

Performance Tuning and Monitoring

Part 5

Reclaim memory, tune swap and enhance network performance.

Last month, we had worked on tuning our memory requirements—changing the kernel's behaviour for memory allocation, exploring the memory overcommit feature, and enhancing performance by tuning the ARP cache. To conclude this series on 'Performance Tuning and Monitoring', we will discuss memory reclamation, tuning swap and enhancing network performance.

States of virtual memory page

The operating system manages the virtual memory pages by reading a label known as a state. These labels (or states) help the virtual memory to take the required action

on the page. So let's first understand the different states of a virtual memory page:

- **FREE** – the page is free or available for immediate allocation.
 - **INACTIVE CLEAN** – the page is not active and the contents are being written to the disk. There's nothing to be done on the page. In other words, the page is available for allocation.
 - **INACTIVE DIRTY** – 'dirty' in this context means that the contents have been modified after reading from disk but not written back on the disk, and the page is currently not in use.
 - **ACTIVE** – the page is currently in use by some process.
- Because memory is a limited resource,

the kernel cannot store the dirty pages in RAM forever. As the kernel runs short of memory, the pages that have been dirtied, but are no longer used by a process, must be flushed back to the disk. This is handled by the *pdflush* kernel thread.

The default is a minimum of two threads. Use the following command to view the current status of *pdflush*:

```
[root@legacy ~]# sysctl -a | grep pdflush
```

```
vm.nr_pdflush_threads = 2
```

```
[root@legacy ~]
```

The idea behind *pdflush* is to avoid a sudden surprising load of heavy disk I/O. The goal is to write dirtied pages back to the disk at constant intervals. The default behaviour of *pdflush* is to write dirty pages to the disk as a kernel background task. The logic is simple: if the ratio of the number of pages that are dirty, to the total number of pages of the RAM available, reaches a certain percentage, *pdflush* writes the dirty pages synchronously.

These are some of the other parameters that you can use to tune the behaviour of *pdflush*:

1. Tuning as per the length/size of memory
 - *vm.dirty_background_ratio* – sets the percentage (of total memory) of dirty pages at which *pdflush* starts writing to disk.
 - *vm.dirty_ratio* – sets the percentage (of total memory) of dirty pages at which the process itself starts writing dirty data to disk.
2. Tuning according to time
 - *vm.dirty_writeback_centisecs* – sets the interval between *pdflush* wake-ups (in 1/100th of a second). Set zero to disable *pdflush*.
 - *vm.dirty_expire_centisecs* – sets the time when the data is old enough (in 1/100th of a second) to be written to disk by *pdflush*

Here is a sample default configuration for *pdflush* values on my system:

```
[root@legacy ~]# sysctl -a | grep vm.dirty
```

```
vm.dirty_background_ratio = 10
```

```
vm.dirty_background_bytes = 0
```

```
vm.dirty_ratio = 20
```

```
vm.dirty_bytes = 0
```

```
vm.dirty_writeback_centisecs = 500
```

```
vm.dirty_expire_centisecs = 3000
```

```
[root@legacy ~]#
```

As you can see, *pdflush* will start writing the dirty pages to the disk as soon as 10 per cent of the total memory gets dirty. On the other hand, the process itself will start writing the data to disk as soon as 20 per cent of the total memory is dirty.

You can always reset these values by defining the new values in */etc/sysctl.conf* and then running *sysctl -p*.

Tuning swap

We all know what swap is and how it is important to us. We also know that swap is very slow compared to the physical memory (RAM). Naturally, when a system heavily relies on swap it basically means we need more RAM.

A common controversial question is how much swap space to dedicate to a system. The answer depends on several factors. However, generally, you can use the following guidelines:

- If the amount of RAM is less than 1 GB, the ideal swap space should be twice the size of RAM.
 - If the amount of RAM is between 1-2 GB, the ideal swap space should be 1.5 times the size of RAM.
 - If the amount of RAM is between 2-8 GB, the ideal swap space should equal to the size of RAM
 - If the amount of RAM is more than 8 GB, the ideal swap space should be 0.75 times the size of RAM
- Let's now see how to improve the swap performance.

Ideally, dedicate a set of high performance disks spread across two or more controllers. If the swap resides on a busy disk, then, to reduce swap latency, it should be located as close to a busy partition (like */var*) as possible to reduce seek time for the drive head.

While using different partitions or hard disks of different speeds for swap, you can assign priorities to each partition so that the kernel will use the higher priority (actually high performance) disk first. You can set the priority using the *pri* option in your */etc/fstab* file. Here is an example of the */etc/fstab* file under such circumstances:

```
/dev/sda6    swap        swap        pri=4        0    0
/dev/sda8    swap        swap        pri=4        0    0
/dev/sdb3    swap        swap        pri=1        0    0
```

The kernel will first use the swap with a higher *pri* value—in our example, */dev/sda6* and */dev/sda8*. It will not use */dev/sdb3* until all of */dev/sda6* and */dev/sda8* are fully used. In addition, the kernel will distribute visit counts in a round-robin style across all the devices with equal priorities.

The basics of network tuning

First, let's look at what happens when a packet is transmitted out of the NIC (network interface card):

1. Writes data to socket file (goes in transmit buffer).
2. Kernel encapsulates data into protocol data units.
3. PDUs are moved to the per-device transmit queue.
4. Driver copies PDU from head of queue to NIC.
5. NIC raises the interrupt when PDU is transmitted.

Now let's see what actually happens when a packet is received on NIC:

1. NIC receives frame and uses DMA to copy into receive buffer.
2. NIC raises CPU interrupt.
3. Kernel handles interrupt and schedules a *softirq*.
4. The *softirq* is handled to move packets up to the IP layer for the routing decision.
5. If it's a local delivery, then the packet is decapsulated and placed into the socket's receive buffer.

The 'txqueuelen' value, as reported by the *ifconfig* command, is the length of the transmit queue on the NIC. Packets are scheduled by a queuing discipline. I ran the *ifconfig* command to find out its value in my system:

```
[root@legacy ~]# ifconfig
eth0    Link encap:Ethernet  HWaddr 00:15:C5:C0:B6:76
        inet addr:172.24.0.252  Bcast:172.24.255.255  Mask:255.255.0.0
        inet6 addr: fe80::215:c5ff:fec0:b676/64  Scope:Link
        UP BROADCAST MULTICAST  MTU:1500  Metric:1

        RX packets:27348 errors:0 dropped:0 overruns:0 frame:0

        TX packets:20180 errors:0 dropped:0 overruns:0 carrier:0

        collisions:0 txqueuelen:1000

        RX bytes:22661585 (21.6 MiB)  TX bytes:3683788 (3.5 MiB)

        Interrupt:17
```

As you can see, it's set to 1000, by default. Let's now issue another interesting command related to the previous 'txqueuelen' value:

```
[root@legacy ~]# tc -s qdisc show dev eth0

qdisc pfifo_fast 0: root bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1

Sent 3601538 bytes 20199 pkt (dropped 0, overlimits 0 requeues 0)

rate 0bit Opps backlog 0b 0p requeues 0
```

The above command basically outputs the 'qdisc' for eth0, and also notifies us whether the connection is dropping packets. If it is, then we need to increase the length of the queue. Here's how we can increase the value to 10001:

```
[root@legacy ~]# ip link set eth0 txqueuelen 10001
```

We verify whether the new value has taken effect by running the *ifconfig* command again:

```
[root@legacy ~]# ifconfig
eth0    Link encap:Ethernet  HWaddr 00:15:C5:C0:B6:76
```

```
inet addr:172.24.0.252  Bcast:172.24.255.255  Mask:255.255.0.0
inet6 addr: fe80::215:c5ff:fec0:b676/64  Scope:Link
UP BROADCAST MULTICAST  MTU:1500  Metric:1
```

```
RX packets:27501 errors:0 dropped:0 overruns:0 frame:0
```

```
TX packets:20209 errors:0 dropped:0 overruns:0 carrier:0
```

```
collisions:0 txqueuelen:10001
```

```
RX bytes:22693532 (21.6 MiB)  TX bytes:3685904 (3.5 MiB)
```

```
Interrupt:17
```

We can also control the size of the maximum send and receive transport socket buffers. We can use the following values in our */etc/sysctl.conf* file:

- *net.core.rmem_default* – the default buffer size used for the receive buffer
- *net.core.wmem_default* – the default buffer size used for the transmit buffer
- *net.core.rmem_max* – the maximum receive buffer used by an application
- *net.core.wmem_max* – the maximum send buffer used by an application

We can always find out the current values of these parameters using the *sysctl -a* command as follows:

```
[root@legacy ~]# sysctl -a | grep net.core.r

net.core.rmem_max = 131071

net.core.rmem_default = 114688

[root@legacy ~]#
[root@legacy ~]# sysctl -a | grep net.core.w

net.core.wmem_max = 131071

net.core.wmem_default = 114688

net.core.warnings = 1
[root@legacy ~]#
```

We'll now try to understand how to tune the size of the TCP buffer.

A high-bandwidth connection is also known as a fat pipe, and a high-latency connection is also referred to as a long pipe. For maximum throughput, we would like to transmit continuously. TCP buffers can be tuned by using the *net.ipv4.tcp_rmem* parameter for reads and *net.ipv4.tcp_wmem* parameter for writes. For each, *sysctl* holds three values: the minimum, the default and the maximum size.

For tuning the TCP buffer, we need to calculate the BDP (Bandwidth Delay Product), i.e., the length of pipe. The formula for calculating the BDP is: *Length of Pipe =*

Bandwidth x Delay (RTT), i.e., $L = B \times D$. The value of BDP is in bytes.

Using *ping*, we can determine the RTT value. Let's assume that we are using a 100 MiBps NIC, and the value of time that *ping* gives us is roughly 25.999 ms.

```
So the BDP = L      = 100 MiBps x 25.999 ms
                  = [ (100 x 1024 x 1024) x ( 0.025 ) ] / 8 bytes
                  = 327680 bytes
```

Let's look at the default values for the TCP read buffer in our system:

```
[root@legacy ~]# cat /proc/sys/net/ipv4/tcp_rmem
```

```
4096      87380      3366912
```

```
[root@legacy ~]#
```

```
[root@legacy ~]# sysctl -a | grep tcp_rmem
```

```
net.ipv4.tcp_rmem = 4096      87380      3366912
```

```
[root@legacy ~]#
```


Now I can change the values that have been calculated by BDP, by using:

```
echo ' 327680 327680 327680' > /proc/sys/net/ipv4/tcp_rmem
```

...or I can edit the `/etc/sysctl.conf` file, as shown below:

```
net.ipv4.tcp_rmem = "327680 327680 327680"
```

Remember to run `sysctl -p` after making any changes to `/etc/sysctl.conf`. Also remember to take a look at the man page for the `tc` command.

With this, I conclude this series of articles on 'Performance Tuning and Monitoring'. I do hope I have been able to share some goods tips with you, and look forward to sharing with you something more exciting, soon.  **END**

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.