# Operating System Tuning

## Kernel Tuning

The following Linux kernel tuning suggestions are provided:

# must be higher than (‘disk i/o structures’+ ‘max asyncio per engine’)\* ‘max online engines’

# or best just to set to max conceivable (1MB)

**fs.aio-max-nr = 1048576**

#each device plus each concurrent user will need a file handle – plus those used by the

#software…plus factor in file system devices…default is 3MB

#or best to just set to 2x higher than default

**fs.file-max = 6291456**

#standard shared memory tuning…

#first disable shared memory security blocks

**kernel.exec-shield = 0**

**kernel.randomize\_va\_space = 0**

# shmall is the total amount of shared memory allowed (in 4k pages) – period

# attempts to allocate shared memory above this will fail. Multiply by 4096

# to see bytes

**kernel.shmall = 4294967296**

# shmmax is the largest shared mem segment size (in bytes) (64GB) ASE, Backup

# server and other Sybase processes can use multiple shared memory segments –

# each segment can be up to shmmax in size (but may be smaller due to free

# contiguous memory space). Unless a small box, set in even GB.

**kernel.shmmax = 68719476736**

#maximum number of shared memory segments system wide

**kernel.shmmni = 4096**

# ASE 15.0.3 and higher can use huge pages – do NOT use for ASE’s earlier than

# 15.0.3 as this will block the memory from being available to ASE. Each huge

# page equivalates to 2MB of memory – but ASE will round shmmax up to nearest 256MB

# as a result (so make sure shmall is higher than result…if shmmax is in even GB,

# this will not be a problem).

**vm.nr\_hugepages = 2048**

#Range of local ports for outgoing connections. Quite small by default: 1024 4999

**net.ipv4.ip\_local\_port\_range = 16384 65536**

# tune network tcp read/write memory buffers for send/recv packets

# descriptions from ip-sysctl.txt (Linux source)

# This sets the default OS receive buffer size for all types of connections/protocols

# such as both UDP and TCP and non-TCP protocols.

**net.core.rmem\_default = 262144**

# This sets the max OS receive buffer size for all types of connections/protocols

# such as both UDP and TCP and non-TCP protocols.

**net.core.rmem\_max = 16777216**

# This sets the default OS send buffer size for all types of connections/protocols

# such as both UDP and TCP and non-TCP protocols.

**net.core.wmem\_default = 262144**

# This sets the max OS send buffer size for all types of connections/protocols

# such as both UDP and TCP and non-TCP protocols.

**net.core.wmem\_max = 16777216**

#tcp\_mem - vector of 3 INTEGERs: min, pressure, max

# low: below this number of pages TCP is not bothered about its

# memory appetite.

#

# pressure: when amount of memory allocated by TCP exceeds this number

# of pages, TCP moderates its memory consumption and enters memory

# pressure mode, which is exited when memory consumption falls

# under "low".

#

# high: number of pages allowed for queueing by all TCP sockets.

#

# Defaults are calculated at boot time from amount of available

# memory.

#

# Notes: some writings on web say the defaults are fine, but I have seen

# on 32GB systems where the values are (in 4K pages)

#

# net.ipv4.tcp\_mem = 196608 262144 393216

#

# …the “high” value being pretty high (>1GB) when viewed in terms of 4K pages.

# On larger enterprise systems with >64GB of memory, you might want to

# constrain this to 2GB of memory. Remember, this is in 4K pages – whereas

# the next ones (tcp\_wmem and tcp\_rmem) are in bytes

**net.ipv4.tcp\_mem = 262144 262144 524288**

#tcp\_wmem - vector of 3 INTEGERs: min, default, max

# min: Amount of memory reserved for send buffers for TCP socket.

# Each TCP socket has rights to use it due to fact of its birth.

# Default: 4K

#

# default: Amount of memory allowed for send buffers for TCP socket

# by default. This value overrides net.core.wmem\_default used

# by other protocols, it is usually lower than net.core.wmem\_default.

# Default: 16K

#

# max: Maximal amount of memory allowed for automatically selected

# send buffers for TCP socket. This value does not override

# net.core.wmem\_max, "static" selection via SO\_SNDBUF does not use this.

# Default: 128K

**net.ipv4.tcp\_wmem = 262144 262144 16777216**

#tcp\_rmem - vector of 3 INTEGERs: min, default, max

# min: Minimal size of receive buffer used by TCP sockets.

# It is guaranteed to each TCP socket, even under moderate memory

# pressure.

# Default: 8K

#

# default: default size of receive buffer used by TCP sockets.

# This value overrides net.core.rmem\_default used by other protocols.

# Default: 87380 bytes. This value results in window of 65535 with

# default setting of tcp\_adv\_win\_scale and tcp\_app\_win:0 and a bit

# less for default tcp\_app\_win. See below about these variables.

#

# max: maximal size of receive buffer allowed for automatically

# selected receiver buffers for TCP socket. This value does not override

# net.core.rmem\_max, "static" selection via SO\_RCVBUF does not use this.

# Default: 87380\*2 bytes..

**net.ipv4.tcp\_rmem = 4194304 4194304 16777216**

# Maximal number of remembered connection requests, which are

# still did not receive an acknowledgment from connecting client.

# Default value is 1024 for systems with more than 128Mb of memory,

# and 128 for low memory machines. If server suffers of overload,

# try to increase this number

**net.ipv4.tcp\_max\_syn\_backlog = 1280**

# help eliminate idle connections

**net.ipv4.tcp\_syncookies = 1**

# don't cache ssthresh from previous connection

**net.ipv4.tcp\_no\_metrics\_save = 1**

**net.ipv4.tcp\_moderate\_rcvbuf = 1**

# recommended to increase this for 1 GigE …for 10 GigE use 30000

**net.core.netdev\_max\_backlog = 2500**

# How frequent probes are retransmitted, when a probe isn't acknowledged. Default: 75 seconds.

**net.ipv4.tcp\_keepalive\_intvl = 75**

# How many keepalive probes TCP will send, until it decides that the connection is broken.

# Default value: 9. Multiplied with tcp\_keepalive\_intvl, this gives the time a link can be

# non-responsive after a keepalive has been sent.

**net.ipv4.tcp\_keepalive\_probes = 9**

# How often TCP sends out keepalive messages when keepalive is enabled. Default: 7200 secs

**net.ipv4.tcp\_keepalive\_time = 1800**

# Ignore broadcast messages

**net.ipv4.icmp\_echo\_ignore\_broadcasts = 1**

These should be entered in /etc/sysctl.conf and then can be loaded using the sysctl –p command as normal. In addition to the above, you may want to monitor swapping with vmstat and watch the si & so columns. If swapping seems to be happening, decrease the vm.swappiness kernel parameter from the default of 60 to something considerably lower.

## IO/Filesystem Tuning

Generally, with ASE systems using raw partitions, file system tuning is not necessary. However, if using file systems for tempdb devices, some basic file system tuning parameters are recommended. The exact tuning parameters will depend on the type of file system and features implemented. However, generally speaking the following apply:

* The file system block size should be the same size as the server page size. While smaller block size vs. page size is not as much an issue, typically the default block size is 4096 and many ASE systems use a default 2048 page size – which results in near doubling of the system calls. Consequently, it is best to simply match the block size and ASE page size.
* Benchmark high I/O using directio with different journaling options to determing the optimal configuration and journal location. For example, with ext3, some have found benefit with ***cache=writeback***. For xfs, consider ***logbufs=8***.
* Disable filesystem/directory modification time tracking with the mount options: ***noatime*** and ***nodiratime***.
* Consider tuning the ionode ratio for large files.
* If using a SAN, consider changing the default IO scheduler to NOOP or deadline (vs. default of CFQ) – especially if using a version of RedHat Enterprise Linux prior to RHEL 5.5. If using RHEL 5.5. or higher, you may wish to benchmark the system with both NOOP and CFQ schedulers under high concurrent load to determine the difference.
* At a script to rc.init to increase /sys/block/<devname>/queue/nr\_requests from default of 128 to at least 1024 or higher. This is critical. With the default settings, once the ASE (in conjunction with other processes such as backup server) has 128 IO’s outstanding in the device queue, future IO’s will synchronously block, causing high cpu usage within ASE (yet low OS cpu usage for ASE).
* monitor IO service and queue times with iostat –x. The 3rd and 2nd last columns (await and svctm) are key columns. Generally on enterprise storage, io service times should be ~2ms – higher is possible with different RAID levels (e.g. RAID 5), but if the service time exceeds 8ms, it is considered a slow IO. For Linux, the IO service time from ASE’s perspective is the ‘await’ column which includes the IO elevator queue time along with the actual physical service time. A high queue time points to nr\_requests issues or elevator tuning problems.

Because of the special mount options used, often the file systems used for database devices will be dedicated for database devices vs. normal file system usage. Especially if used for tempdb, this may allow mount options to skip fsck on reboots.

Changing the io scheduler can help significantly on older RHEL releases prior to 5.5 due to a bug in the CFQ scheduler. While this can be changed by modifying the /boot/grub/grub.conf – it also can be changed on a per device basis by:

## syntax for reading the current scheduler:

## cat /sys/block/{DEVICE-NAME}/queue/scheduler

cat /sys/block/sda/queue/scheduler

## syntax for changing the current scheduler:

## echo {SCHEDULER-NAME} > /sys/block/{DEVICE-NAME}/queue/scheduler

echo noop > /sys/block/hda/queue/scheduler

As with changing the number of requests (/sys/block/<devname>/queue/nr\_requests), using echo to change the value is not persistent past the next boot. As a result, you will need to create a script that does this each time and then add it to the rc.initd boot script. To avoid this, the best way to change it is to modify /boot/grub/grub.conf and add elevator=noop as a kernel parameter:

# grub.conf generated by anaconda

#

# Note that you do not have to rerun grub after making changes to this file

# NOTICE: You have a /boot partition. This means that

# all kernel and initrd paths are relative to /boot/, eg.

# root (hd0,0)

# kernel /vmlinuz-version ro root=/dev/VolGroup00/LogVol00

# initrd /initrd-version.img

#boot=/dev/sdk

default=0

timeout=10

splashimage=(hd0,0)/grub/splash.xpm.gz

hiddenmenu

title CentOS (2.6.18-164.11.1.el5xen)

root (hd0,0)

kernel /xen.gz-2.6.18-164.11.1.el5

module /vmlinuz-2.6.18-164.11.1.el5xen ro root=/dev/VolGroup00/LogVol00 rhgb elevator=noop

module /initrd-2.6.18-164.11.1.el5xen.img

title CentOS (2.6.18-164.11.1.el5)

root (hd0,0)

kernel /vmlinuz-2.6.18-164.11.1.el5 ro root=/dev/VolGroup00/LogVol00 rhgb elevator=noop

initrd /initrd-2.6.18-164.11.1.el5.img

title CentOS (2.6.18-164.el5xen)

root (hd0,0)

kernel /xen.gz-2.6.18-164.el5

module /vmlinuz-2.6.18-164.el5xen ro root=/dev/VolGroup00/LogVol00 rhgb quiet

module /initrd-2.6.18-164.el5xen.img

title CentOS-base (2.6.18-164.el5)

root (hd0,0)

kernel /vmlinuz-2.6.18-164.el5 ro root=/dev/VolGroup00/LogVol00 rhgb quiet

initrd /initrd-2.6.18-164.el5.img

The above changes the default I/O scheduler to the noop scheduler. In addition, there are a number of I/O scheduler tuning parameters for each I/O scheduler that may need to be tuned to provide optimal I/O. Changing the I/O scheduler should also be considered if you see a lot of await time in the iostat –x output. An example of the impact of the I/O scheduler and queue wait times can be seen in the following output from a iostat execution:

-- output from iostat –x

avg-cpu: %user %nice %system %iowait %steal %idle

1.36 0.00 3.50 11.15 0.00 83.99

Device: rrqm/s wrqm/s r/s w/s rsec/s wsec/s avgrq-sz avgqu-sz  **await svctm** %util

sda 0.00 8.82 0.00 1.00 0.00 78.56 78.40 0.02 24.80 24.80 2.48

sda1 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

sda2 0.00 8.82 0.00 1.00 0.00 78.56 78.40 0.02 24.80 24.80 2.48

sdb 0.00 0.80 41.28 12.02 330.26 98.60 8.05 1.17  **21.95 17.79**  94.83

sdb1 0.00 0.80 41.28 12.02 330.26 98.60 8.05 1.17  **21.95 17.79**  94.83

sdc 0.00 0.80 0.80 29.06 6.41 238.88 8.21 3.03 101.61 6.47 19.32

sdc1 0.00 0.80 0.80 29.06 6.41 238.88 8.21 3.03 101.61 6.47 19.32

sdd 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

sdd1 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

dm-0 0.00 0.00 0.00 9.82 0.00 78.56 8.00 0.23 23.84 2.53 2.48

dm-1 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00 0.00

In addition to the extremely slow IO service times (~18ms – more than double the 8ms at which point IOs are considered slow), the IO scheduler queueing effect added insult to injury by delaying the IOs another 4ms for a total of ~22ms per IO. Note that while other devices appear to have worse times, there is little IO activity on those devices, consequently the numbers are skewed.

Note that if using raw devices, raw devices in Linux used to be simply mapped to block vs. char devices – so the rc.init change to increase /sys/block/<devname>/queue/nr\_requests may still applicable….although later (RHEL 5.4), it appears that true char device support was available. Additionally, you will need to use udev to make sure the devices (by major and minor number) come back as the same devices. This is not unlike using the device label or the device UUID in the /etc/fstab to ensure the same file system devices are mounted in the same file systems. For example, even with file systems, if a disk is not recognized (say /dev/sdf) all the subsequent devices are simply shifted up a letter, so what used to be /dev/sdg is now /dev/sdf. If the fstab used the /dev device path for mount points, the wrong devices are mounted into the wrong filesystems and then ASE would have issues starting – or would not be able to recover properly. So, for example, you probably should use something like:

LABEL=raid\_cn0 /db\_devices/adaptec/raid\_cn0 xfs defaults,noatime,nodiratime,logbufs=8 0 0

or

UUID=09ca6f99-3018-45c6-8386-e04c5d2095cb /db\_devices/lsi/raid\_ssd0 xfs defaults,noatime,nodiratime,logbufs=8 0 0

in /etc/fstab instead of:

/dev/sdm /db\_devices/lsi/raid\_ssd0 xfs defaults,noatime,nodiratime,logbufs=8 0 0

…even for file systems. However, not all RAID controllers/disk pairs allow LABELs to work properly (e.g. the LSI controller above along with 4 SSD’s attached as a single RAID LUN does not), consequently in /etc/fstab, sometimes you are forced to use the device path at which point you are left to hope that all the devices are mounted in the correct locations. Using udev forces this to be true as device major and minor numbers. So, while raw device support requires udev to ensure that the raw devices are implemented and mounted at the same point, even using filesystem devices may require similar behavior in /etc/fstab