Optimizing servers - Tuning the GNU/Linux Kernel

Linux is the world-leading open-source kernel.

It is designed to peform well on a wide range of hardware.

File Handle Limits

When you're serving a lot of traffic it is usually the case that the traffic you're serving is coming from a large number of local files.

The kernel has built-in limits on the number of files that a process can open, and raising these limits, at a cost of some system memory, is usually a sane thing to attempt.

You can view the current limit on the number of open-files by running:

```
$ cat /proc/sys/fs/file-max
```

The limit can be raised interactively by running, as `root`:

```
# sysctl -w fs.file-max=100000
```

If you wish that change to be made persistently you should append to the file `/etc/sysctl.conf` the line:

```
fs.file-max = 100000
```

Then run the following command to make your change take effect:

```
# sysctl -p
```

Socket Tuning

For servers which are handling large numbers of concurent sessions, there are some TCP options that should probabaly be tweaked.

With a large number of clients comnunicating with your server it wouldn't be unusual to have a 20,000 open sockets or more. To increase that range you append the following to the bottom of `/etc/sysctl.conf`:

```
# Use the full range of ports.
net.ipv4.ip_local_port_range = 1024 65535
```

You can also increase the recycling time of sockets, avoiding large numbers of them staying in the `TIME_WAIT` status by adding these values to `/etc/sysctl.conf`:

```
# Enables fast recycling of TIME_WAIT sockets.
# (Use with caution according to the kernel documentation!)
net.ipv4.tcp_tw_recycle = 1
# Allow reuse of sockets in TIME_WAIT state for new connections
# only when it is safe from the network stack’s perspective.
net.ipv4.tcp_tw_reuse = 1
```

Finally one problem you'll find is that if a socket is listening and busy a connection-backlog will pile up. The kernel will keep pending connections in a buffer before failing. You can tweak several values to increase the size of the backlog:

```
#
# 16MB per socket - which sounds like a lot, but will virtually never
# consume that much.
#
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
# Increase the number of outstanding syn requests allowed.
# c.f. The use of syncookies.
net.ipv4.tcp_max_syn_backlog = 4096
net.ipv4.tcp_syncookies = 1
# The maximum number of "backlogged sockets".  Default is 128.
net.core.somaxconn = 1024
```

The trade-off here is that a connecting client will see a slow connection, but this is almost certainly better than a `Connection Refused` error.

Once you've made those additions you can cause them to be loaded by running:

```
# sysctl -p
```

Finally if you've changed these limits you will need to restart the associated daemons. (For example "`service nginx restart`".)

Process Scheduler

If you're running a recent (newer than approx 2.6.32) you've got the 'Completely Fair Scheduler' (CFS) For modern systems serving lots of connections on lots of cores, you may hit issues with process migration.

There's a kernel parameter that determines how long a migrated process has to be running before the kernel will consider migrating it again to another core. The sysctl name is `sched_migration_cost_ns`, default value 50000 (that's ns so 0.5 ms):

```
$ cat /proc/sys/kernel/sched_migration_cost_ns
```

(It was renamed from sched_migration_cost at some point between 3.5 and 3.8)

Forking servers, like PostgreSQL or Apache, scale to much higher levels of concurrent connections if this is made larger, by at least an order of magnitude:

The limit can be raised interactively by running, as `root`:

```
# sysctl -w kernel.sched_migration_cost_ns=5000000
```

If you wish that change to be made persistently you should append to the file `/etc/sysctl.conf` the line:

```
kernel.sched_migration_cost_ns = 5000000
```

Another parameter that can dramatically impact forking servers is `sched_autogroup_enabled`. This setting groups tasks by TTY, to improve perceived responsiveness on an interactive system. On a server with a long running forking daemon, this will tend to keep child processes from migrating away as soon as they should. It can be disabled like so:

```
# sysctl -w kernel.sched_autogroup_enabled=0
```

Various PostgreSQL users have reported (on the postgresql performance mailing list) gains up to 30% on highly concurrent workloads on multi-core systems.

If you wish that change to be made persistently you should append to the file `/etc/sysctl.conf` the line:

```
kernel.sched_autogroup_enabled = 0
```

Then run the following command to make your change take effect:

```
# sysctl -p
```

Filesystem Tuning

You almost certainly want to disable the "`atime`" option on your filesystems.

With this disabled that the last time a file was accessed won't be constantly updated every time you read a file, since this information isn't generally useful inand causes extra disk hits, its typically disabled.

To do this, just edit `/etc/fstab` and add "notime" as a mount option for the filesystem. For example:

```
    /dev/rd/c0d0p3          /test                   ext3    noatime
```

Swap Tuning

- **TODO**

RAID Tuning

It seems to be the case that if you have the `deadline` scheduler this is best for RAID setups, however this is something that you'll want to test yourself.

Boot your kernel with `elevator=deadline` appended to the command-line and compare the result via your favourite filesystem test.