前言

软中断基本原理，可参考这篇博客：https://www.cnblogs.com/poloyy/p/13435519.html

中断

- 一种异步的事件处理机制，用来提供系统的并发处理能力
- 当中断事件发生，会触发执行中断处理程序
- 中断处理程序分为上半部和下半部
- **上半部：**硬中断，快速处理中断
- **下半部：**软中断，用来异步处理上半部未完成的工作

软中断

- 每个 CPU 都对应一个软中断内核线程，名字是 ksoftirqd/CPU 编号
- 当软中断事件的频率过高时，内核线程也会因为 CPU 使用率过高而导致软中断处理不及时，进而引发网络收发延迟，调度缓慢等性能问题

软中断频率过高案例

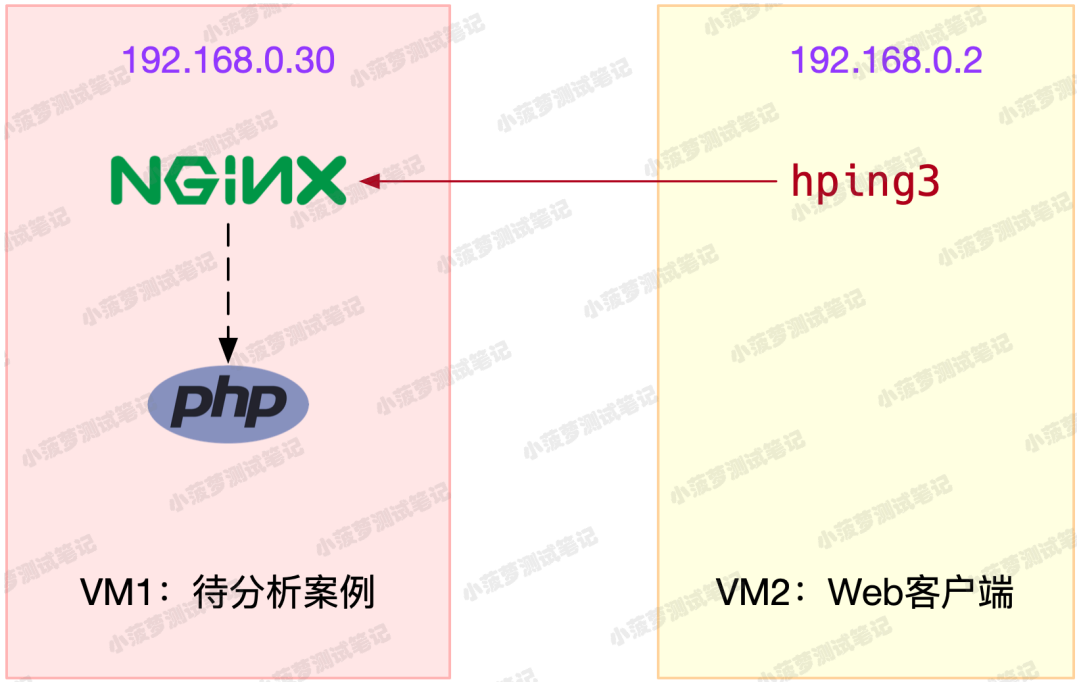
系统配置

Ubuntu 18.04， 2 CPU，2GB 内存，共两台虚拟机

三个工具

- sar：是一个**系统活动报告工具**，既可以实时查看系统的当前活动，又可以配置保存和报告 历史统计数据。
- hping3：是一个可以**构造 TCP/IP 协议数据包的工具**，可以对系统进行安全审计、防火墙 测试等。
- tcpdump：是一个常用的**网络抓包工具**，常用来分析各种网络问题

虚拟机关系



通过 docker 运行案例

在 VM1 中执行命令

```
docker run -itd --name=nginx -p 80:80 nginx
```

通过 curl 确认 Nginx 正常启动

在 VM2 中执行命令

```
curl http://172.20.72.58/
```

通过 hping3 模拟 Nginx 的客户端请求

在 VM2 中执行命令

```
hping3 -S -p 80 -i u100 172.20.72.58
```

- **-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**：u100 表示每隔 100 微秒发送一个网络帧

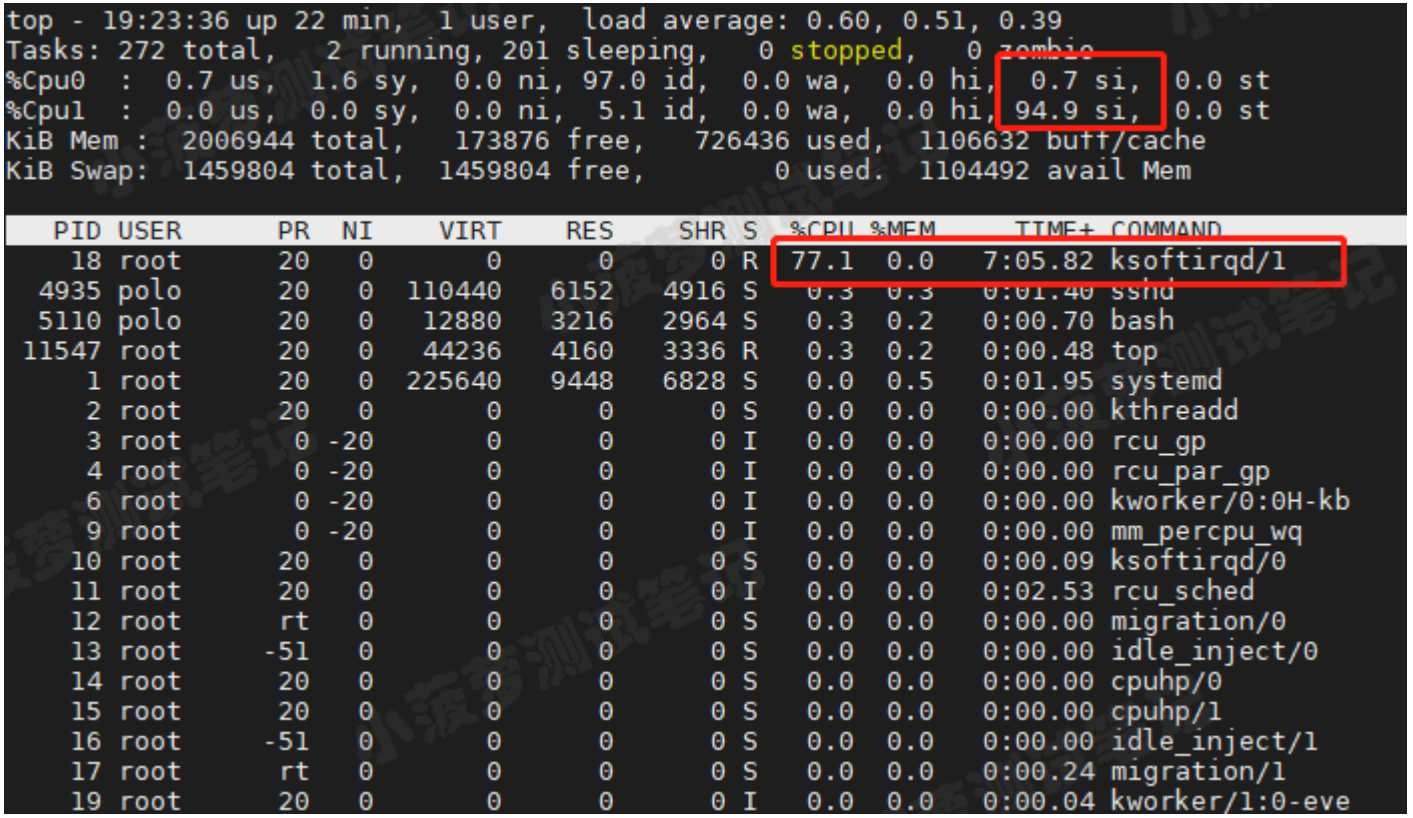
回到 VM1

感觉系统响应明显变慢了，即便只 是在终端中敲几个回车，都得很久才能得到响应

分析系统为什么会响应变慢

以下命令均在 VM1 中执行

通过 top 命令查看系统资源使用情况



```
top - 19:23:36 up 22 min,  1 user,  load average: 0.60, 0.51, 0.39
Tasks: 272 total,   2 running, 201 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.7 us,  1.6 sy,  0.0 ni, 97.0 id,  0.0 wa,  0.0 hi,  0.7 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,  5.1 id,  0.0 wa,  0.0 hi, 94.9 si,  0.0 st
KiB Mem :  2006944 total,   173876 free,   726436 used,  1106632 buff/cache
KiB Swap:  1459804 total,  1459804 free,        0 used.  1104492 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
   18 root      20   0       0      0      0 R  77.1  0.0   7:05.82 ksoftirqd/1
 4935 polo      20   0  110440   6152   4916 S   0.3  0.3   0:01.40 sshd
 5110 polo      20   0   12880   3216   2964 S   0.3  0.2   0:00.70 bash
11547 root      20   0   44236   4160   3336 R   0.3  0.2   0:00.48 top
    1 root      20   0  225640   9448   6828 S   0.0  0.5   0:01.95 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthreadd
    3 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 rcu_par_gp
    6 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:0H-kb
    9 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 mm_percpu_wq
   10 root      20   0       0      0      0 S   0.0  0.0   0:00.09 ksoftirqd/0
   11 root      20   0       0      0      0 I   0.0  0.0   0:02.53 rcu_sched
   12 root      rt   0       0      0      0 S   0.0  0.0   0:00.00 migration/0
   13 root     -51   0       0      0      0 S   0.0  0.0   0:00.00 idle_inject/0
   14 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/0
   15 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/1
   16 root     -51   0       0      0      0 S   0.0  0.0   0:00.00 idle_inject/1
   17 root      rt   0       0      0      0 S   0.0  0.0   0:00.24 migration/1
   19 root      20   0       0      0      0 I   0.0  0.0   0:00.04 kworker/1:0-eve
```

1. 系统 CPU 使用率（用户态 us 和内核态 sy ）并不高
2. 平均负载适中，只有 2 个 R 状态的进程，无僵尸进程
3. 但是软中断进程1号（ksoftirqd/1）的 CPU 使用率偏高，而且处理软中断的 CPU 占比已达到 94
4. 此外，并无其他异常进程
5. 可以猜测，软中断就是罪魁祸首

确认是什么类型的软中断

观察 `/proc/softirqs` 文件的内容，就能知道各种**软中断类型的次数**

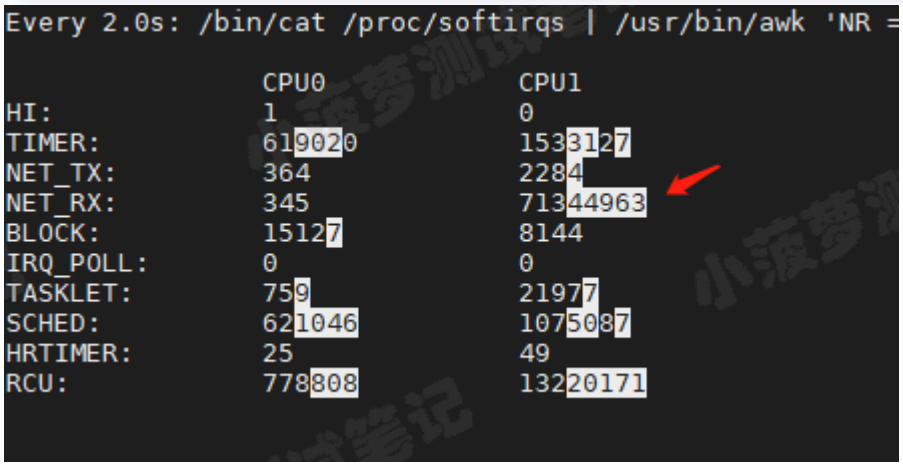
这里的各类软中断次数，又是什么时间段里的次数呢?

- 它是系统运行以来的**累积中断次数**
- 所以直接查看文件内容，得到的只是累积中断次数，对这里的问题并没有直接参考意义
- **中断次数的变化速率**才是我们需要关注的

通过 watch 动态查看命令输出结果

因为我的机器是两核，如果直接读取 `/proc/softirqs` 会打印 128 核的信息，但对于我来说，只要看前面两核的信息足以，所以需要写提取关键数据

```
watch -d "/bin/cat /proc/softirqs | /usr/bin/awk 'NR == 1{printf \"
%-15s %-15s %-15s\n\",\" \",\$1,\$2}; NR > 1{printf \"
%-15s %-15s %-15s\n\",\$1,\$2,\$3}'"
```



结果分析

- TIMER（定时中断）、 NET_RX（网络接收）、SCHED（内核调度）、RCU（RCU 锁）等这几个软中断都在不停变化
- 而 **NET_RX**，就是网络数据包**接收软中断**的**变化速率最快**
- 其他几种类型的软中断，是保证 Linux 调度、时钟、临界区保护这些正常工作所必需的，所以有变化时正常的

通过 sar 查看系统的网络收发情况

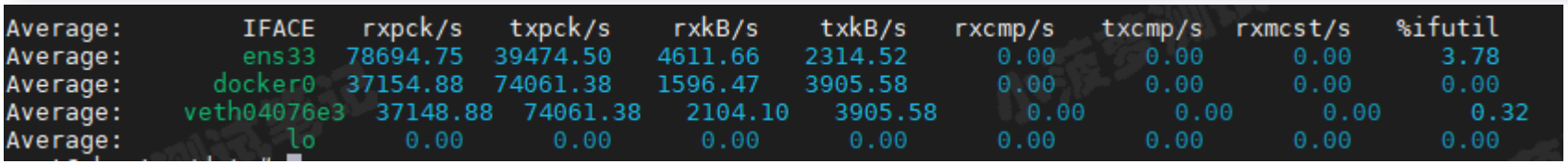
上面确认了从网络接收的软中断入手，所以第一步应该要看下系统的网络接收情况

sar 的好处

- 不仅可以观察网络收发的吞吐量（BPS，每秒收发的字节数）
- 还可以观察网络收发的 PPS（每秒收发的网络帧数）

执行 sar 命令

```
1   sar -n DEV 1
```



Average:	IFACE	rxpck/s	txpck/s	rxkB/s	txkB/s	rxcmp/s	txcmp/s	rxmcst/s	%ifutil
Average:	ens33	78694.75	39474.50	4611.66	2314.52	0.00	0.00	0.00	3.78
Average:	docker0	37154.88	74061.38	1596.47	3905.58	0.00	0.00	0.00	0.00
Average:	veth04076e3	37148.88	74061.38	2104.10	3905.58	0.00	0.00	0.00	0.32
Average:	lo	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

- 第二列：IFACE 表示网卡
- 第三、四列：rxpck/s 和 txpck/s 分别表示每秒接收、发送的网络帧数**【PPS】**
- 第五、六列：rxkB/s 和 txkB/s 分别表示每秒接收、发送的千字节数**【BPS】**

结果分析

对网卡 ens33 来说

- 每秒接收的网络帧数比较大，几乎达到 8w，而发送的网络帧数较小，只有接近 4w
- 每秒接收的千字节数只有 4611 KB，发送的千字节数更小，只有2314 KB

docker0 和 veth04076e3

- 数据跟 ens33 基本一致只是发送和接收相反，发送的数据较大而接收的数据较小
- 这是 **Linux 内部网桥转发**导致的，暂且不用深究，只要知道这是系统把 ens33 收到的包转发给 Nginx 服务即可

异常点

- 前面说到是网络数据包接收软中断的问题，那就重点看 ens33
- 接收的 PPS 达到 8w，但接收的 BPS 只有 5k 不到，**网络帧**看起来是比较小的
- 4611 * 1024 / 78694 = 64 字节，说明平均每个网络帧只有 60 字节，这显然是很小的网络帧，也就是常说的**小包问题**

灵魂拷问

如何知道这是一个什么样的网络帧，它又是从哪里发过来的呢？

通过 tcpdump 抓取网络包

已知条件

Nginx 监听在 80 端口， 它所提供的 HTTP 服务是基于 TCP 协议的

执行 tcpdump 命令

```
tcpdump -i ens33 -n tcp port 80
```

- **-i ens33：**只抓取 ens33 网卡
- **-n：**不解析协议名和主机名
- **tcp port 80：**表示只抓取 tcp 协议并且端口号为 80 的网络帧

```
22:02:06.434365 IP 172.20.72.59.52177 > 172.20.72.58.80: Flags [S], seq 926957474, win 512, length 0
22:02:06.434442 IP 172.20.72.59.52178 > 172.20.72.58.80: Flags [S], seq 1631945002, win 512, length 0
22:02:06.434458 IP 172.20.72.59.52179 > 172.20.72.58.80: Flags [S], seq 1776271251, win 512, length 0
22:02:06.434462 IP 172.20.72.59.51539 > 172.17.0.2.80: Flags [R], seq 1950670260, win 0, length 0
22:02:06.434462 IP 172.20.72.59.51540 > 172.17.0.2.80: Flags [R], seq 56764012, win 0, length 0
22:02:06.434463 IP 172.20.72.59.51541 > 172.20.72.58.80: Flags [R], seq 1932888892, win 0, length 0
22:02:06.434466 IP 172.20.72.59.52180 > 172.20.72.58.80: Flags [S], seq 1821904557, win 512, length 0
22:02:06.434468 IP 172.20.72.59.51542 > 172.17.0.2.80: Flags [R], seq 1387214814, win 0, length 0
22:02:06.434483 IP 172.20.72.59.51547 > 172.17.0.2.80: Flags [R], seq 591266924, win 0, length 0
22:02:06.434488 IP 172.20.72.59.52183 > 172.20.72.58.80: Flags [S], seq 1698473271, win 512, length 0
22:02:06.434493 IP 172.20.72.59.51551 > 172.17.0.2.80: Flags [R], seq 368961372, win 0, length 0
22:02:06.434498 IP 172.20.72.59.52185 > 172.20.72.58.80: Flags [S], seq 858378268, win 512, length 0
22:02:06.434503 IP 172.20.72.59.51556 > 172.20.72.58.80: Flags [R], seq 1255707293, win 0, length 0
22:02:06.434506 IP 172.20.72.59.51557 > 172.20.72.58.80: Flags [R], seq 998337708, win 0, length 0
22:02:06.434510 IP 172.20.72.59.51559 > 172.20.72.58.80: Flags [R], seq 1296865613, win 0, length 0
22:02:06.434514 IP 172.20.72.59.51560 > 172.17.0.2.80: Flags [R], seq 1257561450, win 0, length 0
22:02:06.434523 IP 172.20.72.59.51561 > 172.20.72.58.80: Flags [R], seq 1104302410, win 0, length 0
22:02:06.434526 IP 172.20.72.59.51562 > 172.20.72.58.80: Flags [R], seq 1942047880, win 0, length 0
22:02:06.434528 IP 172.20.72.59.52189 > 172.20.72.58.80: Flags [S], seq 1060292925, win 512, length 0
22:02:06.434537 IP 172.20.72.59.52190 > 172.20.72.58.80: Flags [S], seq 1510527426, win 512, length 0
22:02:06.434539 IP 172.20.72.59.51563 > 172.20.72.58.80: Flags [R], seq 1667384698, win 0, length 0
22:02:06.434541 IP 172.20.72.59.52191 > 172.20.72.58.80: Flags [S], seq 1283115994, win 512, length 0
22:02:06.434543 IP 172.20.72.59.52192 > 172.20.72.58.80: Flags [S], seq 1174115258, win 512, length 0
```

172.20.72.59.52195 > 172.20.72.58.80

- 表示网络帧从 172.20.72.59 的 52195 端口发 送到 172.20.72.58 的 80 端口

- 也就是从运行 hping3 机器的 52195 端口发送网络帧， 目的为 Nginx 所在机器的 80 端口

Flags [S]

表示这是一个 SYN 包

性能分析结果

结合 **sar** 命令发现的 PPS 接近 4w 的现象，可以认为这就是从 172.20.72.59 这个地址发送过来的 **SYN FLOOD 攻击**

解决 SYN FLOOD 问题

从交换机或者硬件防火墙中**封掉来源 IP**，这样 SYN FLOOD 网络帧就不会发送到服务器中

后续的期待

至于 SYN FLOOD 的原理和更多解决思路在后面会讲到哦

分析的整体思路

1. 系统出现卡顿，执行命令，响应也会变慢
2. 通过 **top** 查看系统资源情况
3. 发现 CPU 使用率（**us 和 sy**）均不高，平均负载适中，没有超 CPU 核数的运行状态的进程，也没有僵尸进程
4. 但是发现处理软中断的 CPU 占比（**si**）较高，在进程列表也可以看到软中断进程 CPU 使用率偏高，猜测是软中断导致系统变卡顿的主要原因

5. 通过 **/proc/sorfirqs** 查看软中断类型和变化频率，发现直接 cat 的话会打印 128 个核的信息，但只想要两个核的信息
6. 所以结合 **awk** 进行过滤，再通过 **watch** 命令可以动态输出查看结果
7. 发现有多个软中断类型在变化，重点是 **NET_RX** 变化频率超高，而且幅度也很大，它是**网络数据包接收软中断**，暂且认为它是问题根源
8. 既然跟网络有关系，可以先通过 **sar** 命令查看系统网络接收和发送的整体情况
9. 然后可以看到接收的 PPS 会比接收的 BPS 大很多，做下运算，发现网络帧会非常小，也就是常说的**小包**问题
10. 接下来，通过 **tcpdump** 抓取 80端口的 tcp 协议网络包，会发现大量来自 VM2 发送的 SYN 包，结合 sar 命令，确认是 SYN FLOOD 攻击

__EOF__

posted @ 2020-08-11 14:11 小菠萝测试笔记 阅读(5439) 评论(0) 编辑 收藏 举报