Performance Tuning Linux Instances on EC2

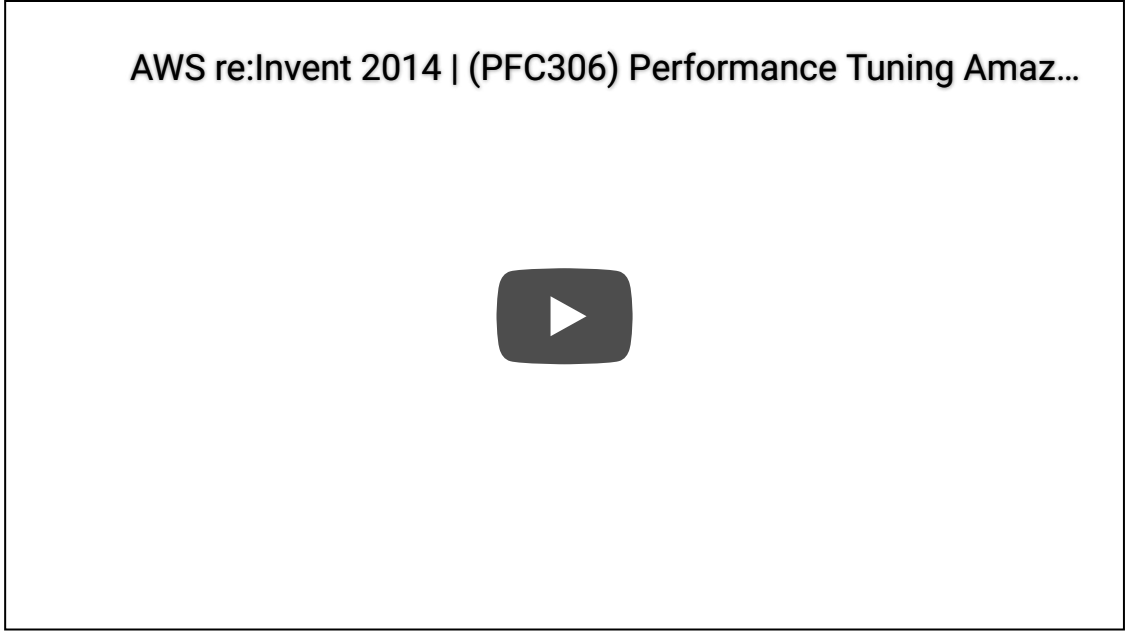
03 Mar 2015

At the last AWS re:Invent, I gave a talk on "Performance Tuning EC2 Instances", where I showed how my team (Performance and Reliability Engineering) tunes Linux EC2 instances at Netflix. This includes instance selection, EC2 features, Linux kernel tuning, and the use of observability.

This is the most comprehensive tuning talk I've given, and summarizes the different ways we tune at the instance level. It should be useful for anyone running Linux in the cloud, not just in EC2.

The slides are on slideshare:

It was also videoed, which is on youtube:



I often share my work on performance observability, but not tuning. Observability is where the bigger wins are, as you can discover and then eliminate unnecessary work. It can also help show that tuning is required. But I've also been meaning to share same examples of tuning, and had my chance at AWS re:Invent.

In the 3rd section of the talk, I included the tunables we are using on Ubuntu Trusty, to show examples of what is possible. I've included them below for easy browsing. Please watch the video for context.

WARNING: These tunables were developed in late 2014, for Ubuntu Trusty instances on EC2.

CPU

```
schedtool –B PID
```

Virtual Memory

```
vm.swappiness = 0       # from 60
```

Huge Pages

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled  # from madvise
```

File System

```
vm.dirty_ratio = 80                     # from 40
vm.dirty_background_ratio = 5           # from 10
vm.dirty_expire_centisecs = 12000       # from 3000
mount -o defaults,noatime,discard,nobarrier …
```

Storage I/O

```
/sys/block/*/queue/rq_affinity  2
/sys/block/*/queue/scheduler        noop
/sys/block/*/queue/nr_requests  256
/sys/block/*/queue/read_ahead_kb    256
mdadm –chunk=64 ...
```

Networking

```
net.core.somaxconn = 1000
net.core.netdev_max_backlog = 5000
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.ipv4.tcp_wmem = 4096 12582912 16777216
net.ipv4.tcp_rmem = 4096 12582912 16777216
net.ipv4.tcp_max_syn_backlog = 8096
net.ipv4.tcp_slow_start_after_idle = 0
net.ipv4.tcp_tw_reuse = 1
net.ipv4.ip_local_port_range = 10240 65535
net.ipv4.tcp_abort_on_overflow = 1    # maybe
```

Hypervisor (Xen)

```
echo tsc > /sys/devices/system/clocksource/clocksource0/current_clocksource
```

Setting the clocksource came from a performance regression we found when moving to Ubuntu Trusty, which can be fixed by switching clocksource to TSC. Best case example (so far): CPU usage reduced by 30%, and average app latency reduced by 43%. Beware of clock drift, as in the (distant) past TSC has been unreliable.

In the talk I described these tunables as our medicine cabinet, and to "consider these best before 2015". Tuning is a process, not a product. Copy-n-pasting these tunables is a little like taking someone else's medication; doing so years later is like taking someone else's *expired* medication.

As an update: slide 62 shows "Broken Java stacks" in a flame graph, which we now have a workaround for (an OpenJDK patch I wrote). See my Linux Profiling at Netflix post, where I have an example flame graph with working Java stacks.

AWS re:Invent was a massive event, and there were many talks I missed. Fortunately they were recorded, and Adrian Cockcroft published a list of interesting talks which are worth checking out.

You can comment here, but I can't guarantee your comment will remain here forever: I might switch comment systems at some point (eg, if disqus add advertisements).