Linux网络包接收过程的监控与调优

原创 张彦飞allen 开发内功修炼 2020-10-19 09:18

上一篇文章中《图解Linux网络包接收过程》，我们梳理了在Linux系统下一个数据包被接收的整个过程。Linux内核对网络包的接收过程大致可以分为接收到RingBuffer、硬中断处理、ksoftirqd软中断处理几个过程。其中在ksoftirqd软中断处理中，把数据包从RingBuffer中摘下来，送到协议栈的处理，再之后送到用户进程socket的接收队列中。

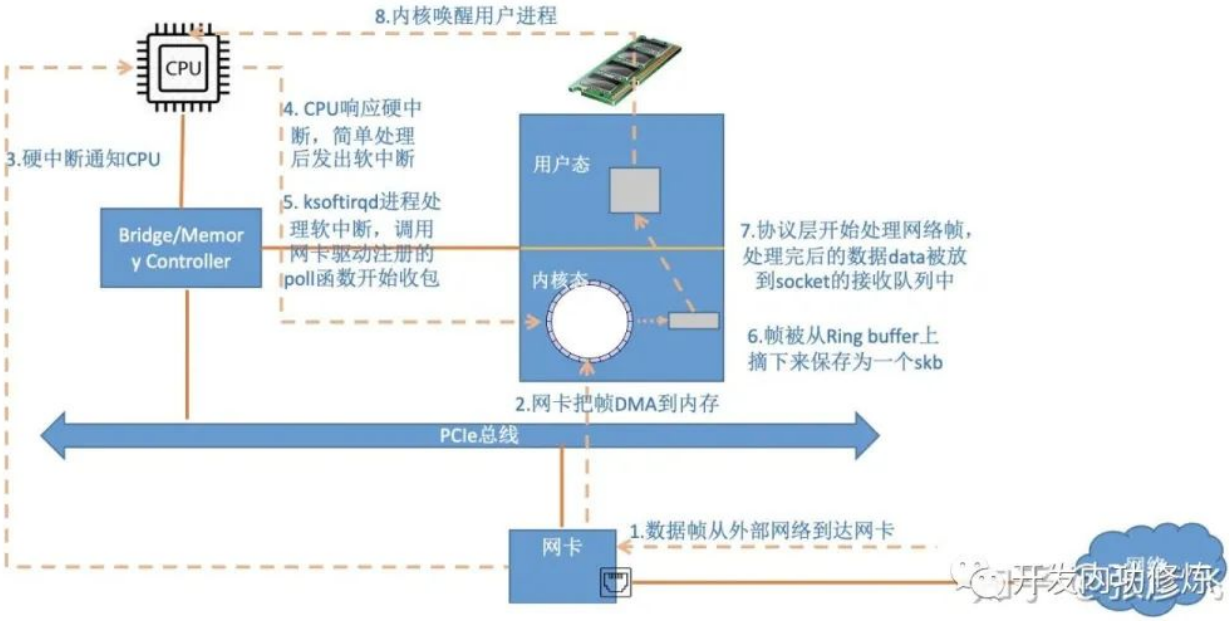

图1 Linux内核接收网络包过程

理解了Linux工作原理之后，还有更重要的两件事情。第一是动手监控，会实际查看网络包接收的整体情况。第二是调优，当你的服务器有问题的时候，你能找到瓶颈所在，并会利用内核开放的参数进行调节。

一 先说几个工具

在正式内容开始之前，我们先来了解几个Linux下监控网卡时可用的工具。

1）ethtool

首先第一个工具就是我们在上文中提到的 `ethtool`，它用来查看和设置网卡参数。这个工具其实本身只是提供几个通用接口，真正的实现是都是在网卡驱动中的。正因为该工具是由驱动直接实现的，所以个人觉得它最重要。

该命令比较复杂，我们选几个今天能用到的说

- `-i` 显示网卡驱动的信息，如驱动的名称、版本等
- `-S` 查看网卡收发包的统计情况
- `-g/-G` 查看或者修改RingBuffer的大小
- `-l/-L` 查看或者修改网卡队列数
- `-c/-C` 查看或者修改硬中断合并策略

实际查看一下网卡驱动：

```
# ethtool -i eth0
driver: ixgbe
……
```

这里看到我的机器上网卡驱动程序是ixgbe。有了驱动名称，就可以在源码中找到对应的代码了。对于 `ixgbe` 来说，其驱动的源代码位于 `drivers/net/ethernet/intel/ixgbe` 目录下。ixgbe_ethtool.c`下都是实现的供ethtool使用的相关函数，如果ethtool哪里有搞不明白的，就可以通过这种方式查找到源码来读。另外我们前文《图解Linux网络包接收过程》里提到的NAPI收包时的poll回调函数，启动网卡时的open函数都是在这里实现的。

2）ifconfig

网络管理工具ifconfig不只是可以为网卡配置ip，启动或者禁用网卡，也包含了一些网卡的统计信息。

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.162.42.51  netmask 255.255.248.0  broadcast 10.162.47.255
        inet6 fe80::6e0b:84ff:fed5:88d1  prefixlen 64  scopeid 0x20<link>
        ether 6c:0b:84:d5:88:d1  txqueuelen 1000  (Ethernet)
        RX packets 2953454  bytes 414212810 (395.0 MiB)
        RX errors 0  dropped 4636605  overruns 0  frame 0
        TX packets 127887  bytes 82943405 (79.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

- RX packets：接收的总包数
- RX bytes：接收的字节数
- RX errors：表示总的收包的错误数量
- RX dropped：数据包已经进入了 Ring Buffer，但是由于其它原因导致的丢包
- RX overruns：表示了 fifo 的 overruns，这是由于 Ring Buffer不足导致的丢包

3）伪文件系统/proc

Linux 内核提供了 /proc 伪文件系统，通过/proc可以查看内核内部数据结构、改变内核设置。我们先跑一下题，看一下这个伪文件系统里都有啥：

- `/proc/sys` 目录可以查看或修改内核参数
- `/proc/cpuinfo` 可以查看CPU信息
- `/proc/meminfo` 可以查看内存信息
- `/proc/interrupts` 统计所有的硬中断
- `/proc/softirqs` 统计的所有的软中断信息
- `/proc/slabinfo` 统计了内核数据结构的slab内存使用情况
- **`/proc/net/dev` 可以看到一些网卡统计数据**

详细聊下伪文件 `/proc/net/dev`，通过它可以看到内核中对网卡的一些相关统计。包含了以下信息：

- bytes: 发送或接收的数据的总字节数
- packets: 接口发送或接收的数据包总数
- errs: 由设备驱动程序检测到的发送或接收错误的总数
- drop: 设备驱动程序丢弃的数据包总数
- fifo: FIFO缓冲区错误的数量
- frame: The number of packet framing errors.（分组帧错误的数量）
- colls: 接口上检测到的冲突数

所以，伪文件 `/proc/net/dev` 也可以作为我们查看网卡工作统计数据的工具之一。

4）伪文件系统sysfs

sysfs和/proc类似，也是一个伪文件系统，但是比proc更新，结构更清晰。其中的`/sys/class/net/eth0/statistics/`也包含了网卡的统计信息。

```
# cd /sys/class/net/eth0/statistics/
# grep . * | grep tx
tx_aborted_errors:0
tx_bytes:170699510
tx_carrier_errors:0
tx_compressed:0
tx_dropped:0
tx_errors:0
tx_fifo_errors:0
tx_heartbeat_errors:0
tx_packets:262330
tx_window_errors:0
```

好了，简单了解过这几个工具以后，让我们正式开始今天的行程。

二 RingBuffer监控与调优

前面我们看到，当网线中的数据帧到达网卡后，第一站就是RingBuffer（网卡通过DMA机制将数据帧送到RingBuffer中）。因此我们第一个要监控和调优的就是网卡的RingBuffer，我们使用`ethtool`来查看一下：

```
# ethtool -g eth0
Ring parameters for eth0:
Pre-set maximums:
RX:     4096
RX Mini:    0
RX Jumbo:   0
TX:     4096
Current hardware settings:
RX:     512
RX Mini:    0
RX Jumbo:   0
TX:     512
```

这里看到我手头的网卡设置RingBuffer最大允许设置到4096，目前的实际设置是512。

> 这里有一个小细节，ethtool查看到的是实际是Rx bd的大小。Rx bd位于网卡中，相当于一个指针。RingBuffer在内存中，Rx bd指向RingBuffer。Rx bd和RingBuffer中的元素是一一对应的关系。在网卡启动的时候，内核会为网卡的Rx bd在内存中分配RingBuffer，并设置好对应关系。

在Linux的整个网络栈中，RingBuffer起到一个任务的收发中转站的角色。对于接收过程来讲，网卡负责往RingBuffer中写入收到的数据帧，ksoftirqd内核线程负责从中取走处理。只要ksoftirqd线程工作的足够快，RingBuffer这个中转站就不会出现问题。但是我们设想一下，假如某一时刻，瞬间来了特别多的包，而ksoftirqd处理不过来了，会发生什么？这时RingBuffer可能瞬间就被填满了，后面再来的包网卡直接就会丢弃，不做任何处理！

那我们怎么样能看一下，我们的服务器上是否有因为这个原因导致的丢包呢？前面我们介绍的四个工具都可以查看这个丢包统计，拿 `ethtool` 来举例：

```
# ethtool -S eth0
……
rx_fifo_errors: 0
tx_fifo_errors: 0
```

rx_fifo_errors如果不为0的话（在 ifconfig 中体现为 overruns 指标增长），就表示有包因为RingBuffer装不下而被丢弃了。那么怎么解决这个问题呢？很自然首先我们想到的是，加大RingBuffer这个“中转仓库”的大小。通过ethtool就可以修改。

```
# ethtool -G eth1 rx 4096 tx 4096
```

这样网卡会被分配更大一点的”中转站“，可以解决偶发的瞬时的丢包。不过这种方法有个小副作用，那就是排队的包过多会增加处理网络包的延时。所以另外一种解决思路更好，那就是让内核处理网络包的速度更快一些，而不是让网络包傻傻地在RingBuffer中排队。怎么加快内核消费RingBuffer中任务的速度呢，别着急，我们继续往下看...

三 硬中断监控与调优

在数据被接收到RingBuffer之后，下一个执行就是就是硬中断的发起。我们先来查看硬中断，然后再聊下怎么优化。

1）监控

硬中断的情况可以通过内核提供的伪文件 `/proc/interrupts` 来进行查看。

```
$ cat  /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
 0:         34          0          0          0   IO-APIC-edge      timer
 ......
 27:       351          0          0 1109986815   PCI-MSI-edge      virtio1-input.0
 28:      2571          0          0          0   PCI-MSI-edge      virtio1-output.0
 29:         0          0          0          0   PCI-MSI-edge      virtio2-config
 30:   4233459 1986139461     244872     474097   PCI-MSI-edge      virtio2-input.0
 31:         3          0          2          0   PCI-MSI-edge      virtio2-output.0
```

上述结果是我手头的一台虚机的输出结果。上面包含了非常丰富的信息，让我们一一道来：

- 网卡的输入队列 `virtio1-input.0` 的中断号是27
- 27号中断都是由CPU3来处理的
- 总的中断次数是1109986815。

这里有两个细节我们需要关注一下。

（1）为什么输入队列的中断都在CPU3上呢？

这是因为内核的一个配置，在伪文件系统中可以查看到。

```
#cat /proc/irq/27/smp_affinity
8
```

`smp_affinity` 里是CPU的亲和性的绑定，8是二进制的1000,第4位为1，代表的就是第4个CPU核心-CPU3.

（2）对于收包来过程来讲，硬中断的总次数表示的是Linux收包总数吗？
不是，硬件中断次数不代表总的网络包数。第一网卡可以设置中断合并，多个网络帧可以只发起一次中断。第二NAPI 运行的时候会关闭硬中断，通过poll来收包。

2）多队列网卡调优

现在的主流网卡基本上都是支持多队列的，我们可以通过将不同的队列分给不同的CPU核心来处理，从而加快Linux内核处理网络包的速度。这是最为有用的一个优化手段。

每一个队列都有一个中断号，可以独立向某个CPU核心发起硬中断请求，让CPU来 poll 包。通过将接收进来的包被放到不同的内存队列里，多个CPU就可以同时分别向不同的队列发起消费了。这个特性叫做RSS（Receive Side Scaling，接收端扩展）。通过 ethtool 工具可以查看网卡的队列情况。

```
# ethtool -l eth0
Channel parameters for eth0:
Pre-set maximums:
RX:     0
TX:     0
Other:      1
Combined:   63
Current hardware settings:
RX:     0
TX:     0
Other:      1
Combined:   8
```

上述结果表示当前网卡支持的最大队列数是63，当前开启的队列数是8。对于这个配置来讲，最多同时可以有8个核心来参与网络收包。如果你想提高内核收包的能力，直接简单加大队列数就可以了，这比加大RingBuffer更为有用。因为加大RingBuffer只是给个更大的空间让网络帧能继续排队，而加大队列数则能让包更早地被内核处理。 ethtool 修改队列数量方法如下：

```
#ethtool -L eth0 combined 32
```

我们前文说过，硬中断发生在哪一个核上，它发出的软中断就由哪个核来处理。所有通过加大网卡队列数，这样硬中断工作、软中断工作都会有更多的核心参与进来。

> 每一个队列都有一个中断号，每一个中断号都是绑定在一个特定的CPU上的。如果你不满意某一个中断的CPU绑定，可以通过修改/proc/irq/{中断号}/smp_affinity来实现。

一般处理到这里，网络包的接收就没有大问题了。但如果你有更高的追求，或者是说你并没有更多的CPU核心可以参与进来了，那怎么办？放心，我们也还有方法提高单核的处理网络包的接收速度。

3）硬中断合并

先来讲一个实际中的例子，假如你是一位开发同学，和你对口的产品经理一天有10个小需求需要让你帮忙来处理。她对你有两种中断方式：

- 第一种：产品经理想到一个需求，就过来找你，和你描述需求细节，然后让你帮你来改
- 第二种：产品经理想到需求后，不来打扰你，等攒够5个来找你一次，你集中处理

我们现在不考虑及时性，只考虑你的工作整体效率，你觉得那种方案下你的工作效率会高呢？或者换句话说，你更喜欢哪一种工作状态呢？很明显，只要你是一个正常的开发，都会觉得第二种方案更好。对人脑来讲，频繁的中断会打乱你的计划，你脑子里刚才刚想到一半技术方案可能也就废了。当产品经理走了以后，你再想捡起来刚被中断之的工作的时候，很可能得花点时间回忆一会儿才能继续工作。

对于CPU来讲也是一样，CPU要做一件新的事情之前，要加载该进程的地址空间，load进程代码，读取进程数据，各级别cache要慢慢热身。因此如果能适当降低中断的频率，多攒几个包一起发出中断，对提升CPU的工作效率是有帮助的。所以，网卡允许我们对硬中断进行合并。

现在我们来看一下网卡的硬中断合并配置。

```
# ethtool -c eth0
Coalesce parameters for eth0:
Adaptive RX: off  TX: off
……

rx-usecs: 1
rx-frames: 0
rx-usecs-irq: 0
rx-frames-irq: 0
……
```

我们来说一下上述结果的大致含义

- Adaptive RX: 自适应中断合并，网卡驱动自己判断啥时候该合并啥时候不合并
- rx-usecs：当过这么长时间过后，一个RX interrupt就会被产生
- rx-frames：当累计接收到这么多个帧后，一个RX interrupt就会被产生

如果你想好了修改其中的某一个参数了的话，直接使用 `ethtool -C` 就可以，例如：

```
ethtool -C eth0 adaptive-rx on
```

不过需要注意的是，减少中断数量虽然能使得Linux整体吞吐更高，不过一些包的延迟也会增大，所以用的时候得适当注意。

四 软中断监控与调优

在硬中断之后，再接下来的处理过程就是ksoftirqd内核线程中处理的软中断了。之前我们说过，软中断和它对应的硬中断是在同一个核心上处理的。因此，前面硬中断分散到多核上处理的时候，软中断的优化其实也就跟着做了，也会被多核处理。不过软中断也还有自己的可优化选项。

1）监控

软中断的信息可以从 /proc/softirqs 读取：

```
$ cat /proc/softirqs
                    CPU0       CPU1       CPU2       CPU3
          HI:          0          2          2          0
       TIMER:  704301348 1013086839  831487473 2202821058
      NET_TX:      33628      31329      32891     105243
      NET_RX:  418082154 2418421545  429443219 1504510793
       BLOCK:         37          0          0   25728280
BLOCK_IOPOLL:          0          0          0          0
     TASKLET:     271783     273780     276790     341003
       SCHED: 1544746947 1374552718 1287098690 2221303707
     HRTIMER:          0          0          0          0
         RCU: 3200539884 3336543147 3228730912 3584743459
```

2）软中断budget调整

不知道你有没有听说过番茄工作法，它的大致意思就是你要有一整段的不被打扰的时间，集中精力处理某一项作业。这一整段时间时长被建议是25分钟。对于我们的Linux的处理软中断的ksoftirqd来说，它也和番茄工作法思路类似。一旦它被硬中断触发开始了工作，它会集中精力处理一波儿网络包（绝不只是1个），然后再去做别的事情。

我们说的处理一波儿是多少呢，策略略复杂。我们只说其中一个比较容易理解的，那就是 `net.core.netdev_budget` 内核参数。

```
# sysctl -a | grep
net.core.netdev_budget = 300
```

这个的意思说的是，ksoftirqd一次最多处理300个包，处理够了就会把CPU主动让出来，以便Linux上其它的任务可以得到处理。那么假如说，我们现在就是想提高内核处理网络包的效率。那就可以让ksoftirqd

进程多干一会儿网络包的接收，再让出CPU。至于怎么提高，直接修改不这个参数的值就好了。

```
# sysctl -w net.core.netdev_budget=600
```

如果要保证重启仍然生效，需要将这个配置写到/etc/sysctl.conf

3）软中断GRO合并

GRO和硬中断合并的思想很类似，不过阶段不同。硬中断合并是在中断发起之前，而GRO已经到了软中断上下文中了。

如果应用中是大文件的传输，大部分包都是一段数据，不用GRO的话，会每次都将一个小包传送到协议栈（IP接收函数、TCP接收）函数中进行处理。开启GRO的话，Linux就会智能进行包的合并，之后将一个大包传给协议处理函数。这样CPU的效率也是就提高了。

```
# ethtool -k eth0 | grep generic-receive-offload
generic-receive-offload: on
```

如果你的网卡驱动没有打开GRO的话，可以通过如下方式打开。

```
# ethtool -K eth0 gro on
```

> GRO说的仅仅只是包的接收阶段的优化方式，对于发送来说是GSO。

五 总结

在网络技术这一领域里，有太多的知识内容都停留在理论阶段了。你可能觉得你的网络学的滚瓜烂熟了，可是当你的线上服务出现问题的时候，你还是不知道该怎么排查，怎么优化。这就是因为只懂了理论，而不清楚Linux是通过哪些内核机制将网络技术落地的，各个内核组件之间怎么配合，每个组件有哪些参数可以做调整。我们用两篇文章详细讨论了Linux网络包的接收过程，以及这个过程中的一些统计数据如何查看，如何调优。相信消化完这两篇文章之后，你的网络的理解直接能提升1个Level，你对线上服务的把控能力也会更加如鱼得水。

收录于合集 #开发内功修炼之网络篇 42

‹ 上一篇
图解Linux网络包接收过程

下一篇 ›
聊聊TCP连接耗时的那些事儿

喜欢此内容的人还喜欢

第二本书交稿了
开发内功修炼

结婚就是找个“好好睡觉”的人！
小北

苹果换充电接口了，但你的Type-C线还是用不上
澎湃美数课