Sysctl tweaks

Quelle: https://wiki.mikejung.biz/Sysctl_tweaks

Other Tuning Pages

- OS Tuning - Information on Transparent Huge Pages, NUMA and how to optimize applications.
- Performance Analysis - How to track down I/O issues, and how to determine where server load is coming from.

How to modify sysctl settings using sysctl.conf

You can add / modify sysctl settings by editing the sysctl.conf file. If you choose to edit the conf file directly you must run sysctl -p to apply the changes after the edit has been made.

```
# vim /etc/sysctl.conf
```

To ensure changes to sysctl configuration persist across reboots, run sysctl -p.

```
# sysctl -p
```

How to modify sysctl settings using command line

You can also modify sysctl settings by using sysctl command line. By using the “-w” option you can write various changes to the main conf file. You will want to run sysctl -p after you modify the setting to make sure the changes persist after a reboot.

```
# sysctl -w $thing_to_change
```

To ensure changes persist across reboots

```
# sysctl -p
```

How to view current sysctl settings

To view the current sysctl settings you can run the command below which will list all of the settings currently applied to the server.

```
# sysctl -a
```

If you want to make a backup of the sysctl settings before you begin tweaking and changing stuff, run this command, which will output all current sysctl settings, then send the output to a file called sysctl.bak. I always like to do this right away so that I can always revert settings back to the default if

needed.

```
# sysctl -a > /$location_to_save/sysctl.bak
```

Netflix 2014 EC2 sysctl tweaks

http://www.slideshare.net/AmazonWebServices/pfc306-performance-tuning-amazon-ec2-instances-aws-reinvent-2014

Below is a list of tweaks that Brendan Gregg mentioned during his talk at AWS re:invent. Please do not just paste these into sysctl.conf without backing up your existing configuration. Also, please read into each of these settings and make sure you understand what you are changing before you do anything. I've found that a vm.swappiness of 10 or 20 works fairly well for most VPS servers. I do not suggest settings this value to 0 or else you might start to run into issues with processes constantly getting killed off by Linux. If you have already configured services like Apache, PHP-FPM, MySQL and Memcached to use reasonable amounts of RAM and have configured services to use up to 100% RAM then setting vm.swappiness to 0 might be ok. However, if you've configured services to use way more memory than you have, then you may want to allow Linux to swap some files out before you hit the memory limit.

```
# Discourage Linux from swapping out idle processes to disk (default is 60)
vm.swappiness = 20
vm.dirty_ratio = 40
vm.dirty_background_ratio = 10

# Disable Transparent Huge Pages
echo never > /sys/kernel/mm/transparent_hugepage/enabled

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.core.somaxconn

source: http://linux.die.net/man/2/listen

SOMAXCONN can be used as a statement which specifies the max number of connection requests that can be queued for any given listening socket.

If the backlog argument is greater than the value in /proc/sys/net/core/somaxconn, then it is silently

truncated to that value; the default value in this file is 128. In kernels before 2.4.25, this limit was a hard coded value, SOMAXCONN, with the value 128.

source: https://computing.llnl.gov/linux/slurm/high_throughput.html

/proc/sys/net/core/somaxconn: Limit of socket listen() backlog, known in userspace as SOMAXCONN. Defaults to 128. The value should be raised substantially to support bursts of request. For example, to support a burst of 1024 requests, set somaxconn to 1024.

To modify this value in sysctl.conf use the following format

```
net.core.somaxconn = 1000
```

net.core.netdev_max_backlog

net.core.netdev_max_backlog determines the maximum number of packets, queued on the INPUT side, when the interface receives packets faster than kernel can process them. The default value for Ubuntu 15.04 (early beta) is 1000. Many Linux Kernel tuning guides suggest raising this value to over 100,000. If your kernel / server can keep up with the traffic being sent to it then raising this value probably won't help with performance. If your server is getting slammed with requests and cannot keep up with incoming packets then raising this value could help reduce the amount of dropped packets that aren't able to fit in the queue.

```
net.core.netdev_max_backlog = 1000 (default)
```

If you are going to raise net.core.netdev_max_backlog, I would suggest starting with low values and raising this only if needed. Just because Oracle or some Science / HPC guide uses net.core.netdev_max_backlog = 300000 doesn't mean your WordPress blog needs a backlog that's this large. Most of the blogs / articles that mention this setting are using 10Gb or 40Gb NICs, and are running extremely latency sensitive applications, so if you aren't using high end network cards, or doing Science, then you probably won't have to change this setting.

You can view the Kernel Docs here for more information about this setting - https://www.kernel.org/doc/Documentation/sysctl/net.txt

net.core.rmem_max

The net.core.rmem_max setting defines the maximum receive socket buffer size in bytes.

There are a few different settings that all appear to be very similar. You can see that on Ubuntu 15.04 (3.18.0-13-generic) the default value for net.core.rmem_max is 212992. The default and max values are the same in this case. Raising this to a larger value will increase the buffer size, but this can have nasty effects in terms of “buffer bloat” I highly suggest reading about the current state of Linux networking by checking out this article - http://lwn.net/Articles/616241/

```
# sysctl -a | grep -i rmem

net.core.rmem_default = 212992
```

```
net.core.rmem_max = 212992
net.ipv4.tcp_rmem = 4096    87380    6291456
net.ipv4.udp_rmem_min = 4096
```

net.core.wmem_max

The net.core.wmem_max setting defines the maximum send socket buffer size in bytes.

You can see that on Ubuntu 15.04 (3.18.0-13-generic) the default value for net.core.wmem_max is 212992, which is the same size as rmem_max. Raising this to a larger value will increase the send buffer size, but before you adjust this setting I highly suggest reading about the current state of Linux networking by checking out this article - http://lwn.net/Articles/616241/

```
# sysctl -a | grep -i wmem

net.core.wmem_default = 212992
net.core.wmem_max = 212992
net.ipv4.tcp_wmem = 4096    16384    4194304
net.ipv4.udp_wmem_min = 4096
```

net.ipv4.tcp_wmem

tcp_wmem (since Linux 2.4) This is a vector of 3 integers: [min, default, max].

These parameters are used by TCP to regulate send buffer sizes. TCP dynamically adjusts the size of the send buffer from the default values listed below, in the range of these values, depending on memory available.

- **min** Minimum size of the send buffer used by each TCP socket. The default value is the system page size. (On Linux 2.4, the default value is 4K bytes.)

This value is used to ensure that in memory pressure mode, allocations below this size will still succeed. This is not used to bound the size of the send buffer declared using SO_SNDBUF on a socket.

- **default** The default size of the send buffer for a TCP socket. This value overwrites the initial default buffer size from the generic global net.core.wmem_default defined for all protocols.

The default value is 16K bytes.If larger send buffer sizes are desired, this value should be increased (to affect all sockets).

To employ large TCP windows, the /proc/sys/net/ipv4/tcp_window_scaling must be set to a non-zero value (default).

- **max** The maximum size of the send buffer used by each TCP socket. This value does not override the value in /proc/sys/net/core/wmem_max. This is not used to limit the size of the send buffer declared using SO_SNDBUF on a socket.

The default value is calculated using the formula: max(65536, min(4MB, tcp_mem[1]*PAGE_SIZE/128))

On Linux 2.4, the default value is 128K bytes, lowered 64K depending on low-memory systems.)

To modify tcp_wmem min, default, and max values, edit sysctl.conf and use the following format

```
net.ipv4.tcp_wmem = 4096 12582912 16777216
```

net.ipv4.tcp_rmem

source: man tcp

tcp_rmem (since Linux 2.4) This is a vector of 3 integers: [min, default, max].

These parameters are used by TCP to regulate receive buffer sizes. TCP dynamically adjusts the size of the receive buffer from the defaults listed below, in the range of these values, depending on memory available in the system.

- **min** Minimum size of the receive buffer used by each TCP socket. The default value is the system page size. On Linux 2.4, the default value is 4K, lowered to PAGE_SIZE bytes in low-memory systems.

This value is used to ensure that in memory pressure mode, allocations below this size will still succeed. This is not used to bound the size of the receive buffer declared using SO_RCVBUF on a socket.

- **default** The default size of the receive buffer for a TCP socket. The default value is 87380 bytes. (On Linux 2.4, this will be lowered to 43689 in low-memory systems.)

This value overwrites the initial default buffer size from the generic global net.core.rmem_default defined for all protocols.

If larger receive buffer sizes are desired, this value should be increased (to affect all sockets). To employ large TCP windows, the net.ipv4.tcp_window_scaling must be enabled (default).

- **max** The maximum size of the receive buffer used by each TCP socket. The default value is calculated using the formula: max(87380, min(4MB, tcp_mem[1]*PAGE_SIZE/128)). (On Linux 2.4, the default is 87380*2 bytes, lowered to 87380 in low-memory systems).

This value does not override the global net.core.rmem_max. This is not used to limit the size of the receive buffer declared using SO_RCVBUF on a socket.

To modify tcp_rmem min, default, and max values, edit sysctl.conf and use the following format

```
net.ipv4.tcp_rmem = 4096 12582912 16777216
```

net.ipv4.tcp_max_syn_backlog

source: man tcp

- **tcp_max_syn_backlog** (integer; default: see below; since Linux 2.2)

- The maximum number of queued connection requests which have still not received an acknowledgement from the connecting client. If this number is exceeded, the kernel will begin dropping requests.
- The default value of 256 is increased to 1024 when the memory present in the system is adequate or greater (>= 128Mb), and reduced to 128 for those systems with very low memory (⇐ 32Mb).
- It is recommended that if this needs to be increased above 1024, TCP_SYNQ_HSIZEin include/net/tcp.h be modified to keep TCP_SYNQ_HSIZE*16⇐tcp_max_syn_backlog,and the kernel be recompiled.

To modify this value you can add the following line to /etc/sysctl.conf

```
net.ipv4.tcp_max_syn_backlog = $integer
```

You probably don't want this to be set too high unless your server has plenty of resources, even then, if you are getting a large backlog and your server can't keep up you might be better off with using a load balancer and splitting the traffic between multiple servers.

net.ipv4.tcp_slow_start_after_idle

tcp_slow_start_after_idle (Boolean; default: enabled; since Linux 2.6.18)

If enabled, provide RFC 2861 behavior and time out the congestion window after an idle period. An idle period is defined as the current RTO (retransmission timeout). If disabled, the congestion window will not be timed out after an idle period.

Many people suggest disabling this, which should help to improve performance in some cases. To disable tcp_slow_start_after_idle edit sysctl.conf

```
net.ipv4.tcp_slow_start_after_idle = 0
```

net.ipv4.tcp_tw_reuse

tcp_tw_reuse (Boolean; default: disabled; since Linux 2.4.19/2.6)

Allow to reuse TIME_WAIT sockets for new connections when it is safe from protocol viewpoint. It should not be changed without advice/request of technical experts.

If you do wish to enable this option you can do so by modifying sysctl.conf

```
net.ipv4.tcp_tw_reuse = 1
```

net.core.default_qdisc

Ubuntu 16.04 (3.18.0-13-generic) uses pfifo_fast as the default option.

```
# sysctl -a | grep -i net.core.default_qdisc

net.core.default_qdisc = pfifo_fast
```

According to this excellent article about linux buffer bloat the author suggests using the “fq_codel” algorithm.

```
# sysctl -w net.core.default_qdisc=fq_codel
# sysctl -p
```

“Kathie Nichols and Van Jacobson created the “Controlled Delay” or CoDel algorithm. CoDel looks somewhat like RED, in that it starts to drop packets when buffers get too full. But CoDel works by looking at the packets at the head of the queue — those which have been through the entire queue and are about to be transmitted.

If they have been in the queue for too long, they are simply dropped. The general idea is to try to maintain a maximum delay in the queue of 5ms (by default). CoDel has a number of good properties, Stephen said; it drops packets quickly (allowing TCP congestion control algorithms to do their thing) and maintains reasonable latencies. The “fq_codel” algorithm adds an SFQ dispatching mechanism in front of the CoDel queue, maintaining fairness between network connections.”

- Further reading about fq_codel - http://www.bufferbloat.net/projects/codel/wiki/Wiki/

Visitor Submitted NOTE: As of 2019, if implementing BBR at a webserver then fq may be preferable to fq_codel – see this comment from Dave of BufferBloat at CloudFlare blog re. BBR http://disq.us/p/1wlho65

(and for some more in-depth: https://github.com/systemd/systemd/issues/9725#issuecomment-413369212)

Last update: **2019/05/01 15:13**