
- 27.59% ip_sabotage_in
+ 27.26% ip_rcv_finish
+ 8.53% nf_hook_slow
+ 0.84% kmem_cache_alloc_trace
0.50% setup_pre_routing
+ 27.09% ip_rcv
+ 13.88% netvsc_poll
+ 0.50% __kfree_skb_flush
+ 3.01% rcu_process_callbacks
+ 2.51% net_tx_action
+ 0.84% run_rebalance_domains

```
            + 0.67% run_timer_softirq
          + 2.17% schedule
+   100.00%     0.00%  ksoftirqd/0  [kernel.kallsyms]  [k] kthread
```

ksoftirqd中调用栈如下

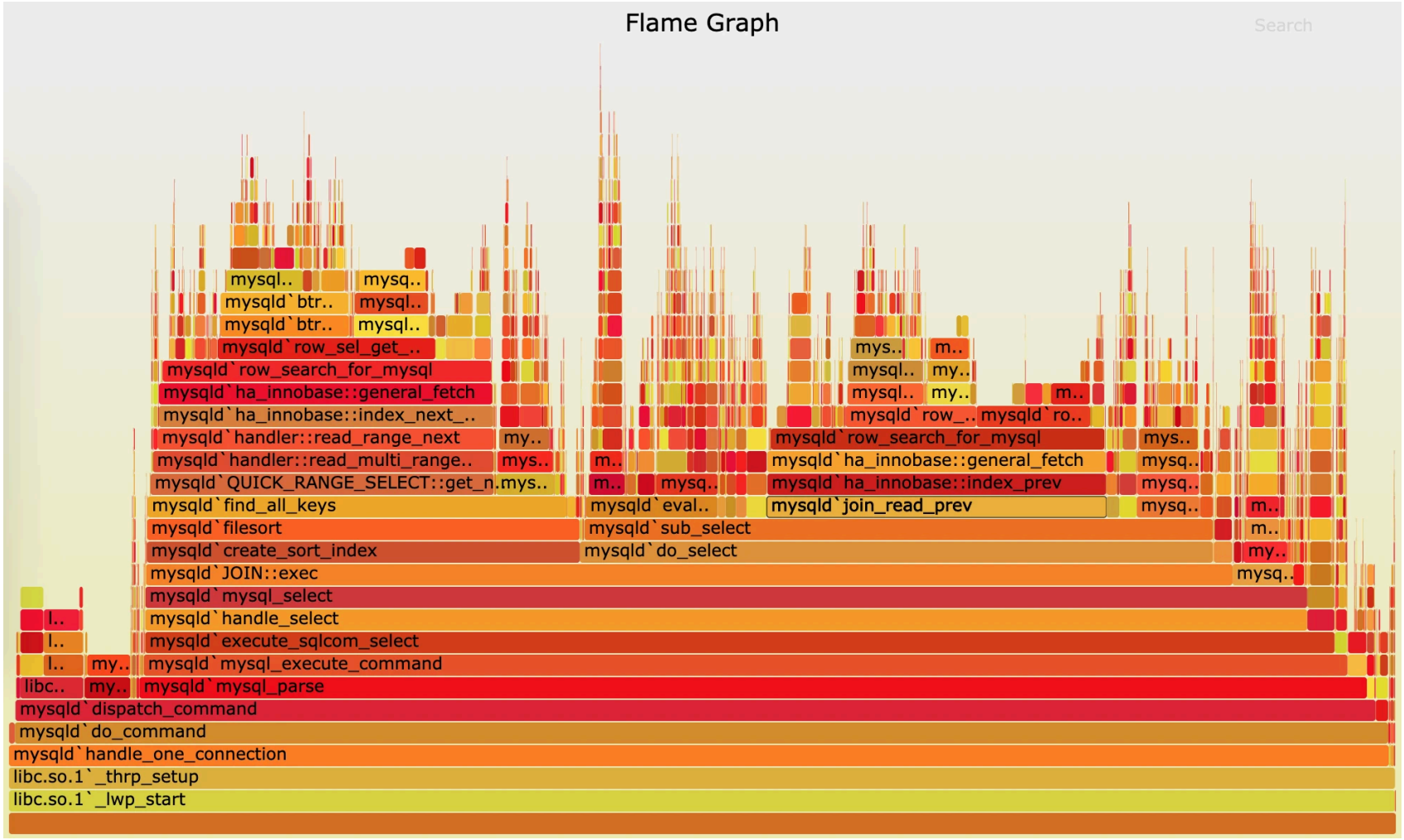
net_rx_action和netif_receive_skb 表示是接受网络包

br_handle_frame 网络的网桥经过

br_nf_pre_routing 表示网桥已经经过了netfilter的PREROUTING,这里说明有DNAT的发生

br_pass_frame_up,网桥处理之后,交给桥接的其他网卡进行处理

除此外,除了perf之外,还有这火焰图来进行汇总数据的这一种展示方式



在这个图上,横轴和纵轴是不一致的

横轴表示采样数和采样比例,一个函数占用的横轴越宽,代表执行时间越长

纵轴是调用栈,从下往上逐个展开函数,下面的函数是上面的函数的父函数

至于颜色,则没有任何的特殊含义

而且在火焰图的内部,也有着不同目标,不同性能的分析

on-CPU: CPU繁忙情况

off-CPU: CPU等待IO,锁等各种资源的阻塞情况

内存火焰图,内存分配和释放情况

热/冷火焰图,将on-CPU和off-CPU结合一下展示

差分火焰图,红色表示增长,蓝色表示衰减,方便对比

进行分析的话,需要先生成火焰图

所以我们需要一些能从perf record中生成火焰图的工具,

$ git clone https://github.com/brendangregg/FlameGraph

$ cd FlameGraph

这样,我们需要先执行perf script ,将record转换为可读的采样

执行stackcollapse-perf.pl脚本,合并调用栈

执行flamegraph.pl脚本,生成火焰图

在Linux中,可以使用管道,简化三个步骤的执行

$ perf script -i /root/perf.data | ./stackcollapse-perf.pl –all | ./flamegraph.pl > ksoftirqd.svg

然后打开浏览器,查看火焰图

那么,我们需要查看中间这一团最大的火

从下往上看,可以看出

还是net_rx_action – netif_receive_skb 到 钩子函数 到网桥

之后到ip_forward之后的,网络包进行发送

这就是利用火焰图,查看整体网络调用的详情

火焰图的作用中,包含去分析Nginx MySQL等场景性能问题

admin 2021-06-16 Linux性能调优 Leave a Comment