Linux低延迟服务器系统调优

饶萌
精通Markdown，了解C++，高延迟交易，INTJ

最近做了一些系统和网络调优相关的测试，达到了期望的效果，有些感悟。同时，我也发现知乎上对Linux服务器低延迟技术的讨论比较欠缺（满嘴高并发现象）；或者对现今cpu + 网卡的低延迟潜力认识不足（动辄FPGA现象），比如一篇知乎高赞的介绍FPGA的文章写到“从延迟上讲，网卡把数据包收到 CPU，CPU 再发给网卡，即使使用 DPDK 这样高性能的数据包处理框架，延迟也有 4~5 微秒。更严重的问题是，通用 CPU 的延迟不够稳定。例如当负载较高时，转发延迟可能升到几十微秒甚至更高”，刚好我前几天做过类似的性能测试，发现一个tcp或udp的echo server可以把网卡到网卡的延迟稳定在1微秒以内，不会比FPGA方案慢很多吧？

因此，我觉得有必要分享下自己的见解。总的来说，我打算分两篇文章讨论相关低延迟技术：

1）系统调优（本文）：一些低延迟相关的Linux系统设置，和一些原则。

2）网络调优：使用solarflare网卡降低网络IO延迟。

这里不打算介绍用户空间的延迟优化，因为太广泛了，另外我之前的文章也分享一些解决某类问题的低延迟类库。

说到低延迟，关键点不在低，而在**稳定**，稳定即可预期，可掌控，其对于诸如高频交易领域来说尤为重要。 而说到Linux的低延迟技术，一个不能不提的词是"kernel bypass"，也就是绕过内核，为什么呢？因为内核处理不仅慢而且延迟不稳定。可以把操作系统想象成一个庞大的框架，它和其他软件框架并没有什么本质的不同，只不过更加底层更加复杂而已。既然是框架，就要考虑到通用性，需要满足各种对类型用户的需求，有时你只需要20%的功能，却只能take all。

因此我认为一个延迟要求很高（比如个位数微秒级延迟）的实时任务是不能触碰内核的，（当然在程序的启动初始化和停止阶段没有个要求，That's how linux works）。 这里的避免触碰是一个比bypass更高的要求：不能以**任何**方式进入内核，不能直接或间接的执行系统调用(trap)，不能出现page fault(exception)，不能被中断(interrupt)。trap和exception是主动进入内核的方式，可以在用户程序中避免，这里不深入讨论（比如在程序初始化阶段分配好所有需要的内存并keep的物理内存中；让其他非实时线程写日志文件等）。本文的关键点在于避免关键线程被中断，这是个比较难达到的要求，但是gain却不小，因为它是延迟稳定的关键点。即使中断发生时线程是空闲的，但重新回到用户态后cpu缓存被污染了，下一次处理请求的延迟也会变得不稳定。

不幸的是Linux并没有提供一个简单的选项让用户完全关闭中断，也不可能这么做（That's how linux works），我们只能想法设法避免让关键任务收到中断。我们知道，中断是cpu core收到的，我们可以让关键线程**绑定**在某个core上，然后避免各种中断源（IRQ)向这个core发送中断。绑定可以通过 `taskset` 或 `sched_setaffinity` 实现，这里不赘述。 避免IRQ向某个core发中断可以通过改写 `/proc/irq/*/smp_affinity` 来实现。例如整个系统有一块cpu共8个核，我们想对core 4~7屏蔽中断，只需把**非**屏蔽中断的core(0 ~ 3)的mask "f"写入smp_affinity文件。这个操作对硬件中断（比如硬盘和网卡）都是有效的，但对软中断无效（比如local timer interrupt和work queue），对于work queue的屏蔽可以通过改写 `/sys/devices/virtual/workqueue/*/cpumask` 来实现，本例中还是写入"f"。

那么剩下的主要就是local timer interrupt(LOC in /proc/interrupts)了。Linux的scheduler time slice是通过LOC实现的，如果我们让线程独占一个core，就不需要scheduler在这个core上切换线程了，这是可以做到的：通过 `isolcpus` 系统启动选项隔离一些核，让它们只能被绑定的线程使

用，同时，为了减少独占线程收到的LOC频率，我们还需要使用"adaptive-ticks"模式，这可以通过 `nohz_full` 和 `rcu_nocbs` 启动选项实现。本例中需要在系统启动选项加入 `isolcpus=4,5,6,7 nohz_full=4,5,6,7 rcu_nocbs=4,5,6,7` 来使得4~7核变成adaptive-ticks。adaptive-ticks的效果是：如果core上的running task只有一个时，系统向其发送LOC的频率会降低成每秒一次，内核文档解释了不能完全屏蔽LOC的原因："Some process-handling operations still require the occasional scheduling-clock tick. These operations include calculating CPU load, maintaining sched average, computing CFS entity vruntime, computing avenrun, and carrying out load balancing. They are currently accommodated by scheduling-clock tick every second or so. On-going work will eliminate the need even for these infrequent scheduling-clock ticks."。

至此，通过修改系统设置，我们能够把中断频率降低成每秒一次，这已经不错了。如果想做的更完美些，让关键线程长时间（比如几个小时）不收到任何中断，只能修改内核延长中断的发送周期。不同kernel版本相关代码有所差异，这里就不深入讨论。不过大家可能会顾虑：这样改变系统运行方式会不会导致什么问题呢？我的经验是，这有可能会影响某些功能的正常运转（如内核文档提到的那些），但我尚未发现程序和系统发生任何异常，说明这项内核修改至少不会影响我需要的功能，我会继续使用。

两个原则：

1）如果一件事情可以被delay一段时间，那它往往能够被delay的更久，因为它没那么重要。

2）不要为不使用的东西付费，对于性能优化来说尤为如此。

如何检测中断屏蔽的效果呢？可以watch `/proc/interrupts` 文件的变化 。更好的方法是用简单的测试程序来验证延迟的稳定性：

```
#include <iostream>

uint64_t now() {
   return __builtin_ia32_rdtsc();
}

int main() {
  uint64_t last = now();
  while (true) {
    uint64_t cur = now();
    uint64_t diff = cur - last;
    if (diff > 300) {
      std::cout << "latency: " << diff << " cycles" << std::endl;
      cur = now();
    }
    last = cur;
  }
  return 0;
}
```

通过taskset绑定一个核运行程序，每进入一次内核会打印一条信息。

最后，除了进入内核以外，影响延迟稳定性的因素还有**cache miss**和**tlb miss**。

对于减少cache miss，一方面需要优化程序，minimize memory footprint，或者说减少一个操作访问cache line的个数，一个缓存友好例子是一种能高速查找的自适应哈希表文章中的哈希表的实现方式。另一方面，可以通过分(lang)配(fei)硬件资源让关键线程占有更多的缓存，比如系统有两块CPU，每块8核，我们可以把第二块CPU的所有核都隔离掉，然后把关键线程绑定到其中的部分核上，可能系统只有一两个关键线程，但它们却能拥有整块CPU的L3 cache。

对于减少tlb miss，可以使用huge pages。

编辑于 2019-03-16 09:37

Linux　系统优化　Linux 内核

写下你的评论...

36 条评论　默认　最新

Nibnat
您好，请问如果做到了屏蔽中断并且测试程序独占核运行，输出应该是 300 并且稳定输出一样的数字吗?
07-09　回复　赞

杰登菲驰
大佬,我把上面说的都做了. 但是跑上面的程序,还是经常有中断,怎么回事啊? 好奇怪!
2021-11-01　回复　1

lotuscrown
验证延迟的稳定性的小程序写的貌似有问题
2021-03-16　回复　赞

whosyourdaddy
如果某个核把所有的中断（包括时钟中断都屏蔽了），这个核上执行一个没有syscall的死循环，那感觉这个核就永远被卡死在那个循环上了?
2021-01-26　回复　赞

量化菜鸟
请问一下，怎么把时钟中断改成一天一次，大概在哪几个函数里面改?
2020-02-21　回复　赞

饶萌 作者
有些kernel线程隔离不了，但不影响性能
2019-08-28　回复　赞

zllz ▸ **饶萌**
好 谢谢你
2019-08-28　回复　赞

饶萌 作者 ▸ **zllz**
没试过，估计改变不大
2019-08-28　回复　赞

zllz
升级内核是否有优化 现在用的是3.10 升级到3.16呢
2019-08-28　回复　赞

zllz
您好 watchdog ksoftirqd进程该怎么隔离呢 或者绑核

2019-08-28 回复 赞

zllz
您好 请教问题，设置isolcpus nohz_full rcu_nocbs 对kworker这些进程没有作用，有解决办法吗
2019-08-22 回复 赞

饶萌 作者 ▸ **zllz**
修改生效，因为每次重启都要修改
2019-08-22 回复 赞

zllz ▸ **饶萌**
修改就生效还是需要重启机器的
2019-08-22 回复 赞

饶萌 作者
文章前面提到了怎么屏蔽work queue
2019-08-22 回复 赞

饶萌 作者
屏蔽了work queue后，虽然kwoker线程还在，但不会影响core
2019-08-22 回复 赞

行动派
你好，如何绑定或者说diable掉Function call interrupts中断，每次插入U盘，Function call interrupts中断都会增加，进而影响系统实时性
2019-07-29 回复 赞

Vivian Zhou
TCP/UDP 确实。虽然不懂技术，但做低延时招聘，确实有要求这个玩的6的。FPGA不是每家都上，毕竟板子贵，要常换。
2019-06-15 回复 赞

zaxon
楼主你比我狠，哈哈。cpu够用可以这么玩。不够用这些大招得慎重
2019-04-05 回复 赞

Ever
IB其实做的很不错额利用RDMA
2019-03-26 回复 赞

牛博恩
安装tuned，选择network-latency profile。完了

2019-03-13 回复 7

云中君
并不能，在很多业务场景下network-latency实测不是最优，要以实测为准。
2022-08-01 回复 1

嗨siri请擅长知乎
不错 看过seastar框架也是这样干的，但就是不知道为啥，这框架跑起来的请求处理延迟不稳定。
2019-03-11 回复 1

造轮子
有个状态图更佳
2019-03-10 回复 1

卡特白
是让延时低于1us还是延时的方差低于1us呀?
2019-03-10 回复 1

卡特白 ▸ **饶萌**
赞！用的是onload? 期待下一篇更新。
2019-03-10 回复 1

饶萌 作者
延时本身，需要专业网卡支持
2019-03-10 回复 1

老狐狸9527
另外，没明白就只看个转发么？处理流程往往是大头，再往高的应用现在有ib，rdma这些成熟实习。
2019-03-09 回复 赞

饶萌 作者
处理流程属于用户程序优化范畴，本文主要讨论在系统层面稳定延迟。
2019-03-09 回复 3

老狐狸9527
与其这样还不如用fpga
2019-03-09 回复 1

老狐狸9527 ▸ **饶萌**
成本是一方面，fpga能更好的达到这种控制效果，另外intel有个系列是fpga加cpu，fpga的固件可以非常方便的改变。x86最通用，但这些最适用的场合交给soc，fpga更合适，软件比固化的硬件还是慢的。
2019-03-09 回复 1

饶萌 作者
你指的是成本的角度吧。如果从易用性和灵活性上看还是CPU方案更人性些，毕竟系统设置只用做一次就可以了，而且并不复杂。
2019-03-09 回复 1

饶萌 作者 ▸ **老狐狸9527**
嗯，具体要看应用场景，CPU和FPGA各有利弊。
2019-03-09 回复 赞

筷子 ▸ **老狐狸9527**
这个还只是概念
2019-06-03 回复 赞

小毛毛毛蛋 ▸ **饶萌**
altera
2020-04-21 回复 赞

Starve Jokes
所以不能langfei硬件资源怎么办。busywait的时候spinlock里面穿插访问点别的东西么。。。
2019-03-09 回复 1

饶萌 作者

硬件资源有限的话就不用想独占缓存了，所有核都用满就行了。

2019-03-09

回复 赞

文章被以下专栏收录

蓝色的味道

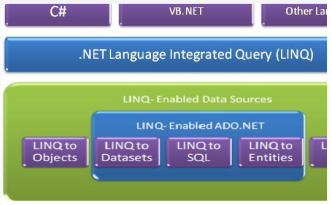
推荐阅读

Linux低延迟服务器系统调优

Skyke... 发表于Skyke...

分享6个网络延迟测试工具，都是老网工的必备好物

网络工程师... 发表于网络工程师...



就是这么坑：Linq的延迟加特性

叶影 发表于C#