Linux tuning

Various knobs that might be relevant for server/HPC workloads

Niklas Edmundsson, HPC2N

Disclaimer(s)

- Everything in this presentation has been tested/used on Ubuntu 10.04 ... 20.04 LTS, RHEL/CentOS 6, 7 (unless noted otherwise)
- Most likely also applies to newer Linux distros/kernels, but check defaults before you start turning knobs.
- There's likely lot of stuff missing, I've deliberately left out things that are mentioned in specific application READMEs etc.

sysctl: kernel.pid_max

- Default: 32768
- PID randomization causes this to be quickly exhausted, PID wraps
  - Especially true on modern multi-core hardware running lots of scripts/processes.
- Simple fix: Increase it to 999999
  - One more digit shown in ps/top/etc
  - Approx 30 times more “pid number space”, so wraps less often
  - HPC2N has been using this for quite some time (5+ years) on everything Linux, no issue discovered in any application (standard workload, HPC, user stuff, etc)
- Can be increased further, but we haven't had the need (yet).

sysctl: kernel.core_pattern

- Default: core
- Annoyed by core files being overwritten if an application crashes multiple times?
- Change to, for example, core.%e.%h.%p
    - core.EXECUTABLE.HOSTNAME.PID

Process scheduling

- Default tuning is for laptop/desktop, ie focus on latency and interactive use
- Not quite optimal for server/HPC workloads since context switches are expensive
- RHEL tuned: https://tuned-project.org/
  - Or equivalent RHEL/CentOS-docco
- The throughput-performance profile suggests:
  - kernel.sched_min_granularity_ns = 10000000
  - kernel.sched_wakeup_granularity_ns = 15000000
- Rolled out everywhere on HPC2N
  - Also on cluster when upgraded to Ubuntu Focal, but no performance followup has been made (lost in the pandemic...)

sysctl:s for ARP cache garbage collection

- On large IP segments you often get many ARP-entries
- If garbage collection happens too early/often machines communicating are forced to do ARP requests instead of useful work.
- Defaults tend to be geared towards small network segments
- HPC2N tuning increases defaults by a factor of 4-8
- Originally an adaption for Akka (672 nodes)
- Applied on all hosts
- For static nets/configs you can also increase the time interval between garbage collection iterations

sysctl:s for ARP cache garbage collection (2)

```
# IPv4
# Minimum number of entries (default 128)
net.ipv4.neigh.default.gc_thresh1 = 1024
# Soft maximum (default 512)
net.ipv4.neigh.default.gc_thresh2 = 2048
# Hard maximum (default 1024)
net.ipv4.neigh.default.gc_thresh3 = 4096
# Interval (default 30 seconds)
net.ipv4.neigh.default.gc_interval = 120
# IPv6
net.ipv6.neigh.default.gc_thresh1 = 1024
net.ipv6.neigh.default.gc_thresh2 = 2048
net.ipv6.neigh.default.gc_thresh3 = 4096
net.ipv6.neigh.default.gc_interval = 120
```

sysctl: vm.min_free_kbytes

- The universal solve-it-all knob to turn whenever you see dmesg/syslog "alloc failed" messages from the kernel
- Shows up in various best practices all over the place
- HPC2N
  - Our typical tuning is to increase to a value that corresponds to approx 0.5 seconds of maximum network bandwidth
  - If you still get "alloc fail" messages, try doubling it again

```
# Reserve more free memory to be able to handle network traffic bursts better.
# Rough rule: size to be able to handle 0.5s burst.
vm.min_free_kbytes = 524288
```

Writing (kernel)-buffers to disk

- A lot of tuning advice you find is about increasing write buffers
  - The tuned throughput-performance profile for example
- Big write buffers has its merits
  - Big files that are written, modified and then copied elsewhere and you want to avoid IO for the temporary files.
  - Intermittent writes where background flush catches up in the breaks.
  - Etc
- However, big write buffers makes the situation worse if you have continuous flows, big files and not enough IO capacity
  - “write storms”
- Many (file) server workloads perform better if you instead decrease the buffers

Writing (kernel)-buffers to disk (2)

- vm.dirty_background_ratio alt. vm.dirty_background_bytes
  - Defaults to background writes starting at 10% RAM used
  - Troublesome for machines where IO performance is close to network bandwidth, when writes start the machine never catches up to empty the buffers.
- vm.dirty_ratio alt. vm.dirty_bytes
  - Defaults to 20% of RAM
  - Troublesome when hit, all writing processes freezes until everything is written.
    - Even more painful if application has a timeout for writes
- vm.dirty_expire_centisecs and vm.dirty_writeback_centisecs
  - Time based trigger for the above

Writing (kernel)-buffers to disk (3)

- Worst case scenario: dCache tape pools
  - Transfers data between tape (TSM) and other dCache pools.
  - Buffered on local disk.
  - Tape transfers have a continuous speed
  - Default buffer settings usually causes data to be written in bursts, affecting other activity (ie transfers to/from tape).
    - Yes, also happens with flash-based systems.
  - Typical tuning to improve the situation is to lower buffers
    - Lower background limits so writes starts earlier
    - You want more margin between dirty_background and the hard dirty limit
    - Take care not to lower it too much, can cause file fragmentation

TCP window size tuning

- Linux default is max 4 MB autotuned TCP-windows
- Not enough for 10/25G hosts doing non-local traffic
- Not enough for (most) 100G hosts doing local traffic
- Cause: Bandwidth-delay product
  - Amount of data en route, ie speed * RTT
- Common bad workaround: "I must use multiple transfers/threads"
- Expect needing 64MB or more within Europe

TCP window size tuning (2)

- Tuned with sysctl, applies for new connections
- Don't touch net.core.[rw]mem_max !
  - Used only by application that "knows best" and uses setsockopt() to fiddle with the buffer size, that disables the autotuning with bad results.
- Modify the `net.ipv4.tcp_rmem` and `net.ipv4.tcp_wmem` sysctl:s
  - Change only the rightmost value.
  - Note that the value set for the read buffer typically needs to be 50% larger than the wanted TCP window size due to overhead.

```
# Typical defaults, listed by sysctl net.ipv4.tcp_rmem net.ipv4.tcp_wmem:
#   net.ipv4.tcp_rmem = 4096        87380   6291456
#   net.ipv4.tcp_wmem = 4096        16384   4194304
#
# Tuning for 64MiB tcp windows, with similar tcp_rmem overhead as default:
net.ipv4.tcp_rmem = 4096        87380   100663296
net.ipv4.tcp_wmem = 4096        16384   67108864
```

BBR congestion control

- Default Linux congestion control has problem dealing with links having spurious high package loss.
  - Typical (over)reaction: slow ramp up from full stop
  - Other symptoms: Low transfer speeds without apparent reason
- BBR is able to detect spurious loss and reacts in a more appropriate manner.
- BBR is available in Linux distributions with a sufficiently recent kernel, for example Ubuntu 18.04, RHEL 8, Debian 10, and more.
- Highly recommended to enable everywhere
  - On HPC2N, enabled on everything from Ubuntu 18.04 and up