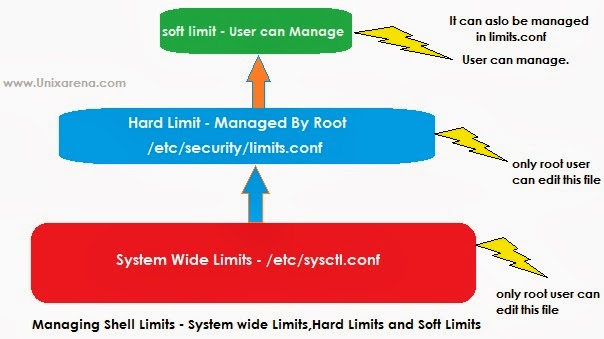
Ulimit Values in Linux

Shells like bash/csh/ksh are responsible to provide the control over various system resources to the user. Otherwise, one normal user may utilize the complete system resources and system won’t be available for other users. So setting the limit to users is very important and you need to be very careful before granting shell limits to them. You need to be always make sure that system is not going out its system wide limit. For an example , if the maximum system process limit is 64K and if you grated process limit to 4 users as 24K. When these all four users try to use the maximum no of process, system will run out of its limit and you will see fork errors on the system.

[](http://1.bp.blogspot.com/-pVq4SKqq5c4/UsG79z6cB1I/AAAAAAAACDQ/WI4bNDu3WDs/s1600/redhat+linux+limits+shell.jpg)

Here we are going to see how to set the soft limit and hard limit to the users and also we will see how to increase the system limit.

From ulimit man pages,

|  |  |
| --- | --- |
| **Options** | **Explanation** |
| -a | All current limits are reported |
| -b | The maximum socket buffer size |
| -c | The maximum size of core files created |
| -d | The maximum size of a processâs data segment |
| -e | The maximum scheduling priority (“nice”) |
| -f | The maximum size of files written by the shell and its children |
| -i | The maximum number of pending signals |
| -l | The maximum size that may be locked into memory |
| -m | The maximum resident set size (many systems do not honor this limit) |
| -n | The maximum number of open file descriptors (most systems do not allow this value to be set) |
| -p | The pipe size in 512-byte blocks (this may not be set) |
| -q | The maximum number of bytes in POSIX message queues |
| -r | The maximum real-time scheduling priority |
| -s | The maximum stack size |
| -t | The maximum amount of cpu time in seconds |
| -u | The maximum number of processes available to a single user |
| -v | The maximum amount of virtual memory available to the shell |
| -x | The maximum number of file locks |
| -t | The maximum number of threads |

**he  /etc/security/limits.conf  file**

Each line describes a limit for a user in the form:

domain       type       item        value        

Where **domain:**

can be:

       - an user name

       - a group name, with @group syntax

       - the wildcard \*, for default entry that means all user  and all group.

       - the wildcard %, can be also used with %group syntax, for maxlogin limit  
**type:**

can have the two values:

       - "soft" for enforcing the soft limits

       - "hard" for enforcing hard limits  
**item:**

can be one of the following:

       - core - limits the core file size (KB)

       - data - max data size (KB)

       - fsize - maximum filesize (KB)

       - memlock - max locked-in-memory address space (KB)

       - nofile - max number of open files

       - rss - max resident set size (KB)

       - stack - max stack size (KB)

       - cpu - max CPU time (MIN)

       - nproc - max number of processes

       - as - address space limit

       - maxlogins - max number of logins for this user

       - maxsyslogins - max number of logins on the system

       - priority - the priority to run user process with

       - locks - max number of file locks the user can hold

       - sigpending - max number of pending signals

       - msgqueue - max memory used by POSIX message queues (bytes)

       - nice - max nice priority allowed to raise to

       - rtprio - max realtime priority

Linux Tune Network Stack (Buffers Size) To Increase Networking Performance

The default maximum Linux TCP buffer sizes are way too small. TCP memory is calculated automatically based on system memory; you can find the actual values by typing the following commands:  
$ cat /proc/sys/net/ipv4/tcp\_mem  
The default and maximum amount for the receive socket memory:  
$ cat /proc/sys/net/core/rmem\_default  
$ cat /proc/sys/net/core/rmem\_max  
The default and maximum amount for the send socket memory:  
$ cat /proc/sys/net/core/wmem\_default  
$ cat /proc/sys/net/core/wmem\_max  
The maximum amount of option memory buffers:  
$ cat /proc/sys/net/core/optmem\_max

## Tune values

Set the max OS send buffer size (wmem) and receive buffer size (rmem) to 12 MB for queues on all protocols. In other words set the amount of memory that is allocated for each TCP socket when it is opened or created while transferring files:

http://s0.cyberciti.org/images/misc/warning-40px.png**WARNING!** The default value of rmem\_max and wmem\_max is about 128 KB in most Linux distributions, which may be enough for a low-latency general purpose network environment or for apps such as DNS / Web server. However, if the latency is large, the default size might be too small. Please note that the following settings going to increase memory usage on your server.

# echo 'net.core.wmem\_max=12582912' >> /etc/sysctl.conf  
# echo 'net.core.rmem\_max=12582912' >> /etc/sysctl.conf  
  
You also need to set minimum size, initial size, and maximum size in bytes:  
# echo 'net.ipv4.tcp\_rmem= 10240 87380 12582912' >> /etc/sysctl.conf  
# echo 'net.ipv4.tcp\_wmem= 10240 87380 12582912' >> /etc/sysctl.conf  
Turn on window scaling which can be an option to enlarge the transfer window:  
# echo 'net.ipv4.tcp\_window\_scaling = 1' >> /etc/sysctl.conf  
Enable timestamps as defined in RFC1323:  
# echo 'net.ipv4.tcp\_timestamps = 1' >> /etc/sysctl.conf  
Enable select acknowledgments:  
# echo 'net.ipv4.tcp\_sack = 1' >> /etc/sysctl.conf  
By default, TCP saves various connection metrics in the route cache when the connection closes, so that connections established in the near future can use these to set initial conditions. Usually, this increases overall performance, but may sometimes cause performance degradation. If set, TCP will not cache metrics on closing connections.  
# echo 'net.ipv4.tcp\_no\_metrics\_save = 1' >> /etc/sysctl.conf  
Set maximum number of packets, queued on the INPUT side, when the interface receives packets faster than kernel can process them.  
# echo 'net.core.netdev\_max\_backlog = 5000' >> /etc/sysctl.conf  
Now reload the changes:  
# sysctl -p  
Use tcpdump to view changes for eth0:  
# tcpdump -ni eth0

Another thing you can do to help increase TCP throughput with 1GB NICs is to increase the size of the interface queue. For paths with more than 50 ms RTT, a value of 5000-10000 is recommended. To increase txqueuelen, do the following:

|  |  |
| --- | --- |
|  | [root@server1 ~] ifconfig eth0 txqueuelen 5000 |

TCP\_FIN\_TIMEOUT - This setting determines the time that must elapse before TCP/IP can release a closed connection and reuse its resources. During this TIME\_WAIT state, reopening the connection to the client costs less than establishing a new connection. By reducing the value of this entry, TCP/IP can release closed connections faster, making more resources available for new connections. Addjust this in the presense of many connections sitting in the TIME\_WAIT state:

|  |  |
| --- | --- |
|  | [root@server:~]# echo 30 > /proc/sys/net/ipv4/tcp\_fin\_timeout |

TCP\_KEEPALIVE\_INTERVAL - This determines the wait time between isAlive interval probes. To set:

|  |  |
| --- | --- |
|  | [root@server:~]# echo 30 > /proc/sys/net/ipv4/tcp\_keepalive\_intvl |

TCP\_KEEPALIVE\_PROBES - This determines the number of probes before timing out. To set:

|  |  |
| --- | --- |
|  | [root@server:~]# echo 5 > /proc/sys/net/ipv4/tcp\_keepalive\_probes |

TCP\_TW\_RECYCLE - This enables fast recycling of TIME\_WAIT sockets. The default value is 0 (disabled). Should be used with caution with loadbalancers.

|  |  |
| --- | --- |
|  | [root@server:~]# echo 1 > /proc/sys/net/ipv4/tcp\_tw\_recycle |

TCP\_TW\_REUSE - This allows reusing sockets in TIME\_WAIT state for new connections when it is safe from protocol viewpoint. Default value is 0 (disabled). It is generally a safer alternative to tcp\_tw\_recycle

|  |  |
| --- | --- |
|  | [root@server:~]# echo 1 > /proc/sys/net/ipv4/tcp\_tw\_reuse |

Note: The tcp\_tw\_reuse setting is particularly useful in environments where numerous short connections are open and left in TIME\_WAIT state, such as web servers and loadbalancers. Reusing the sockets can be very effective in reducing server load.

Starting in Linux 2.6.7 (and back-ported to 2.4.27), linux includes alternative congestion control algorithms beside the traditional 'reno' algorithm. These are designed to recover quickly from packet loss on high-speed WANs.

There are a couple additional sysctl settings for kernels 2.6 and newer:

Not to cache ssthresh from previous connection:

|  |  |
| --- | --- |
|  | net.ipv4.tcp\_no\_metrics\_save = 1 |

To increase this for 10G NICS:

|  |  |
| --- | --- |
|  | net.core.netdev\_max\_backlog = 30000 |

Starting with version 2.6.13, Linux supports pluggable congestion control algorithms . The congestion control algorithm used is set using the sysctl variable net.ipv4.tcp\_congestion\_control, which is set to bic/cubic or reno by default, depending on which version of the 2.6 kernel you are using.

To get a list of congestion control algorithms that are available in your kernel (if you are running 2.6.20 or higher), run:

|  |  |
| --- | --- |
|  | [root@server1 ~] # sysctl net.ipv4.tcp\_available\_congestion\_control |

\* reno: Traditional TCP used by almost all other OSes. (default)

\* cubic: CUBIC-TCP (NOTE: There is a cubic bug in the Linux 2.6.18 kernel used by Redhat Enterprise Linux 5.3 and Scientific Linux 5.3. Use 2.6.18.2 or higher!)

\* bic: BIC-TCP

\* htcp: Hamilton TCP

\* vegas: TCP Vegas

\* westwood: optimized for lossy networks

If cubic and/or htcp are not listed when you do 'sysctl net.ipv4.tcp\_available\_congestion\_control', try the following, as most distributions include them as loadable kernel modules:

|  |  |
| --- | --- |
|  | [root@server1 ~] # /sbin/modprobe tcp\_htcp |
|  | [root@server1 ~] # /sbin/modprobe tcp\_cubic |

For long fast paths, I highly recommend using cubic or htcp. Cubic is the default for a number of Linux distributions, but if is not the default on your system, you can do the following:

|  |  |
| --- | --- |
|  | [root@server1 ~] # sysctl -w net.ipv4.tcp\_congestion\_control=cubic |

On systems supporting RPMS, You can also try using the ktune RPM, which sets many of these as well.

If you have a load server that has many connections in TIME\_WAIT state decrease the TIME\_WAIT interval that determines the time that must elapse before TCP/IP can release a closed connection and reuse its resources. This interval between closure and release is known as the TIME\_WAIT state or twice the maximum segment lifetime (2MSL) state. During this time, reopening the connection to the client and server cost less than establishing a new connection. By reducing the value of this entry, TCP/IP can release closed connections faster, providing more resources for new connections. Adjust this parameter if the running application requires rapid release, the creation of new connections, and a low throughput due to many connections sitting in the TIME\_WAIT state:

|  |  |
| --- | --- |
|  | [root@host1 ~]# echo 5 > /proc/sys/net/ipv4/tcp\_fin\_timeout |

If you are often dealing with SYN floods the following tunning can be helpful:

|  |  |
| --- | --- |
|  | [root@host1 ~]# sysctl -w net.ipv4.tcp\_max\_syn\_backlog="16384" |
|  | [root@host1 ~]# sysctl -w net.ipv4.tcp\_synack\_retries="1" |
|  | [root@host1 ~]# sysctl -w net.ipv4.tcp\_max\_orphans="400000" |

The parameter on line 1 is the maximum number of remembered connection requests, which still have not received an acknowledgment from connecting clients.  
The parameter on line 2 determines the number of SYN+ACK packets sent before the kernel gives up on the connection. To open the other side of the connection, the kernel sends a SYN with a piggybacked ACK on it, to acknowledge the earlier received SYN. This is part 2 of the three-way handshake.  
And lastly on line 3 is the maximum number of TCP sockets not attached to any user file handle, held by system. If this number is exceeded orphaned connections are reset immediately and warning is printed. This limit exists only to prevent simple DoS attacks, you \_must\_ not rely on this or lower the limit artificially, but rather increase it (probably, after increasing installed memory), if network conditions require more than default value, and tune network services to linger and kill such states more aggressively.   
  
More information on tuning parameters and defaults for Linux 2.6 are available in the file ip-sysctl.txt, which is part of the 2.6 source distribution.

Warning on Large MTUs: If you have configured your Linux host to use 9K MTUs, but the connection is using 1500 byte packets, then you actually need 9/1.5 = 6 times more buffer space in order to fill the pipe. In fact some device drivers only allocate memory in power of two sizes, so you may even need 16/1.5 = 11 times more buffer space!

And finally a warning for both 2.4 and 2.6: for very large BDP paths where the TCP window is > 20 MB, you are likely to hit the Linux SACK implementation problem. If Linux has too many packets in flight when it gets a SACK event, it takes too long to located the SACKed packet, and you get a TCP timeout and CWND goes back to 1 packet. Restricting the TCP buffer size to about 12 MB seems to avoid this problem, but clearly limits your total throughput. Another solution is to disable SACK.

Starting with Linux 2.4, Linux implemented a sender-side autotuning mechanism, so that setting the optimal buffer size on the sender is not needed. This assumes you have set large buffers on the receive side, as the sending buffer will not grow beyond the size of the receive buffer.

However, Linux 2.4 has some other strange behavior that one needs to be aware of. For example: The value for ssthresh for a given path is cached in the routing table. This means that if a connection has has a retransmission and reduces its window, then all connections to that host for the next 10 minutes will use a reduced window size, and not even try to increase its window. The only way to disable this behavior is to do the following before all new connections (you must be root):

|  |  |
| --- | --- |
|  | [root@server1 ~] # sysctl -w net.ipv4.route.flush=1 |

[**view raw**](https://gist.github.com/kaivanov/5e79024da963261c66a5/raw/3b5c3421626647e0d422ea88c7248e629df76360/gistfile1.sh)[**gistfile1.sh**](https://gist.github.com/kaivanov/5e79024da963261c66a5#file-gistfile1-sh) hosted with ❤ by **[GitHub](https://github.com/)**

I would like to also point out how important it is to have a sufficient number of available file descriptors, since pretty much everything on Linux is a file.   
To check your current max and availability run the following:

|  |  |
| --- | --- |
|  | [root@host1 ~]# sysctl fs.file-nr |
|  | fs.file-nr = 197600 0 3624009 |

[**view raw**](https://gist.github.com/kaivanov/20f60c1d272c7afdb11d/raw/6b438a264ad29cffbb377ddbbabcc7ea7f310fe4/gistfile1.sh)[**gistfile1.sh**](https://gist.github.com/kaivanov/20f60c1d272c7afdb11d#file-gistfile1-sh) hosted with ❤ by **[GitHub](https://github.com/)**

The first value (197600) is the number of allocated file handles.  
The second value (0) is the number of unused but allocated file handles. And the third value (3624009) is the system-wide maximum number of file handles. It can be increased by tuning the following kernel parameter:

|  |  |
| --- | --- |
|  | [root@host1 ~]# echo 10000000 > /proc/sys/fs/file-max |

[**view raw**](https://gist.github.com/kaivanov/4138519b3bb9b3cc9965/raw/4c3c9093e84fcac22a1444e6d259780993bad892/gistfile1.sh)[**gistfile1.sh**](https://gist.github.com/kaivanov/4138519b3bb9b3cc9965#file-gistfile1-sh) hosted with ❤ by **[GitHub](https://github.com/)**

To see how many file descriptors are being used by a process you can use one of the following:

|  |  |
| --- | --- |
|  | [root@host1 ~]# lsof -a -p 28290 |
|  | [root@host1 ~]# ls -l /proc/28290/fd | wc -l |

The 28290 number is the process id.   
  
An finally if you are using stateful iptable rules the nf\_conntrack kernel module might run out of memory for connection tracking and an error will be logged: nf\_conntrack: table full, dropping packet   
  
In order to raise that limit, therefore allocate more memory, you need to calculate how much RAM each connection uses. You can get that information from the proc file /proc/slabinfo.   
The nf\_conntrack entry show the active entries, and how big each object is, and how many fit in a slab (each slab fits in one or more kernel page, usually 4K if not using hugepages). Accounting for the overhead of the kernel page size you can see from the slabinfo that each nf\_conntrack object takes about 316 bytes (this will differ on different systems). So to track 1M connections you'll need to allocate roughly 316 MB of memory.

|  |  |
| --- | --- |
|  | [root@host1 ~]# sysctl net.netfilter.nf\_conntrack\_count |
|  | [root@host1 ~]# sysctl net.netfilter.nf\_conntrack\_max |
|  | [root@host1 ~]# sysctl -w net.netfilter.nf\_conntrack\_max=1000000 |
|  | [root@host1 ~]# echo 250000 > /sys/module/nf\_conntrack/parameters/hashsize # hashsize = nf\_conntrack\_max / 4  LIMIT THE SIZE OF YOUR CORE FILES ON LINUX  **ulimit -c** *max-size* |

### I/O and File System Tuning

***For example, for pure file server applications like web and samba servers, you probably want to disable the "atime" option on the most used filesystem.*** This disabled updating the "atime" value for the file, which indicates that the last time a file was accessed. Since this info isn't very useful in this situation, and 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 ext2 noatime 1 2

1. **For fast disk subsystems, it is desirable to use large flushes of dirty memory pages.**

The value stored in **/proc/sys/vm/dirty\_background\_ratio** defines at what percentage of main memory the**pdflush** daemon should write data out to the disk.

If larger flushes are desired then increasing the default value of 10% to a larger value will cause less frequent flushes.

As in the example above the value can be changed to 25 as shown in

**# sysctl -w vm.dirty\_background\_ratio=25**

**vmstat :**

vmstat is a tool in Unix/Linux which is used to Report virtual memory statistics. It shows how much virtual memory there is, how much is free and paging activity. Most important, you can observe page-ins and page-outs as they happen.

**vmstat**  reports  information about processes, memory, paging, block IO, traps, and cpu activity.

> vmstat

procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----

 r  b   swpd   free   buff  cache        si   so    bi    bo   in   cs us sy id wa st

 0  0  21688 12797396 850716 15372668    0    0     3    35    0    0  1  0 99  0  0

 0  0  21688 12789292 850716 15372668    0    0     0  1355  883  744  1  0 99  0  0

 1  0  21688 12784648 850716 15372676    0    0     0    15 1934 1292  6  0 93  0  0

 2  0  21688 12781356 850716 15372736    0    0     0    16 2222 2094  8  1 92  0  0

 0  0  21688 12809992 850716 15376880    0    0     0    16 1959 1621  6  1 93  0  0

 1  0  21688 12809992 850716 15376888    0    0     0    29 1090  860  1  0 99  0  0

 0  0  21688 12808936 850716 15376888    0    0     0   132  931  798  1  0 99  0  0

 0  0  21688 12809496 850716 15376888    0    0     0     5  868  768  1  0 99  0  0

 1  0  21688 12793492 850716 15376888    0    0     0     5 1701 1177  5  0 95  0  0

 1  0  21688 12767728 850716 15376928    0    0     0  1355 8553 2589 48  1 51  0  0

13  0  21688 12765680 850716 15376968    0    0     0    24 7619 3122 44  1 56  0  0

Fields explained as follows:

**Under Procs we have**

       r: The number of processes waiting for run time or placed in run queue or are already executing (running)

       b: The number of processes in uninterruptible sleep. (b=blocked queue, waiting for resource (e.g. filesystem I/O blocked, inode lock))

If runnable threads (r) divided by the number of CPU is greater than one -> possible CPU bottleneck

(The (r) coulmn should be compared with number of CPUs (logical CPUs as in uptime) if we have enough CPUs or we have more threads.)

High numbers in the blocked processes column (b) indicates slow disks.

(r) should always be higher than (b); if it is not, it usually means you have a CPU bottleneck  
  
Note: “cat /proc/cpuinfo” dispalys the cpu info on the machine

cat /proc/cpuinfo|grep processor|wc -l

output: 16

Remember that we need to know the number of CPUs on our server because the vmstat r value must never exceed the number of CPUs. r value of 13 is perfectly acceptable for a 16-CPU server, while a value of 16 would be a serious problem for a 12-CPU server.

Whenever the value of the r column exceeds the number of CPUs on the server, tasks are forced to wait for execution.There are several solutions to managing CPU overload, and these alternatives are:

1.      Add more processors (CPUs) to the server.

2.      Load balance the system tasks by rescheduling large batch tasks to execute during off-peak hours.

**Under Memory we have:**

swpd: shows how many blocks are swapped out to disk (paged). The amount of Virtual memory used.  
            Note: you can see the swap area configured in server using "cat proc/swaps"  
>cat /proc/swaps

Filename                                Type            Size    Used    Priority

/dev/dm-7                               partition       16777208        21688   -1

free: The amount of Idle Memory  
buff: Memory used as buffers, like before/after I/O operations  
cache: Memory used as cache by the Operating System

**Under Swap we have:**

si: Amount of memory swapped in from disk (/s). This shows page-ins

so: Amount of memory swapped to disk (/s). This shows page-outs. The so column is zero consistently, indicating there are no page-outs.

In Ideal condition, si and so should be at 0 most of the time, and we definitely don’t like to see more than 10 blocks per second.

**Under IO we have:**

bi: Blocks received from block device - Read (like a hard disk)  
bo: Blocks sent to a block device – Write

**Under System we have:**

in: The number of interrupts per second, including the clock.  
cs: The number of context switches per second.

(A context switch occurs when the currently running thread is different from the previously running thread, so it is taken off of the CPU.)

It is not uncommon to see the context switch rate be approximately the same as device interrupt rate (in column)

If**cs is high**, it may indicate too much process switching is occurring, thus using memory inefficiently.

If **cs is higher then sy**, system is doing more context switching than actual work.

**High r with high cs -> possible lock contention**  
Lock contention occurs whenever one process or thread attempts to acquire a lock held by another process or thread. The more granular the available locks, the less likely one process/thread will request a lock held by the other. (For example, locking a row rather than the entire table, or locking a cell rather than the entire row.)  
  
When you are seeing blocked processes or high values on waiting on I/O (wa), it usually signifies either real I/O issues where you are waiting for file accesses or an I/O condition associated with paging due to a lack of memory on your system.

**Under CPU we have:**

These are percentages of total CPU time.

       us: % of CPU time spent in user mode (not using kernel code, not able to acces to kernel resources). Time spent running non-kernel code. (user time, including nice time)

       sy: % of CPU time spent running kernel code. (system time)

       id: % of CPU  idle time

       wa: % of CPU time spent waiting for IO.

Note: the memory, swap, and I/O statistics are in blocks, not in bytes. In Linux, blocks are usually 1,024 bytes (1 KB).

**To measure true idle time measure id+wa together:**  
**- if id=0%**, it does not mean all CPU is consumed, because "wait" (wa) can be 100% and waiting for an I/O to complete  
  
**- if wait=0%**, it does not mean I have no I/O waiting issues, because as long I have threads which keep the CPU busy I could have additional threads waiting for I/O, but this will be masked by the running threads  
  
If process A is running and process B is waiting on I/O, the wait% still would have a 0 number.

A 0 number doesn't mean I/O is not occurring, it means that the system is not waiting on I/O.

If process A and process B are both waiting on I/O, and there is nothing that can use the CPU, then you would see that column increase.  
  
**- if wait%** **is high,** it does not mean I have io performance problem, it can be an indication that I am doing some IO but the cpu is not kept busy at all  
  
**- if id% is high** then likely there is no CPU or I/O problem  
  
  
**To measure cpu utilization measure us+sy together (and compare it to physc):**  
**- if us+sy is always greater than 80%**, then CPU is approaching its limits

**- if us+sy = 100%** -> possible CPU bottleneck  
  
**- if sy is high,** your appl. is issuing many system calls to the kernel and asking the kernel to work. It measures how heavily the appl. is using kernel services.  
  
**- if sy  is higher than us**, this means your system is spending less time on real work (not good)

### [What is swappiness and how do we change its value?](http://www.golinuxhub.com/2014/01/what-is-swappiness-and-how-do-we-change.html)

It is a feature in Linux which controls the degree to which the kernel prefers to swap in the procedure of freeing memory. It can be set to values on a scale from 0 to 100. A low value means the kernel will try to avoid swapping as much as possible unless there is almost no [free memory](javascript:void(0);) let on the RAM for any new process. On the other side a higher value would force kernel aggressivey to swap out pages from the physical memory.  
  
The default value for Linux machines is 60. Using a higher value wiil affect the system negatively as accessing a [hard](javascript:void(0);) disk(swap space) for each and every request by a [application](javascript:void(0);) program is a very slow process as compared to doing the same from physical memory. So it should be avoided to transfer active pages to swap space aggressively.  
  
To check the current swappiness value  
$ cat /proc/sys/vm/swappiness  
60  
To change the value  
# echo 40 > /proc/sys/vm/swappiness  
To make the changes affect  
# sysctl -p  
Verify the new parameter  
# sysctl -a | grep swappiness  
vm.swappiness = 40

### [What is Virtual memory, paging and swap space ?](http://www.golinuxhub.com/2014/01/what-is-virtual-memory-paging-and-swap.html)

### Swapping

A process normally runs on physical memory where the memory is divided into sets of pages. A page is a 4kb area of memory and is the basic unit of memory with which both [kernel](javascript:void(0);) and CPU deal.  
  
There might be a situation when all the pages in physical memory goes full. In such cases all the inactive pages inside physical memory is shifted to the secondary storage or the swap space using the paging technique. By doing this physical memory gets free pages which can again be utilized by new processes. This entire process is termed as swapping.  
  
**NOTE:** Swapping is a good idea as it gives you an additional space to store data files and programs when your physical memory is out of space but accessing a [hard disk](javascript:void(0);) is hundred times slower than accessing memory.  
  
[Virtual memory](javascript:void(0);) is a memory management technique that is implemented using both hardware and software which gives an application program the [impression](javascript:void(0);) that it has contiguous working memory (an address space).  
  
In simple terms Virtual memory is a logical combination of RAM memory and swap space which is used by running process  
  
**NOTE:** It is NOT just an additional space used in hard disk to make it act as physical memory

### Paging

This is one of the memory management technique [schemes](javascript:void(0);) by which a computer can store and retrieve data from secondary storage for use in main memory.

### Swap space

This is a space on the hard disk which is used by the [operating system](javascript:void(0);) to store data pages that are currently not needed. This swap sapce can be a partition as well as swap file.  
  
**Amount of RAM in the system recommended amount of swap space**  
4GB of RAM or less  a minimum of 2GB of swap space  
4GB to 16GB of RAM  a minimum of 4GB of swap space  
16GB to 64GB of RAM  a minimum of 8GB of swap space  
64GB to 256GB of RAM  a minimum of 16GB of swap space  
256GB to 512GB of RAM  a minimum of 32GB of swap space

replace the keyword single with the keyword "emergency".

### ****16.**** ****In the ps results few of the processes are having process state as "D" . What does it mean ? Briefly explain different process states ?****

To have a dynamic view of a process in Linux, always use the top command. This command provides a real-time view of the Linux system in terms of processes. The eighth column in the output of this command represents the current state of processes. A process state gives a broader indication of whether the process is currently running, stopped, sleeping etc.

A process in Linux can have any of the following four states…

**Running** – A process is said to be in a running state when either it is actually running/ executing or waiting in the scheduler’s queue to get executed (which means that it is ready to run). That is the reason that this state is sometimes also known as ‘runnable’ and represented by (R).

**Waiting or Sleeping** – A process is said to be in this state if it is waiting for an event to occur or waiting for some resource-specific operation to complete. So, depending upon these scenarios, a waiting state can be subcategorised into an interruptible (S) or uninterruptible (D) state respectively.

**Stopped** – A process is said to be in the stopped state when it receives a signal to stop. This usually happens when the process is being debugged. This state is represented by (T).

**Zombie** – A process is said to be in the zombie state when it has finished execution but is waiting for its parent to retrieve its exit status. This state is represented by (Z).

Apart from these four states, the process is said to be dead after it crosses over the zombie state; ie when the parent retrieves its exit status. ‘Dead’ is not exactly a state, since a dead process ceases to exist.

### ****17.**** ****What is drop cache in Linux and how do you clear it ?****

Cache in Linux memory is where the Kernel stores the information it may need later, as memory is incredible faster than disk.

It is great that the Linux Kernel takes care about that.Linux Operating system is very efficient in managing your computer memory, and will automatically free the RAM and drop the cache if some application needs memory.

Kernels 2.6.16 and newer provide a mechanism to have the kernel drop the page cache and/or inode and dentry caches on command, which can help free up a lot of memory. Now we can throw away that script that allocated a ton of memory just to get rid of the cache.

To free pagecache:

# echo 1 > /proc/sys/vm/drop\_caches

To free dentries and inodes:

# echo 2 > /proc/sys/vm/drop\_caches

To free pagecache, dentries and inodes:

echo 3 > /proc/sys/vm/drop\_caches

This is a non-destructive operation in normal scenarios and will only free things that are completely unused. Dirty objects will continue to be in use until written out to disk and are not freeable. However it is always preferred to run "sync" first to flush useful things out to disk.

#### How to Clear Cache in Linux?

Every Linux System has three options to clear cache without interrupting any processes or services.

**1.** Clear PageCache only.

# sync; echo 1 > /proc/sys/vm/drop\_caches

**2.** Clear dentries and inodes.

# sync; echo 2 > /proc/sys/vm/drop\_caches

**3.** Clear PageCache, dentries and inodes.

# sync; echo 3 > /proc/sys/vm/drop\_caches

**sync** will flush the file system buffer.

#### How to Clear Swap Space in Linux?

If you want to clear Swap space, you may like to run the below command.

# swapoff -a && swapon -a

## Interpreting Free

To see how much memory you are currently using, run free -m.  It will provide output like:

:~$ free -m

total used free shared buffers cached

Mem: 2008 1951 57 0 142 575

-/+ buffers/cache: 1234 774

Swap: 3812 35 3777

The top row 'used' (1951) value will almost always nearly match the top row total value (2008). Since Linux likes to use any spare memory to cache disk blocks.

The most important 'used' figure to look at is the buffers/cache row used value (1234). This is how much space your applications are currently using. For best performance, this number should be less than your total (2008) memory. To prevent out of memory errors, it needs to be less than the total memory (2008) and swap space (3812).

If you wish to quickly see how much memory is free look at the buffers/cache row free value (774). This is the total memory (2008) - the actual used (1234). (2008 - 1234 = 774)

Note that the kernel does need some RAM for caching, to help improve system performance of some subsystems, for example disk writes. If you are seeing some slowness, and RAM looks ok but tight, adding more RAM (or reducing the amount used by applications) is a good idea.

# 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 1 2

## 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](https://tweaked.io/benchmarking/" \l "fs).

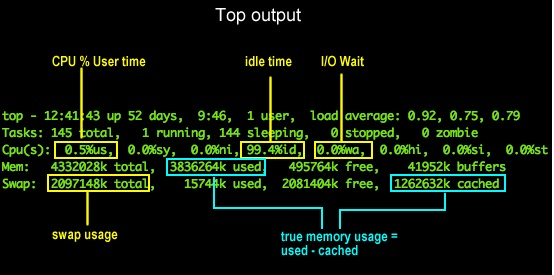
**MEMORY**

Resident memory, labelled RES: How much physical memory, how much RAM, your process is using. **RES is the important number**

Virtual memory, labelled VIRT: How much memory your process thinks it’s using. Usually much bigger than RES.

SHR indicates how much of the VIRT size is actually sharable (memory or libraries).

**Performance issue’s:-**



## STEP 1: CHECK I/O WAIT AND CPU IDLETIME

**How:** use top - look for "wa" (I/O wait) and "id" (CPU idletime)

**Why:** checking I/O wait is the best initial step to narrow down the root cause of server slowness. If I/O wait is low, you can rule out disk access in your diagnosis.

I/O Wait represents the amount of time the CPU waiting for disk or network I/O. **Waiting** is the key here - if your CPU is waiting, it's not doing useful work. It's like a chef who can't serve a meal until he gets a delivery of ingredients. Anything above 10% I/O wait should be considered high.

On the other hand, CPU idle time is a metric you WANT to be high -- the higher this is, the more bandwidth your server has to handle whatever else you throw at it. If your idle time is consistently above 25%, consider it "high enough"

## STEP 2: IO WAIT IS LOW AND IDLE TIME IS LOW: CHECK CPU USER TIME

**How:** use top again -- look for the %us column (first column), then look for a process or processes that is doing the damage.

**Why:** at this point you **expect** the usertime percentage to be high -- there's most likely a program or service you've configured on you server that's hogging CPU. Checking the % user time just confirms this. When you see that the % usertime is high, it's time to see what executable is monopolizing the CPU

Once you've confirmed that the % usertime is high, check the process list (also provided by top). Be default, top sorts the process list by %CPU, so you can just look at the top process or processes.

If there's a single process hogging the CPU in a way that seems abnormal, it's an anomalous situation that a service restart can fix. If there are are multiple processes taking up CPU resources, or it there's one process that takes lots of resources while otherwise functioning normally, than your setup may just be underpowered. You'll need to upgrade your server (add more cores), or split services out onto other boxes. In either case, you have a resolution:

* if situation seems anomalous: **kill the offending processes.**
* if situation seems typical given history: **upgrade server or add more servers.**

## STEP 3: IO WAIT IS LOW AND IDLE TIME IS HIGH

Your slowness isn't due to CPU or IO problems, so it's likely an application-specific issue. It's also possible that the slowness is being caused by another server in your cluster, or by an external service you rely on.

* start by checking important applications for uncharacteristic slowness (the DB is a good place to start),
* think through which parts of your infrastructure could be slowed down externally. For example, do you use an externally hosted email service that could slow down critical parts of your application?

If you suspect another server in your cluster, [strace](http://stackoverflow.com/questions/174942/how-to-use-strace) and [lsof](http://www.catonmat.net/blog/unix-utilities-lsof/) can provide information on what the process is doing or waiting on. Strace will show you which file descriptors are being read or written to (or being attempted to be read from) and lsof can give you a mapping of those file descriptors to network connections.

## STEP 4: IO WAIT IS HIGH: CHECK YOUR SWAP USAGE

**How:** use top or free -m

**Why:** if your box is swapping out to disk a lot, the cache swaps will monopolize the disk and processes with legitimate IO needs will be starved for disk access. In other words, checking disk swap separates "real" IO wait problems from what are actually RAM problems that "look like" IO Wait problems.

An alternative to top is free -m -- this is useful if you find top's frequent updates frustrating to use, and you don't have any console log of changes.

## STEP 5: SWAP USAGE IS HIGH

High swap usage means that you are actually out of RAM. See step 6 below.

## STEP 6: SWAP USAGE IS LOW

Low swap means you have a "real" IO wait problem. The next step is to see what's hogging your IO.

**How:** iotop

[iotop](http://www.cyberciti.biz/hardware/linux-iotop-simple-top-like-io-monitor/) is an awesome tool for identifying io offenders. Two things to note:

1. unless you've already installed iotop, it's probably not already on your system. Recommendation: install it before you need it -- it's no fun trying to install a troubleshooting tool on an overloaded machine.
2. iotop requies a Linux of 2.62 or above

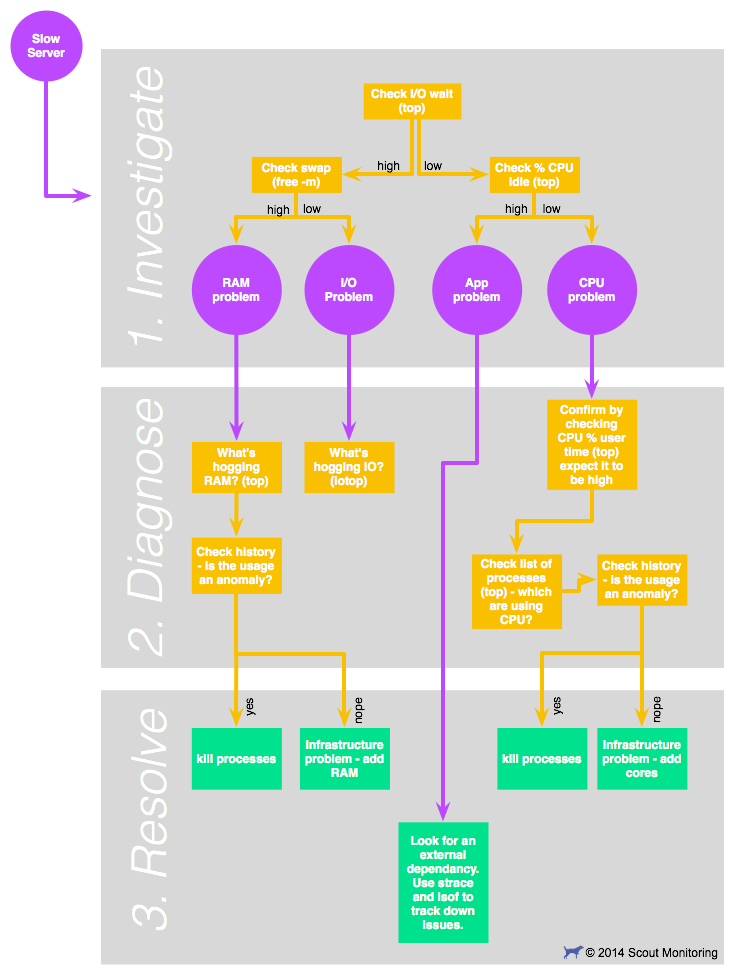
## STEP 7: CHECK MEMORY USAGE

**How:** use top. Once top is running, press the M key - this will sort applications by the memory used.

Important: don't look at the "free" memory -- it's misleading. To get the actual memory available, subtract the "cached" memory from the "used" memory. This is because Linux caches things liberally, and often the memory can be freed up when it's needed. Read here (http://blog.scoutapp.com/articles/2010/10/06/determining-free-memory-on-linux) for more info.

Once you've identified the offenders, the resolution will again depend on whether their memory usage seems business-as-usual or not. For example, a memory leak can be satisfactorily addressed by a one-time or periodic restart of the process.

* if memory usage seems anomalous: **kill the offending processes.**if memory usage seems business-as-usual: **add RAM to the server, or split high-memoryusinservices**

[](https://dl.dropboxusercontent.com/u/18554/slow%20server%20troubleshooting.png)

### ADDITIONAL TIPS

* vmstat is also a very handy tool, because it shows past values instead of an in-place update like top. Running vmstat 1 shows concise metrics on memory, swap, io, and CPU every second.
* Track your disk IO latency and compare to IOPS (I/O operations per second). Sometimes it's not activity in your own server causing the disk IO to be slow in a cloud/virtual environment. Proving this is hard, and you really want to have graphs of historical performance to show your provider!
* Increasing IO latency can mean a failing disk or bad sectors. Keep an eye on this before it escalates to data corruption or complete failure of the disk.
* If your a visual person, Scout's dashboards can help - your data will look like this:

# UNDERSTANDING LINUX CPU LOAD - WHEN SHOULD YOU BE WORRIED?

You might be familiar with Linux load averages already. Load averages are the three numbers shown with the uptime and top commands - they look like this:

load average: 0.09, 0.05, 0.01

Most people have an inkling of what the load averages mean: the three numbers represent averages over progressively longer periods of time (one, five, and fifteen minute averages), and that lower numbers are better. Higher numbers represent a problem or an overloaded machine. But, what's the the threshold? What constitutes "good" and "bad" load average values? When should you be concerned over a load average value, and when should you scramble to fix it ASAP?

First, a little background on what the load average values mean. We'll start out with the simplest case: a machine with one single-core processor.

## THE TRAFFIC ANALOGY

A single-core CPU is like a single lane of traffic. Imagine you are a bridge operator ... sometimes your bridge is so busy there are cars lined up to cross. You want to let folks know how traffic is moving on your bridge. A decent metric would be how many cars are waiting at a particular time. If no cars are waiting, incoming drivers know they can drive across right away. If cars are backed up, drivers know they're in for delays.

So, Bridge Operator, what numbering system are you going to use? How about:

* **0.00 means there's no traffic on the bridge at all**. In fact, between 0.00 and 1.00 means there's no backup, and an arriving car will just go right on.
* **1.00 means the bridge is**exactly**at capacity.** All is still good, but if traffic gets a little heavier, things are going to slow down.
* **over 1.00 means there's backup.** How much? Well, 2.00 means that there are two lanes worth of cars total -- one lane's worth on the bridge, and one lane's worth waiting. 3.00 means there are three lane's worth total -- one lane's worth on the bridge, and two lanes' worth waiting. Etc.

https://dl.dropboxusercontent.com/u/468982/blog/load_averages/20090728-jek9ssauydsi19nbcja26tw8ju.png = load of 1.00

https://dl.dropboxusercontent.com/u/468982/blog/load_averages/20090728-c3278n4dj5t766u5mcjhwb2h57.png = load of 0.50

https://dl.dropboxusercontent.com/u/468982/blog/load_averages/20090728-89jd6aydgwd9j26in49h7y1n7g.png = load of 1.70

This is basically what CPU load is. "Cars" are processes using a slice of CPU time ("crossing the bridge") or queued up to use the CPU. Unix refers to this as the run-queue length: the sum of the number of processes that are currently running plus the number that are waiting (queued) to run.

Like the bridge operator, you'd like your cars/processes to never be waiting. So, your CPU load should ideally stay below 1.00. Also like the bridge operator, you are still ok if you get some temporary spikes above 1.00 ... but when you're consistently above 1.00, you need to worry.

## SO YOU'RE SAYING THE IDEAL LOAD IS 1.00?

Well, not exactly. The problem with a load of 1.00 is that you have no headroom. In practice, many sysadmins will draw a line at 0.70:

* The **"Need to Look into it"** Rule of Thumb: **0.70** If your load average is staying above > 0.70, it's time to investigate before things get worse.
* The **"Fix this now"** Rule of Thumb: **1.00**. If your load average stays above 1.00, find the problem and fix it now. Otherwise, you're going to get woken up in the middle of the night, and it's not going to be fun.
* The **"Arrgh, it's 3AM WTF?"** Rule of Thumb: **5.0**. If your load average is above 5.00, you could be in serious trouble, your box is either hanging or slowing way down, and this will (inexplicably) happen in the worst possible time like in the middle of the night or when you're presenting at a conference. Don't let it get there.

## WHAT ABOUT MULTI-PROCESSORS? MY LOAD SAYS 3.00, BUT THINGS ARE RUNNING FINE!

Got a quad-processor system? It's still healthy with a load of 3.00.

On multi-processor system, the load is relative to the number of processor cores available. The "100% utilization" mark is 1.00 on a single-core system, 2.00, on a dual-core, 4.00 on a quad-core, etc.

If we go back to the bridge analogy, the "1.00" really means "one lane's worth of traffic". On a one-lane bridge, that means it's filled up. On a two-late bridge, a load of 1.00 means its at 50% capacity -- only one lane is full, so there's another whole lane that can be filled.

Same with CPUs: a load of 1.00 is 100% CPU utilization on single-core box. On a dual-core box, a load of 2.00 is 100% CPU utilization.

## MULTICORE VS. MULTIPROCESSOR

While we're on the topic, let's talk about multicore vs. multiprocessor. For performance purposes, is a machine with a single dual-core processor basically equivalent to a machine with two processors with one core each? Yes. Roughly. There are lots of subtleties here concerning amount of cache, frequency of process hand-offs between processors, etc. Despite those finer points, for the purposes of sizing up the CPU load value, the total number of cores is what matters, regardless of how many physical processors those cores are spread across.

Which leads us to a two new Rules of Thumb:

* The "number of cores = max load" Rule of Thumb: on a multicore system, your load should not exceed the number of cores available.
* The "cores is cores" Rule of Thumb: How the cores are spread out over CPUs doesn't matter. Two quad-cores == four dual-cores == eight single-cores. It's all eight cores for these purposes.

## BRINGING IT HOME

Let's take a look at the load averages output from uptime:

~ $ uptime  
23:05 up 14 days, 6:08, 7 users, load averages: 0.65 0.42 0.36

This is on a dual-core CPU, so we've got lots of headroom. I won't even think about it until load gets and stays above 1.7 or so.

Now, what about those three numbers? 0.65 is the average over the last minute, 0.42 is the average over the last five minutes, and 0.36 is the average over the last 15 minutes. Which brings us to the question:

**Which average should I be observing? One, five, or 15 minute?**

For the numbers we've talked about (1.00 = fix it now, etc), you should be looking at the five or 15-minute averages. Frankly, if your box spikes above 1.0 on the one-minute average, you're still fine. It's when the 15-minute average goes north of 1.0 and stays there that you need to snap to. (obviously, as we've learned, adjust these numbers to the number of processor cores your system has).

**So # of cores is important to interpreting load averages ... how do I know how many cores my system has?**

cat /proc/cpuinfo to get info on each processor in your system. Note: not available on OSX, Google for alternatives. To get just a count, run it through grep and word count: grep 'model name' /proc/cpuinfo | wc –l

## MEMORY CACHING & BUFFERS

Reading data from a disk is far slower than accessing data from memory. Linux caches blocks from the disk in memory. In fact, Linux uses all free RAM for the buffer cache to make reading data as efficient as possible.

What happens if a program needs more memory than what’s available? The buffer cache will shrink to accommodate the increased memory needs. The buffer cache works like your most efficient coworker: when things aren’t busy, he runs around making things run smoother. When an important task comes up, he drops the less important chores.

## ACTUAL FREE MEMORY

The actual free memory available is:

Actual Free Memory = Free (39 MB) + Buffers (95) + Cached (3590) = 3,724 MB 

That’s 95x more free memory than than we initially thought.

If you are using [Scout](http://scoutapp.com/), Eric Lindvall’s [Memory Profiler plugin](http://scoutapp.com/plugin_urls/15-memory-profiler) already takes the buffer cache into account when determining free memory. This plugin is installed by default on newly monitored servers.