System Tuning Info for Linux Servers

NOTE: Most of the info on this page is about 3 years, and one or two kernel versions out of date.

This page is about optimizing and tuning Linux based systems for server oriented tasks. Most of the info presented here I've used myself, and have found it to be beneficial. I've tried to avoid the well tread ground (hdparm, turning off hostname lookups in apache, etc) as that info is easy to find elsewhere.

Some cases where you might want to apply some of benchmarking, high traffic web sites, or in case of any load spike (say, a web transfered virus is pegging your servers with bogus requests)

File and Disk Tuning

Benchmark performance is often heavily based on disk I/O performace. So getting as much disk I/O as possible is the real key.

Depending on the array, and the disks used, and the controller, you may want to try software raid. It is tough to beat software raid performace on a modern cpu with a fast disk controller.

The easiest way to configure software raid is to do it during the install. If you use the gui installer, there are options in the disk partion screen to create a "md" or multiple-device, linux talk for a software raid partion. You will need to make partions on each of the drives of type "linux raid", and then after creating all these partions, create a new partion, say " /test", and select md as its type. Then you can select all the partions that should be part of it, as well as the raid type. For pure performance, RAID 0 is the way to go.

Note that by default, I belive you are limited to 12 drives in a MD device, so you may be limited to that. If the drives are fast enough, that should be sufficent to get >100 MB/s pretty consistently.

One thing to keep in mind is that the position of a partion on a hardrive does have performance implications. Partions that get stored at the very outer edge of a drive tend to be significantly faster than those on the inside. A good benckmarking trick is to use RAID across several drives, but only use a very small partion on the outside of the disk. This give both consistent performance, and the best performance. On most moden drives, or least drives using ZCAV (Zoned Constant Angular Velocity), this tends to be sectors with the lowest address, aka, the first partions. For a way to see the differences illustrated, see the ZCAV page.

This is just a summary of software RAID configuration. More detailed info can be found elsewhere including the Software-RAID-HOWTO, and the docs and man pages from the **raidtools** package.

File System Tuning

Some of the default kernel paramaters for system performance are geared more towards workstation performance that file server/large disk io type of operations. The most important of these is the "bdflush" value in /proc/sys/vm/bdflush

These values are documented in detail in /usr/src/linux/Documenation/sysctl/vm.txt.

A good set of values for this type of server is:

```
echo 100 5000 640 2560 150 30000 5000 1884 2 > /proc/sys/vm/bdflush
```

(you change these values by just echo'ing the new values to the file. This takes effect immediately. However, it needs to be reinitilized at each kernel boot. The simplest way to do this is to put this command into the end of /etc/rc.d/rc.local)

Also, for pure file server applications like web and samba servers, you probably want to disable the "atime" option on the filesystem. This disabled updating the "atime" value for the file, which indicates that the last time a file was accessed. Since this info isnt 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
```

With these file system options, a good raid setup, and the bdflush values, filesystem performace should be suffiecent.

The disk i/o elevators is another kernel tuneable that can be tweaked for improved disk i/o in some cases.

SCSI Tuning

SCSI tuning is highly dependent on the particular scsi cards and drives in questions. The most effective variable when it comes to SCSI card performace is tagged command queueing.

For the Adaptec aic7xxx seriers cards (2940's, 7890's, *160's, etc) this can be enabled with a module option like:

```
aic7xx=tag_info:{{0,0,0,0,}}
```

This enabled the default tagged command queing on the first device, on the first 4 scsi ids.

```
options aic7xxxaic7xxx=tag_info:{{24.24.24.24.24.24}}
```

in /etc/modules.conf will set the TCQ depth to 24

You probably want to check the driver documentation for your particular scsi modules for more info.

Disk I/O Elevators

On systems that are consistently doing a large amount of disk I/O, tuning the disk I/O elevators may be useful. This is a 2.4 kernel feature that allows some control over latency vs throughput by changing the way disk io elevators operate.

This works by changing how long the I/O scheduler will let a request sit in the queue before it has to be handled. Since the I/O scheduler can collapse some request together, having a lot of items in the queue means more can be cooalesced, which can increase throughput.

Changing the max latency on items in the queue allows you to trade disk i/o latency for throughput, and vice versa.

The tool "/sbin/elvtune" (part of util-linux) allows you to change these max latency values. Lower values means less latency, but also less thoughput. The values can be set for the read and write queues seperately.

To determine what the current settings are, just issue:

```
/sbin/elvtune /dev/hda1
```

substituting the approriate device of course. Default values are 8192 for read, and 16384 for writes.

To set new values of 2000 for read and 4000 for example:

```
/sbin/elvtune -r 2000 -w 4000 /dev/hda1
```

Note that these values are for example purposes only, and are not recomended tuning values. That depends on the situation.

The units of these values are basically "sectors of writes before reads are allowed". The kernel attempts to do all reads, then all writes, etc in an attempt to prevent disk io mode switching, which can be slow. So this allows you to alter how long it waits before switching.

One way to get an idea of the effectiveness of these changes is to monitor the output of `isostat -d -x DEVICE`. The "avgrq-sz" and "avgqu-sz" values (average size of request and average queue length, see man page for iostat) should be affected by these elevator changes. Lowering the latency should cause the "avqrq-sz" to go down, for example.

See the **elvtune** man page for more info. Some info from when this feature was introduced is also at Lwn.net

This info contributed by Arjan van de Ven.

Network Interface Tuning

Most benchmarks benifit heavily from making sure the NIC's in use are well supported, with a well written driver. Examples include eepro100, tulip's, newish 3com cards, and acenic and sysconect gigabit cards.

Making sure the cards are running in full duplex mode is also very often critical to benchmark performace.

Depending on the networking hardware used, some of the cards may not autosense properly and may not run full duplex by default.

Many cards include module options that can be used to force the cards into full duplex mode. Some examples for common cards include

```
alias eth0 eepro100
options eepro100 full_duplex=1
alias eth1 tulip
options tulip full_duplex=1
```

Though full duplex gives the best overall performance, I've seen some circumstances where setting the cards to half duplex will actually increase thoughput, particulary in cases where the data flow is heavily one sided.

If you think your in a situation where that may help, I would suggest trying it and benchmarking it.

TCP tuning

For servers that are serving up huge numbers of concurent sessions, there are some tcp options that should probabaly be enabled. With a large # of clients doing their best to kill the server, its probabaly not uncommon for the server to have 20000 or more open sockets.

In order to optimize TCP performace for this situation, I would suggest tuning the following parameters.

```
echo 1024 65000 > /proc/sys/net/ipv4/ip_local_port_range
```

Allows more local ports to be available. Generally not a issue, but in a benchmarking scenario you often need more ports available. A common example is clients running `ab` or `http_load` or similar software.

In the case of firewalls, or other servers doing NAT or masquerading, you may not be able to use the full port range this way, because of the need for high ports for use in NAT.

Increasing the amount of memory associated with socket buffers can often improve performance. Things like NFS in particular, or apache setups with large buffer configured can benefit from this.

```
echo 262143 > /proc/sys/net/core/rmem_max
echo 262143 > /proc/sys/net/core/rmem_default
```

This will increase the amount of memory available for socket input queues. The "wmem_*" values do the same for output queues.

Note: With 2.4.x kernels, these values are supposed to "autotune" fairly well, and some people suggest just instead changing the values in:

```
/proc/sys/net/ipv4/tcp_rmem
/proc/sys/net/ipv4/tcp_wmem
```

There are three values here, "min default max".

These reduce the amount of work the TCP stack has to do, so is often helpful in this situation.

```
echo 0 > /proc/sys/net/ipv4/tcp_sack
echo 0 > /proc/sys/net/ipv4/tcp_timestamps
```

File Limits and the like

Open tcp sockets, and things like apache are prone to opening a large amount of file descriptors. The default number of available FD is 4096, but this may need to be upped for this scenario.

The theorectial limit is roughly a million file descriptors, though I've never been able to get close to that many open.

I'd suggest doubling the default, and trying the test. If you still run out of file descriptors, double it again.

For example:

```
echo 128000 > /proc/sys/fs/inode-max
echo 64000 > /proc/sys/fs/file-max
```

and as root:

```
ulimit -n 64000
```

Note: On 2.4 kernels, the "inode-max" entry is no longer needed.

You probabaly want to add these to /etc/rc.d/rc.local so they get set on each boot.

There are more than a few ways to make these changes "sticky". In Red Hat Linux, you can you /etc/sysctl.conf and /etc/security/limits.conf to set and save these values.

If you get errors of the variety "Unable to open file descriptor" you definately need to up these values.

You can examine the contents of /proc/sys/fs/file-nr to determine the number of allocated file handles, the number of file handles currently being used, and the max number of file handles.

Process Limits

For heavily used web servers, or machines that spawn off lots and lots of processes, you probabaly want to up the limit of processes for the kernel.

Also, the 2.2 kernel itself has a max process limit. The default values for this are 2560, but a kernel recompile can take this as high as 4000. This is a limitation in the 2.2 kernel, and has been removed from 2.3/2.4.

The values that need to be changed are:

If your running out how many task the kernel can handle by default, you may have to rebuild the kernel after editing:

```
/usr/src/linux/include/linux/tasks.h
```

and change:

```
#define NR_TASKS        2560    /* On x86 Max 4092, or 4090 w/APM
configured.*/
```

to

```
#define NR_TASKS        4000    /* On x86 Max 4092, or 4090 w/APM
configured.*/
```

and:

```
#define MAX_TASKS_PER_USER (NR_TASKS/2)
```

to

```
#define MAX_TASKS_PER_USER (NR_TASKS)
```

Then recompile the kernel.

also run:

```
ulimit -u 4000
```

Note: This process limit is gone in the 2.4 kernel series.

Threads

Limitations on threads are tightly tied to both file descriptor limits, and process limits.

Under Linux, threads are counted as processes, so any limits to the number of processes also applies to threads. In a heavily threaded app like a threaded TCP engine, or a java server, you can quickly run out of threads.

For starters, you want to get an idea how many threads you can open. The `thread-limit` util mentioned in the Tuning Utilities section is probabaly as good as any.

The first step to increasing the possible number of threads is to make sure you have boosted any process limits as mentioned before.

There are few things that can limit the number of threads, including process limits, memory limits, mutex/semaphore/shm/ipc limits, and compiled in thread limits. For most cases, the process limit is the first one to run into, then the compiled in thread limits, then the memory limits.

To increase the limits, you have to recompile glibc. Oh fun!. And the patch is essentially two lines!. Woohoo!

```
--- ./linuxthreads/sysdeps/unix/sysv/linux/bits/local_lim.h.akl Mon Sep  4
19:37:42 2000
+++ ./linuxthreads/sysdeps/unix/sysv/linux/bits/local_lim.h     Mon Sep  4
19:37:56 2000
@@ -64,7 +64,7 @@
 /* The number of threads per process.  */
 #define _POSIX_THREAD_THREADS_MAX      64
 /* This is the value this implementation supports.  */
-#define PTHREAD_THREADS_MAX    1024
+#define PTHREAD_THREADS_MAX  8192

 /* Maximum amount by which a process can descrease its asynchronous I/O
    priority level.  */
--- ./linuxthreads/internals.h.akl      Mon Sep  4 19:36:58 2000
+++ ./linuxthreads/internals.h  Mon Sep  4 19:37:23 2000
@@ -330,7 +330,7 @@
    THREAD_SELF implementation is used, this must be a power of two and
    a multiple of PAGE_SIZE.  */
 #ifndef STACK_SIZE
-#define STACK_SIZE  (2 * 1024 * 1024)
+#define STACK_SIZE  (64 * PAGE_SIZE)
 #endif

 /* The initial size of the thread stack.  Must be a multiple of PAGE_SIZE.
  * */
```

Now just patch glibc, rebuild, and install it. ;-> If you have a package based system, I seriously suggest making a new package and using it.

Some info how to do this are Jlinux.org. They describe how to increase the number of threads so Java apps can use them.

NFS

A good resource on NFS tuning on linux is the linux NFS HOW-TO. Most of this info is gleaned from there.

But the basic tuning steps include:

Try using NFSv3 if you are currently using NFSv2. There can be very significant performance increases with this change.

Increasing the read write block size. This is done with the **rsize** and **wsize** mount options. They need to the mount options used by the NFS clients. Values of 4096 and 8192 reportedly increase performance alot. But see the notes in the HOWTO about experimenting and measuring the performance implications. The limits on these are 8192 for NFSv2 and 32768 for NFSv3

Another approach is to increase the number of nfsd threads running. This is normally controlled by the nfsd init script. On Red Hat Linux machines, the value "RPCNFSDCOUNT" in the nfs init script controls this value. The best way to determine if you need this is to experiment. The HOWTO mentions a way to determin thread usage, but that doesnt seem supported in all kernels.

Another good tool for getting some handle on NFS server performance is `nfsstat`. This util reads the info in /proc/net/rpc/nfs[d] and displays it in a somewhat readable format. Some info intended for tuning Solaris, but useful for it's description of the nfsstat format

See also the tcp tuning info

Apache config

Make sure you starting a ton of initial daemons if you want good benchmark scores.

Something like:

```
#######
MinSpareServers 20
MaxSpareServers 80
StartServers 32

# this can be higher if apache is recompiled
MaxClients 256

MaxRequestsPerChild 10000
```

Note: Starting a massive amount of httpd processes is really a benchmark hack. In most real world cases, setting a high number for max servers, and a sane spare server setting will be more than adequate. It's just the instant on load that benchmarks typically generate that the StartServers helps with.

The MaxRequestPerChild should be bumped up if you are sure that your httpd processes do not leak memory. Setting this value to 0 will cause the processes to never reach a limit.

One of the best resources on tuning these values, especially for app servers, is the mod_perl performance tuning

documentation.

Bumping the number of available httpd processes

Apache sets a maximum number of possible processes at compile time. It is set to 256 by default, but in this kind of scenario, can often be exceeded.

To change this, you will need to chage the hardcoded limit in the apache source code, and recompile it. An example of the change is below:

```
--- apache_1.3.6/src/include/httpd.h.prezab     Fri Aug  6 20:11:14 1999
+++ apache_1.3.6/src/include/httpd.h    Fri Aug  6 20:12:50 1999
@@ -306,7 +306,7 @@
   * the overhead.
   */
 #ifndef HARD_SERVER_LIMIT
-#define HARD_SERVER_LIMIT 256
+#define HARD_SERVER_LIMIT 4000
 #endif

 /*
```

To make useage of this many apache's however, you will also need to boost the number of processes support, at least for 2.2 kernels. See the section on kernel process limits for info on increasing this.

The biggest scalability problem with apache, 1.3.x versions at least, is it's model of using one process per connection. In cases where there large amounts of concurent connections, this can require a large amount resources. These resources can include RAM, schedular slots, ability to grab locks, database connections, file descriptors, and others.

In cases where each connection takes a long time to complete, this is only compunded. Connections can be slow to complete because of large amounts of cpu or i/o usage in dynamic apps, large files being transfered, or just talking to clients on slow links.

There are several strategies to mitigate this. The basic idea being to free up heavyweight apache processes from having to handle slow to complete connections.

Static Content Servers

If the servers are serving lots of static files (images, videos, pdf's, etc), a common approach is to serve these files off a dedicated server. This could be a very light apache setup, or any many cases, something like thttpd, boa, khttpd, or TUX. In some cases it is possible to run the static server on the same server, addressed via a different hostname.

For purely static content, some of the other smaller more lightweight web servers can offer very good performance. They arent nearly as powerful or as flexible as apache, but for very specific performance crucial tasks, they can be a big win.

Boa: http://www.boa.org/
thttpd: http://www.acme.com/software/thttpd/
mathopd: http://mathop.diva.nl

If you need even more ExtremeWebServerPerformance, you probabaly want to take a look at TUX, written by Ingo Molnar. This is the current world record holder for SpecWeb99. It probabaly owns the right to be

called the worlds fastest web server.

Proxy Usage

For servers that are serving dynamic content, or ssl content, a better approach is to employ a reverse-proxy. Typically, this would done with either apache's mod_proxy, or Squid. There can be several advantages from this type of configuration, including content caching, load balancing, and the prospect of moving slow connections to lighter weight servers.

The easiest approache is probabaly to use mod_proxy and the "ProxyPass" directive to pass content to another server. mod_proxy supports a degree of caching that can offer a significant performance boost. But another advantage is that since the proxy server and the web server are likely to have a very fast interconnect, the web server can quickly serve up large content, freeing up a apache process, why the proxy slowly feeds out the content to clients. This can be further enhanced by increasing the amount of socket buffer memory thats for the kernel. See the section on tcp tuning for info on this.

proxy links

- Info on using mod_proxy in conjuction with mod_perl
- webtechniques article on using mod_proxy
- mod_proxy home page
- Squid
- Using mod_proxy with Zope

ListenBacklog

One of the most frustrating thing for a user of a website, is to get "connection refused" error messages. With apache, the common cause of this is for the number of concurent connections to exceed the number of available httpd processes that are available to handle connections.

The apache ListenBacklog paramater lets you specify what backlog paramater is set to listen(). By default on linux, this can be as high as 128.

Increasing this allows a limited number of httpd's to handle a burst of attempted connections.

There are some experimental patches from SGI that accelerate apache. More info at:

http://oss.sgi.com/projects/apache/

I havent really had a chance to test the SGI patches yet, but I've been told they are pretty effective.

Samba Tuning

Depending on the type of tests, there are a number of tweaks you can do to samba to improve its performace over the default. The default is best for general purpose file sharing, but for extreme uses, there are a couple of tweaks.

The first one is to rebuild it with mmap support. In cases where you are serving up a large amount of small files, this seems to be particularly useful. You just need to add a "--with-mmap" to the configure line.

You also want to make sure the following options are enabled in the /etc/smb.conf file:

```
read raw = no
read prediction = true
level2 oplocks = true
```

One of the better resources for tuning samba is the "Using Samba" book from O'reily. The chapter on performance tuning is available online.

Openldap tuning

The most important tuning aspect for OpenLDAP is deciding what attributes you want to build indexes on.

I use the values:

```
cachesize 10000
dbcachesize 100000
sizelimit 10000
loglevel 0
dbcacheNoWsync

index cn,uid
index uidnumber
index gid
index gidnumber
index mail
```

If you add the following parameters to /etc/openldap/slapd.conf before entering the info into the database, they will all get indexed and performance will increase.

SysV shm

Some applications, databases in particular, sometimes need large amounts of SHM segments and semaphores. The default limit for the number of shm segments is 128 for 2.2.

This limit is set in a couple of places in the kernel, and requires a modification of the kernel source and a recompile to increase them.

A sample diff to bump them up:

```
--- linux/include/linux/sem.h.save      Wed Apr 12 20:28:37 2000
+++ linux/include/linux/sem.h   Wed Apr 12 20:29:03 2000
@@ -60,7 +60,7 @@
        int semaem;
 };

-#define SEMMNI  128             /* ?  max # of semaphore identifiers */
+#define SEMMNI  512             /* ?  max # of semaphore identifiers */
 #define SEMMSL  250             /* <= 512 max num of semaphores per id */
 #define SEMMNS  (SEMMNI*SEMMSL) /* ? max # of semaphores in system */
 #define SEMOPM  32             /* ~ 100 max num of ops per semop call */
--- linux/include/asm-i386/shmparam.h.save      Wed Apr 12 20:18:34 2000
+++ linux/include/asm-i386/shmparam.h   Wed Apr 12 20:28:11 2000
@@ -21,7 +21,7 @@
  * Keep _SHM_ID_BITS as low as possible since SHMMNI depends on it and
  * there is a static array of size SHMMNI.
  */
-#define _SHM_ID_BITS   7
+#define _SHM_ID_BITS   10
 #define SHM_ID_MASK    ((1<<_SHM_ID_BITS)-1)

 #define SHM_IDX_SHIFT  (_SHM_ID_BITS)
```

Theoretically, the _SHM_ID_BITS can go as high as 11. The rule is that _SHM_ID_BITS + _SHM_IDX_BITS must be <= 24 on x86.

In addition to the number of shared memory segments, you can control the maximum amount of memory allocated to shm at run time via the /proc interface. /proc/sys/kernel/shmmax indicates the current. Echo a new value to it to increase it.

```
echo "67108864" > /proc/sys/kernel/shmmax
```

To double the default value.

A good resource on this is Tunings The Linux Kernel's Memory.

The best way to see what the current values are, is to issue the command:

```
ipcs -l
```

Ptys and ttys

The number of ptys and ttys on a box can sometimes be a limiting factor for things like login servers and database servers.

On Red Hat Linux 7.x, the default limit on ptys is set to 2048 for i686 and athlon kernels. Standard i386 and similar kernels default to 256 ptys.

The config directive CONFIG_UNIX98_PTY_COUNT defaults to 256, but can be set as high as 2048. For 2048 ptys to be supported, the value of UNIX98_PTY_MAJOR_COUNT needs to be set to 8 in include/linux/major.h

With the current device number scheme and allocations, the maximum number of ptys is 2048.

Benchmarks

Lies, damn lies, and statistics.

But aside from that, a good set of benchmarking utilities are often very helpful in doing system tuning work. It is impossible to duplicate "real world" situations, but that isnt really the goal of a good benchmark. A good benchmark typically tries to measure the performance of one particular thing very accurately. If you understand what the benchmarks are doing, they can be very useful tools.

Some of the common and useful benchmarks include:

Bonnie

Bonnie has been around forever, and the numbers it produces are meaningful to many people. If nothing else, it's good tool for producing info to share with others. This is a pretty common utility for testing driver performance. It's only drawback is it sometimes requires the use of huge datasets on large memory machines to get useful results, but I suppose that goes with the territory.

Check Doug Ledford's list of benchmarks for more info on Bonnie. There is also a somwhat newer version of Bonnie called Bonnie++ that fixes a few bugs, and includes a couple of extra tests.

Dbench

My personal favorite disk io benchmarking utility is `dbench`. It is designed to simulate the disk io load of a system when running the NetBench benchmark suite. It seems to do an excellent job at making all the drive lights blink like mad. Always a good sign.

Dbench is available at The Samba ftp site and mirrors

http_load

A nice simple http benchmarking app, that does integrity checking, parallel requests, and simple statistics. Generates load based off a test file of urls to hit, so it is flexible.

http_load is available from ACME Labs

dkftpbench

A (the?) ftp benchmarking utility. Designed to simulate real world ftp usage (large number of clients, throttles connections to modem speeds, etc). Handy. Also includes the useful dklimits utility .

dkftpbench is available from Dan kegel's page

tiobench

A multithread disk io benchmarking utility. Seems to do an a good job at pounding on the disks. Comes with some useful scripts for generating reports and graphs.

The tiobench site.

dt

dt does a lot. disk io, process creation, async io, etc.

dt is available at The dt page

ttcp

A tcp/udp benchmarking app. Useful for getting an idea of max network bandwidth of a device. Tends to be more accurate than trying to guestimate with ftp or other protocols.

netperf

Netperf is a benchmark that can be used to measure the performance of many different types of networking. It provides tests for both unidirecitonal throughput, and end-to-end latency. The environments currently measureable by netperf include: TCP and UDP via BSD Sockets, DLPI, Unix Domain Sockets, Fore ATM API, HiPPI.

Info: http://www.netperf.org/netperf/NetperfPage.html
Download: ftp://ftp.sgi.com/sgi/src/netperf/

Info provided by Bill Hilf.

httperf

httperf is a popular web server benchmark tool for measuring web server performance. It provides a flexible facility for generating various HTTP workloads and for measuring server performance. The focus of httperf is not on implementing one particular benchmark but on providing a robust, high-

performance tool that facilitates the construction of both micro- and macro-level benchmarks. The three distinguishing characteristics of httperf are its robustness, which includes the ability to generate and sustain server overload, support for the HTTP/1.1 protocol, and its extensibility to new workload generators and performance measurements.

Info: http://www.hpl.hp.com/personal/David_Mosberger/httperf.html
Download: ftp://ftp.hpl.hp.com/pub/httperf/

Info provided by Bill Hilf.

Autobench

Autobench is a simple Perl script for automating the process of benchmarking a web server (or for conducting a comparative test of two different web servers). The script is a wrapper around httperf. Autobench runs httperf a number of times against each host, increasing the number of requested connections per second on each iteration, and extracts the significant data from the httperf output, delivering a CSV or TSV format file which can be imported directly into a spreadsheet for analysis/graphing.

Info: http://www.xenoclast.org/autobench/
Download: http://www.xenoclast.org/autobench/downloads

Info provided by Bill Hilf.

General benchmark Sites

Doug Ledford's page

ResierFS benchmark page

System Monitoring

Standard, and not so standard system monitoring tools that can be useful when trying to tune a system.

vmstat

This util is part of the procps package, and can provide lots of useful info when diagnosing performance problems.

Heres a sample vmstat output on a lightly used desktop:

```
   procs                  memory    swap          io     system  cpu
 r  b  w   swpd   free   buff  cache  si  so    bi    bo   in    cs  us  sy id
 1  0  0   5416   2200   1856  34612   0   1     2     1  140   194   2   1 97
```

And heres some sample output on a heavily used server:

```
   procs                  memory    swap          io     system  cpu
 r  b  w   swpd   free   buff  cache  si  so    bi    bo   in    cs  us  sy id
16  0  0   2360 264400  96672   9400   0   0     0     1   53    24   3   1 96
24  0  0   2360 257284  96672   9400   0   0     0     6 3063 17713  64  36 0
15  0  0   2360 250024  96672   9400   0   0     0     3 3039 16811  66  34 0
```

The interesting numbers here are the first one, this is the number of the process that are on the run queue. This value shows how many process are ready to be executed, but can not be ran at the

moment because other process need to finish. For lightly loaded systems, this is almost never above 1-3, and numbers consistently higher than 10 indicate the machine is getting pounded.

Other interseting values include the "system" numbers for in and cs. The in value is the number of interupts per second a system is getting. A system doing a lot of network or disk I/o will have high values here, as interupts are generated everytime something is read or written to the disk or network.

The cs value is the number of context switches per second. A context switch is when the kernel has to take off of the executable code for a program out of memory, and switch in another. It's actually _way_ more complicated than that, but thats the basic idea. Lots of context swithes are bad, since it takes some fairly large number of cycles to performa a context swithch, so if you are doing lots of them, you are spending all your time chaining jobs and not actually doing any work. I think we can all understand that concept.

netstat

Since this document is primarily concerned with network servers, the `netstat` command can often be very useful. It can show status of all incoming and outgoing sockets, which can give very handy info about the status of a network server.

One of the more useful options is:

```
netstat -pa
```

The `-p` options tells it to try to determine what program has the socket open, which is often very useful info. For example, someone nmap's their system and wants to know what is using port 666 for example. Running netstat -pa will show you its satand running on that tcp port.

One of the most twisted, but useful invocations is:

```
netstat -a -n|grep -E "^(tcp)"| cut -c 68-|sort|uniq -c|sort -n
```

This will show you a sorted list of how many sockets are in each connection state. For example:

```
      9  LISTEN
     21  ESTABLISHED
```

ps

Okay, so everyone knows about ps. But I'll just highlight one of my favorite options:

```
ps -eo pid,%cpu,vsz,args,wchan
```

Shows every process, their pid, % of cpu, memory size, name, and what syscall they are currently executing. Nifty.

Utilities

Some simple utilities that come in handy when doing performance tuning.

dklimits

a simple util to check the acutally number of file descriptors available, ephemeral ports available, and poll()-able sockets. Handy. Be warned that it can take a while to run if there are a large number of fd's

available, as it will try to open that many files, and then unlinkt them.

This is part of the dkftpbench package.

fd-limit

a tiny util for determining the number of file descriptors available.

fd-limit.c

thread-limit

A util for determining the number of pthreads a system can use. This and fd-count are both from the system tuning page for Volano chat, a multithread java based chat server.

thread-limit.c

System Tuning Links

http://www.kegel.com
Check out the "c10k problem" page in particular, but the entire site has _lots_ of useful tuning info.

http://linuxperf.nl.linux.org/

Site organized by Rik Van Riel and a few other folks. Probabaly the best linux specific system tuning page.

http://www.citi.umich.edu/projects/citi-netscape/

Linux Scalibity Project at Umich.

NFS Performance Tunging

Info on tuning linux kernel NFS in particular, and linux network and disk io in general

http://home.att.net/~jageorge/performance.html

Linux Performace Checklist. Some useful content.

http://www.linux.com/enhance/tuneup/

Miscelaneous performace tuning tips at linux.com

http://www.psc.edu/networking/perf_tune.html#Linux

Summary of tcp tuning info

Music

Careful analysis and benchmarking has shown that server will respond positively to being played the approriate music. For the common case, this can be about anything, but for high performane servers, a more careful choice needs to be made.

The industry standard for pumping up a server has always been "Crazy Train", By Ozzy Ozbourne. While this has

been proven over and over to offer increased performance, in some circumstances I recomdend alternatives.

A classic case is the co-located server. Nothing like packing up your pride and joy and shipping it to strange far off locations like Sunnyvale and Herndon, VA. Its enough to make a server homesick, so I like to suggest choosing a piece of music that will remind them of home and tide them over till the bigger servers stop picking on them. For servers from North Carolina, I like to play the entirety of "feet in mud again" by Geezer Lake. Nothing like some good old NC style avant-metal-alterna-prog.

Comentary, controverys,chatter. chit-chat. Chat and irc servers have their own unique set of problems. I find the polyrythmic and incessant restatement of purpose of Elephant Talk by King Crimson a good way to bend those servers back into shape.

btw, Xach says "Crazy Train" has the best guitar solo ever.

Thanks

Folks that have sent me new info, corrected info, or just sat still long enough for me to ask them lots of questions.

- Zach Brown
- Arjan van de Ven
- Xach Beane
- Michael K. Johnson
- James Manning

TODO

- add info about mod_proxy, caching, listenBacklog, etc
- Add info for oracle tuning
- any other useful server specific tuning info I stumble across
- add info about kernel mem limits, PAE, bigmeme, LFS and other kernel related stuff likely to be useful

Changes

Nov 19 2001
: s/conf.modules/modules.conf info on httpperf/autobench/netperf from Bill Hilf.

Oct 16 2001
: Added links to the excellent mod_perl tuning guide, and the online chapter for tuning samba. Added some info about the use of MaxRequestsPerChild, mod_proxy, and listenBacklog to the apache section

---

alikins@redhat.com