

Performance Tuning and Monitoring

Part 4



Tune memory requirements, learn about changing kernel behaviour in relation to memory allocation, fiddle with the memory overcommit feature, and enhance performance by tuning the ARP cache.

Last month we learned how to tune the performance of our hard disks and RAID arrays. Performance tuning is always incomplete without tuning the memory. Therefore, in this article we will learn how to tune the memory to enhance the performance of the system using TLB, strace, demand paging, memory overcommit, and by tuning the ARP cache.

Memory addressing and performance

For performance reasons, the memory on a computer is organised into fixed-sized blocks, or chunks, known as pages. The size of the pages varies from processor to processor. On a normal x86 system, the default page size is 4 KB (4,096 bytes).

The RAM on the computer is divided into page-frames. One page frame of RAM can hold one page of data. Whenever a process tries to access a particular memory location it must be directed to the appropriate page-frame in the memory (RAM). In case that information is not available in the page-

frame (RAM), then the kernel must find that information and load it into the page-frame.

Each process has its own page table. Each page table entry (PTE) contains information about one page-frame that has been assigned to the process.

The Linux kernel also reserves some memory at boot time for internal functions. You can use the `dmesg | grep Memory` command to view the amount of reserved memory.

```
[root@legacy ~]# dmesg | grep Memory
[ 0.000000] Memory: 3351360k/3406668k available
(3264k kernel code, 54032k reserved, 2046k data, 404k
init, 2501460k highmem)
```

You can also verify your page size by running the `getconf -a | grep SIZE` command. Shown below is the output from my machine:

```
[root@legacy ~]# getconf -a | grep SIZE
PAGESIZE                4096
PAGE_SIZE                4096
<<<<output truncated>>>>
```

Viewing the virtual memory of a process

You can view the virtual memory that is being taken up by a process using a number of tools. Whenever a process runs, the set of pages required by that process is known as its working set. Obviously, the working set for a process changes continuously over its lifetime as its memory requirements also change.

The kernel constantly trims pages that have not been recently used from the working set of a process. If the pages contain modified data, then those must be written to the disk, else the kernel will allocate the pages to some other process that needs them.

You can use tools like *pmmap*, *memusage*, KDE or GNOME System Monitor to view the memory usage of a process. Figure 1 shows the truncated output from the *memusage* command for the Firefox process on my machine. Note that you need to install *glibc-utils* to use *memusage*.

You can also use the simple *time* command to view statistics like CPU time, system time, etc, for a process. The following snippet shows the output for *time* when used to check the memory status for the *man date* command:

```
[root@legacy ~]# /usr/bin/time -v man date | grep -i size
Command being timed: "man date"
User time (seconds): 0.05
System time (seconds): 0.04
Percent of CPU this job got: 87%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:00.11
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 0
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 1
Minor (reclaiming a frame) page faults: 10063
Voluntary context switches: 73
Involuntary context switches: 53
Swaps: 0
File system inputs: 256
File system outputs: 16
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0
```

Tuning TLB performance

When a process that's running needs to access a memory location, it uses a linear address. On x86 machines this linear address is a 32-bit address that can grow up to 4GB of memory, and it does not refer to the actual location. To access an actual physical memory location, linear addresses must be translated into physical addresses by

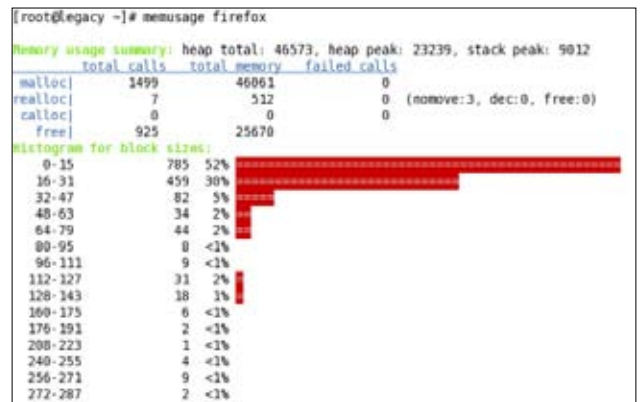


Figure 1: memusage output for Firefox

hardware circuits in the CPU.

Now, translating a linear address into a physical address takes time. So normally all CPUs have a small cache known as Translation Lookaside Buffer (TLB). TLB maps the physical address associated with the most recently-accessed virtual address. TLB is limited in size.

It is actually a table in the processor containing a cross-reference between virtual and real addresses of recently-referenced pages.

Tuning TLB can be very useful for applications requiring a big contiguous memory space. It is normally done at the boot time. First, check the current *hugepage* or TLB status using *cat /proc/meminfo* or *x86info* or *dmesg*. The following snippet shows the status of hugepages on my machine:

```
[root@legacy ~]# cat /proc/meminfo | grep Huge
HugePages_Total: 0
HugePages_Free: 0
HugePages_Rsvd: 0
HugePages_Surp: 0
Hugepagesize: 4096 kB
```

As you can see, I'm not yet using hugepages. So, why don't we configure the system to use the TLB or hugepages? Let's say I want to use 10 hugepages of contiguous memory.

Step 1 Configure */etc/sysctl.conf* for the number of pages you intend to use. Append *vm.nr_hugepages=10* in */etc/sysctl.conf* file and run *sysctl -p*.

Step 2 Edit */boot/grub/grub.conf* to use the hugepages—because this memory is allocated at the boot time—as shown below:

```
title Red Hat Enterprise Linux Server (2.6.18-92.el5)
root (hd0,0)
kernel /boot/vmlinuz- 2.6.18-92.el5 ro root=LABEL=/ rhgb quiet
hugepages=10
initrd /boot/initrd- 2.6.18-92.el5.img
```

```
[root@legacy ~]# strace -c ls
anaconda-ks.cfg      cachegrind.out.6941  install.log.syslog
bonnie++-1.03e.tar   Desktop             iptables-squid-email
cache1.c             Documents           Music
cache2.c             Download            myrules
cachegrind.out.6723  fping-2.4b2-9.fc11.i586.rpm  nagios-3.0.6
cachegrind.out.6831  install.log         nagios-3.0.6.tar
% time      seconds  usecs/call   calls    errors syscall
-----
89.17      0.000280    280         1        0      execve
10.83      0.000034    1          26        0      mmap2
0.00       0.000000    0          12        0      read
0.00       0.000000    0           6        0      write
0.00       0.000000    0          13        0      open
0.00       0.000000    0          15        0      close
0.00       0.000000    0           1        0      1 access
0.00       0.000000    0           3        0      brk
0.00       0.000000    0           2        0      ioctl
0.00       0.000000    0           4        0      munmap
0.00       0.000000    0           1        0      uname
0.00       0.000000    0           7        0      mprotect
0.00       0.000000    0           2        0      rt_sigaction
0.00       0.000000    0           1        0      rt_sigprocmask
0.00       0.000000    0           1        0      getrlimit
0.00       0.000000    0          13        0      fstat64
0.00       0.000000    0           2        0      getdents64
0.00       0.000000    0           1        0      fcntl64
0.00       0.000000    0           2        0      1 futex
0.00       0.000000    0           1        0      set_thread_area
0.00       0.000000    0           1        0      set_tid_address
0.00       0.000000    0           1        0      statfs64
0.00       0.000000    0           1        0      set_robust_list
-----
100.00     0.000314    117         2 total
```

Figure 2: Tracing ls with strace

Step 3 Configure hugetlbfs if needed by an application. Mount it on any location and make it permanent. Append the following line to your `/etc/fstab` file:

```
none /hugepage hugetlbfs defaults 0 0
```

Run `mount -a` to activate the changes.

Now, if we cat the `/proc/meminfo` file, we can see the hugepages are activated in the system and can be used by any application that needs them:

```
[root@legacy ~]# cat /proc/meminfo | grep Huge
```

```
HugePages_Total: 10
HugePages_Free: 10
HugePages_Rsvd: 0
Hugepagesize: 4096 kB
```

Configuring the TLB increases the chances of the TLB cache being hit more frequently, and significantly reduces page table entry (PTE) visit count.

Viewing system calls

Many a time, after devoting a good amount of time downloading and compiling a program, it simply refuses to run. We are left with the last resort of tracing the output of the program that's misbehaving. Tracing the output of a program throws up a lot of data that is not usually available when the program is run normally.

`strace` comes in handy for tracing the system calls of a program. When it is run in conjunction with a program, it outputs all the calls that the program makes to the kernel. A program might often fail to initiate because of insufficient memory. Tracing the output of the program

```
[root@legacy ~]# strace -c -o lsoutput.txt ls
anaconda-ks.cfg      cachegrind.out.6941  install.log.syslog
bonnie++-1.03e.tar   Desktop             iptables-squid-email
cache1.c             Documents           Music
cache2.c             Download            myrules
cachegrind.out.6723  fping-2.4b2-9.fc11.i586.rpm  nagios-3.0.6
cachegrind.out.6831  install.log         nagios-3.0.6
[root@legacy ~]# cat lsoutput.txt
% time      seconds  usecs/call   calls    errors syscall
-----
94.26      0.000476    79         6        0      write
5.74       0.000029    2          13        0      fstat64
0.00       0.000000    0          12        0      read
0.00       0.000000    0          13        0      open
0.00       0.000000    0          15        0      close
0.00       0.000000    0           1        0      execve
0.00       0.000000    0           1        0      1 access
0.00       0.000000    0           3        0      brk
0.00       0.000000    0           2        0      ioctl
0.00       0.000000    0           4        0      munmap
0.00       0.000000    0           1        0      uname
0.00       0.000000    0           7        0      mprotect
0.00       0.000000    0           2        0      rt_sigaction
0.00       0.000000    0           1        0      rt_sigprocmask
0.00       0.000000    0           1        0      getrlimit
0.00       0.000000    0           1        0      getrlimit
0.00       0.000000    0          26        0      mmap2
0.00       0.000000    0           2        0      getdents64
0.00       0.000000    0           1        0      fcntl64
0.00       0.000000    0           2        0      1 futex
0.00       0.000000    0           1        0      set_thread_area
0.00       0.000000    0           1        0      set_tid_address
0.00       0.000000    0           1        0      statfs64
0.00       0.000000    0           1        0      set_robust_list
-----
100.00     0.000505    117         2 total
```

Figure 3: Redirecting the output of strace to a text file

will let us pinpoint the exact issue.

To understand this better, take a look at Figure 2, which shows the output thrown by `strace` when used along with the `ls` command. As you can see, it displays all the system calls made by `ls`, along with the number of times each one was made. You can also redirect the output of `strace` by using the `-o` option as shown in Figure 3.

By now you must have realised how `strace` can be very handy in investigating locks, identifying problems caused by improper permissions and also identifying IO problems.

Tuning memory caches

In addition to the hardware cache, there are other types of caches in the Linux kernel—implemented in the main memory. The idea behind it is to prevent the kernel from having to read the information on the hard disk as much as possible, thereby reducing latency.

The most important of them all is page cache. It plays a very pivotal role in disk IO.

Demand paging

The kernel delays the memory allocation until a process asks for it for performance reasons. Due to the locality of reference, the processes do not use the whole memory they had asked for at one time. Just think about it -- while executing a large program, it is more likely that only a small part of the executable image is run at a given time. So it will be a total waste of memory to allocate space for the full executable image.

Linux delays the allocation of memory until it is asked for by the process. This process of delaying the memory allocation for a process until the last minute is known as demand paging.

Tuning page allocation

Here's how you can check and modify the current page size:

```
[root@legacy ~]# cat /proc/sys/vm/min_free_kbytes
3031
[root@legacy ~]#
[root@legacy ~]# echo 4000 > /proc/sys/vm/min_free_kbytes
[root@legacy ~]# cat /proc/sys/vm/min_free_kbytes
4000
[root@legacy ~]#
```

To make this change permanent across reboots, append the following as in your `/etc/sysctl.conf` file:

```
vm.min_free_kbytes = 4000
```

...and run `sysctl -p`. Please note that it should only be configured when an application regularly needs to allocate large blocks of memory and then frees up the same memory.

Memory overcommit

Airlines sometimes tend to sell more tickets than the actual number of seats available. This is to be on the safe side in case a number of passengers cancel their tickets at the last moment. Linux manages memory in a very similar fashion. Whenever any process asks for memory allocation, Linux allocates it assuming that a program is asking for more memory than it actually requires.

Memory overcommit can be set by defining the `vm.overcommit_memory` parameter under the `/etc/sysctl.conf` file. It takes three values:

- 0 – means heuristic overcommit (i.e., always refuses to over-commit)
- 1 – means always overcommit (whether available or not)
- 2 – commits all of swap plus a percentage of RAM, controlled by `vm.overcommit_ratio` (default is 50).

We can verify the current values of the overcommit parameters as follows:

```
[root@legacy ~]# sysctl -a | grep overcommit
vm.overcommit_ratio = 50
vm.overcommit_memory = 0
```

In order to change the parameter from 0 to 1, we can simply modify the parameter in the `/etc/sysctl.conf` file as follows:

```
vm.overcommit_memory = 1
```

If the `vm.overcommit_memory` is set to 2, then the system will allocate memory equal to the amount of swap plus a percentage (default 50 per cent) of

physical memory. We can control the `vm.overcommit_ratio` also by appending a corresponding entry in `/etc/sysctl.conf` as follows:

```
vm.overcommit_ratio = 40
```

As always, do run `sysctl -p` to commit your changes.

ARP cache


Address Resolution Protocol (ARP) resolves or maps IP addresses to hardware addresses (MAC). In Linux, by default, the ARP cache size is limited to 512 entries as the soft limit and 1,024 entries as hard limit. The soft limit automatically becomes the hard limit after five seconds.

To see the current soft and hard limits, you can use `sysctl -a | grep -i thresh`, as shown:

```
[root@legacy ~]# sysctl -a | grep -i thresh
kernel.sched_shares_thresh = 4
kernel.random.read_wakeup_threshold = 64
kernel.random.write_wakeup_threshold = 128
kernel.softlockup_thresh = 60
net.netfilter.nf_conntrack_frag6_low_thresh = 196608
net.netfilter.nf_conntrack_frag6_high_thresh = 262144
net.ipv4.route.gc_thresh = 32768
net.ipv4.neigh.default.gc_thresh1 = 128
net.ipv4.neigh.default.gc_thresh2 = 512
net.ipv4.neigh.default.gc_thresh3 = 1024
<<<<output truncated>>>>
```

We can easily configure these values under the `/etc/sysctl.conf` file as explained earlier. One very interesting attribute here is the *garbage collection frequency* (time in seconds). When the hard limit is exceeded, the kernel garbage collector rescans the cache after the time specified and purges entries to take it under the limit.

```
[root@legacy ~]# sysctl -a | grep gc_interval
net.ipv4.route.gc_interval = 60
net.ipv4.neigh.default.gc_interval = 30
net.ipv6.neigh.default.gc_interval = 30
net.ipv6.route.gc_interval = 30
```

I'll end here, for this month. In the next issue I will conclude this series by explaining memory reclamation and network performance boosting. 

By: Alok Srivastava

The author is the founder of Network NUTS and holds MCP, MCSE, MCDBA, MCT, CCNA, CCNP, RHCE and RHCSS certifications. Under his leadership, Network NUTS has been a winner of the "Best Red Hat Training Partner in North India" for the last three years in a row. He has also been a trainer for the Indian Air Force, NIC, LIC, IFFCO, Wartsila India Ltd, the Government of Rajasthan, Bajaj Allianz, etc. You can reach him at alok@networknuts.net.