Red Hat Performance Briefs

Low Latency Performance Tuning Guide for Red Hat Enterprise Linux 6

Jeremy Eder, Senior Software Engineer

Version 1.0

September 2012

redhat.

1801 Varsity Drive™
Raleigh NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588
Research Triangle Park NC 27709 USA

The GPG fingerprint of the security@redhat.com key is:
CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

Send feedback to refarch-feedback@redhat.com

Table of Contents

1 Executive Summary

This paper provides a tactical tuning overview of Red Hat Enterprise Linux 6 for latency-sensitive workloads. In a sense, this document is a cheat-sheet for getting started on this complex topic and is intended to *complement* existing Red Hat documentation.

It is very important to gain a deep understanding of these tuning suggestions before you apply them in any production scenario, as your-mileage-will-vary. Note that certain features mentioned in this paper may require the latest minor version of Red Hat Enterprise Linux 6.

2 Process Scheduling

Linux provides the user with a system of priorities and scheduler policies that influence the task scheduler. A common trait of latency-sensitive tasks is that they should run continuously on the CPU with as few interruptions as possible. If your application is being scheduled off the CPU in favor of an unimportant task, try increasing its priority using **`nice`**. Another option is to change the scheduler policy to FIFO with a priority of 1. The example command below runs YOURPROC ahead of the userspace and some kernel threads. Note there are some kernel threads that run FIFO policy. This has been met with mixed results, and often using SCHED_OTHER via **`nice`** performs similarly.

```
# chrt -f 1 ./YOURPROC
# nice -20 ./YOURPROC
```

When using the SCHED_FIFO policy for your application, it is possible to introduce latency spikes or other anomalies by blocking kernel threads that use SCHED_OTHER. All SCHED_OTHER tasks are of lower priority than SCHED_FIFO. For this reason it is important that you test extensively when using the FIFO scheduler. As an example, a kernel thread relevant to networking is **`ksoftirqd`**. To determine if observed latency blips are priority-related, try the below command. In the report, you might see **`ksoftirqd`** at the top of the list, meaning it was likely blocked:

```
# perf sched record -o /dev/shm/perf.data ./YOURPROC
# perf sched -i /dev/shm/perf.data latency > /tmp/perf.latency.report

 Task           | Runtime ms  | Switches | Average delay ms
-----------------------------------------------------------
 ksoftirqd/2:23 |   0.000 ms |        26 | avg: 3388.621 ms
```

2.1 Avoiding Interference

To elaborate further on process priorities and scheduling, it is important to streamline the system such that only essential tasks are scheduled, for example, by ensuring that you have disabled all unnecessary services. There is limited flexibility with regard to kernel threads as compared to userspace threads. Here are some options for task affinity and isolation to reduce jitter and latency:

`isolcpus`
Isolate CPU cores from *userspace tasks*. This can be done with i.e. **`isolcpus=1,3,5,7,9,11`**. **`isolcpus`** requires a reboot, and there will continue to be kernel threads on isolated cores. **`isolcpus`** sets the affinity of the **`init`** task to the inverse bitmask of that specified by **`isolcpus`**. All userspace tasks will inherit from **`init`**, including any new tasks. To land a task on one of the isolated cores, you need to explicitly ask for it via **`taskset/numactl`** or **`sched_setaffinity`**.

See the Tuna User Guide for a flexible isolation tool (part of the Red Hat MRG Realtime product).

Isolate sockets from userspace processes, eventually pushing them to socket 0:

```
# tuna -S1 -i ; tuna -S2 -i ; tuna -S3 -i
```

Push YOURPROC to core 1 using **`tuna`** and set sched/prio to fifo:1

```
# tuna -t `pgrep YOURPROC` -C -p fifo:1 -S 1 -m -P | head -5
```

`cgroups` can provide a user-friendly way of grouping NUMA resources. An example is */etc/cgconfig.conf*:

```
group node0 {
    cpuset {
        cpuset.cpus = “0,2,4,6,8,10”;
        cpuset.cpu_exclusive = 1;
        cpuset.mems = 0;
    }
}

group node1 {
    cpuset {
        cpuset.cpus = “1,3,5,7,9,11”;
        cpuset.cpu_exclusive = 1;
        cpuset.mems = 1 ;
    }
}
```

To launch a task within the node1 cgroup, use:

```
# cgexec -g cpuset:node1 ./YOURPROC
```

See the Resource Management Guide for more information, including how to use /etc/cgrules.conf to write rules that automatically place new tasks in the desired cgroup.

2.2 Scheduler Tunables

The following scheduler tunables have been found to impact latency-sensitive workloads and can be adjusted by the user:

`sched_min_granularity_ns`
Set it to a lower value for latency-sensitive tasks (or huge thread-count). Set it to a higher value for compute-bound/throughput oriented workloads. Adjust by a factor of 2-10x.

`sched_migration_cost`

Specifies the amount of time after the last execution during which a task is considered to be “cache hot” in migration decisions. Increasing the value of this variable reduces task migrations. Adjust by a factor of 2-10x. Task migrations may be irrelevant depending on the specific task affinity settings you've configured.

2.3 Perf

`Perf` is a utility to read performance counters in both hardware (CPU) and kernel tracepoints.

When optimizing for better latency, you may be concerned with CPU counters such as cache-misses, cpu-migrations, page-faults. Software events such as scheduler, network, the VM, or timers can also be tracked and reported using **`perf`**.

Documentation is available in the Developer Guide, man pages, or in the kernel source.

Here are some brief examples to get you started:

Find out what capabilities the CPU/kernel can provide. CPUs may differ:

```
# perf list
```

Find out what the CPU is currently spending cycles on. (i.e. hot code paths/functions:):

```
# perf top
```

Profile YOURPROC with regard to block, ext4, and sched kernel tracepoints:

```
# perf stat -e kmem:* -e block:* -e ext4:* -e sched:* ./YOURPROC
```

To view the hardware cache-misses counter triggered by the command ls:

```
# perf stat -e cache-misses ls
```

3 Memory

Recent server platforms are physically wired up in a NUMA configuration. When you pin a process to a specific CPU or core there are going to be certain combinations that will perform faster than others, because memory banks and/or PCIe slots are local to certain CPU sockets.

3.1 NUMA Topology

Use **`hwloc`** to visualize hardware topology. **`hwloc`** uses the BIOS SLIT table, and *this table may or may not be fully populated depending on your server's BIOS. You may need to determine this experimentally*. See hardware vendor documentation.

`numactl`: Bind apps to sockets and local memory banks. i.e. CPU socket 0/memory bank 0:

```
# numactl -N0 -m0 ./YOURPROC
```

taskset: Bind apps to cores. Kernel best-effort for memory locality.

```
# taskset -c 3 ./YOURPROC
```

View current CPU/memory pinning done with either **numactl** or **taskset**:

```
# grep -i allowed /proc/PID/status
```

- Adjust the VM's tendency to swap by setting **vm.swappiness=0**
- Consider disabling the **ksm** daemon.
- Use the **numastat** or **numa_faults.stp systemtap** script to troubleshoot NUMA misses.
- Use Intel Performance Counter Monitor to track QPI link usage.
- Both network and storage performance can benefit from PCI slot locality and IRQ affinity.

3.2 Hugepages

The default 4KB page size in Linux can introduce a performance hit in systems with a large memory footprint. 2 MB hugepages reduce the number of pages by a factor of 512. Here is one way to configure hugepages.

Getting information about hugepages:

```
# egrep -i 'thp|trans|huge' /proc/meminfo /proc/vmstat
```

3.3 Transparent Hugepages

Transparent hugepages (THP) is an abstraction layer that automates most aspects of creating, managing, and using hugepages. THP instantiates a background kernel thread called **khugepaged** that scans memory for merging candidates. THP is enabled by default. Depending on your tolerance, these scans may introduce jitter. To disable THP:

```
transparent_hugepage=never on the kernel cmdline (requires reboot)
# echo never > /sys/kernel/mm/redhat_transparent_hugepage/enabled
```

4 Network

When tuning network performance for low latency, the goal is to have IRQs land on the same core or socket that is currently executing the application that is interested in the network packet. This increases CPU cache hit-rate (see section on **perf**) and avoids using the inter-processor link.

4.1 IRQ processing

Red Hat has published an in-depth whitepaper on IRQ affinity tuning.

An automated way to achieve affinity (requires Red Hat Enterprise Linux 6.2 or later), is to enable Receive Flow Steering (RFS). RFS (and its transmit counterpart XPS) are highly

dependent on hardware and driver capabilities. Certain drivers steer in hardware.

4.2 Drivers

Certain NIC vendors provide low-latency tuning guidelines, which you should read and test.

They may also provide scripts that line up IRQs with cores, i.e. a 1:1 relationship. Others have the ability to vary the number of RX/TX queues. Consider using those scripts, or try:

```
# irqbalance --oneshot
```

4.3 Network Tunables

tcp_low_latency
Demonstrated minimal impact (1us range).

Use TCP_NODELAY socket option where applicable.

4.4 ethtool

Many statistics (**ethtool -S**) are hardware/driver dependent. Some NICs track stats in firmware, others in-kernel. When tracking is done in firmware, it is difficult to understand the true meaning of stats (i.e. what causes an error counter to increment). Consult your hardware vendor documentation.

Coalescing: (**ethtool -cC**)

Quantity of packets the NIC will accept before triggering an interrupt. Experiment with a coalesce value of 0 or 1. Verify you have enough remaining CPU cycles to handle bursts.

Ring buffers: (**ethtool -gG**)
Set of buffers that provide a very small amount of memory to deal with cases when the CPU is not available to process packets. Driver-dependent, output may be in slots or bytes. There has been a move to Byte Queue Limits, which avoids a head-of-line blocking problem caused by the slots technique. Increase the value to deal with drops, but watch for added latency.

Offload Capabilities: (**ethtool -kK**)
Network adapter may provide some amount of offload capabilities, such as TCP Segmentation Offload (TSO), Large Receive Offload (LRO) etc. Offloads move some of the network processing out of the kernel and onto the NIC. These generally save CPU cycles, but can be a mixed bag in terms of latency performance as many are designed to increase throughput and decrease CPU utilization through batching techniques. For example, Generic Receive Offload (GRO) may aggregate packets up to 64KB. Note that since offloads occur between the OS and the wire, their properties are generally not observable with **tcpdump** on sender/receiver; use a port-mirror or equivalent tap device to see what the wire traffic looks like. Offloads will modify your packet quantity/frame size and flow characteristics. These may vary considerably from the MTU configured in the OS or on the network switch.

Consider:

```
# ethtool -K ethX lro off gro off tso off gso off
```

Make the ethtool configuration persistent using udev rules, your application's startup scripts, or the ETHTOOL_OPTS variable in */etc/sysconfig/network-scripts*.

5 Power

Newer CPUs may alter their performance based on a workload heuristic in order to save power. This is at odds with latency-sensitive workload requirements, causing sub-optimal performance/jitter. Here is an example script to lock the system into certain c-states, that uses the kernel's */dev/cpu_dma_latency* interface documented here.

Use the example script along with the values returned by the below command to lock CPU cores into particular C-states. C0 provides the lowest latency, at cost of higher power, temperature, and cooling costs. Depending on latency requirements, C1 may also be usable. C3 and deeper show measurable performance impact, but should be fine for off-hours workloads.

find /sys/devices/system/cpu/cpu0/cpuidle -name latency \

```
 -o -name name | xargs cat
```

Use **powertop** in Red Hat Enterprise Linux or turbostat (in upstream kernel) to view current c-states.

5.1 Tuned

Consider using the latency-performance **tuned** profile. Currently, tuned profiles also set the I/O scheduler to deadline. If you use one of these profiles, (or clone one and add your site-specific tuning) the kernel command-line option **elevator=deadline** may be redundant. Most **tuned** profiles also change the cpuspeed governor to performance, ensuring the highest clock frequency. Alternatively, disable the cpuspeed service, or modify */etc/sysconfig/cpuspeed* and set **GOVERNOR=performance**. To find current frequency, execute:

```
# find /sys -name cpuinfo_cur_freq | xargs cat
```

5.2 BIOS Configuration

Many server vendors have published BIOS configuration settings geared for lowlatency which must be carefully followed. Ensure you are running the latest BIOS version.

The **rdmsr** utility from the **msr-tools** package is used to read Machine State Registers off of a CPU. For example, on certain newer generation Intel CPUs:

```
# ./rdmsr -d 0x34
97
```

This means 97 SMIs have fired since boot. After implementing low-latency tuning guidelines from your server vendor, this counter should rarely (if ever) increment after boot. It would be useful to query this value before and after a test run to verify if any SMIs have fired.

6 Kernel boot parameters

Click here for kernel documentation for Red Hat Enterprise Linux. An example low-latency command line:

```
nosoftlockup mce=ignore_ce
```

- **`nosoftlockup`** disables logging of backtraces when a process executes on a CPU for longer than the softlockup threshold (default 120 seconds). Typical low-latency programming and tuning techniques might involve spinning on a core or modifying scheduler priorities/policies, which can lead to a task reaching this threshold. If a task has not relinquished the CPU for 120 seconds, the kernel will print a backtrace for diagnostic purposes. Note that adding nosoftlockup to the cmdline will simply disable the printing of this backtrace and does not in itself reduce latency.
- **`mce=ignore_ce`** ignores corrected errors and associated scans that can cause periodic latency spikes.
- Consider **`audit=0`** to disable the kernel components of the audit subsystem which have been measured at about 1-3% CPU utilization when under heavy load. Also ensure to **`chkconfig auditd off`**.

Missing from this list are **`idle`**, **`processor.max_cstate`**, **`intel_idle.max_cstate`** and **`idle`**. These options require a reboot to adjust, and thus we recommend the */dev/cpu_dma_latency* interface as it achieves an equivalent performance improvement without the reboot requirement. You can save considerably in power and cooling costs by locking c-states only as necessary (i.e. during trading hours).

- Consider disabling SELinux using */etc/selinux/config* (measured 1-3% CPU overhead)
- Use **`tuned`** profiles (i.e. enterprise-storage) to control I/O elevator, rather than cmdline.

7 References

- Performance Tuning Guide:

  https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Linux/6/html-single/Performance_Tuning_Guide/index.html

- Realtime Tuning Guide:

  https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_MRG/2/html-single/Realtime_Tuning_Guide/index.html

- Resource Management Guide:

  https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Linux/6/html-single/Resource_Management_Guide/index.html

- Developer Guide:

  https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Linux/6/html-single/Developer_Guide/index.html#perf

- Tuna User Guide:

  https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_MRG/2/html-single/Tuna_User_Guide/index.html

- NUMA Overview:

  https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Linux/6/html-single/Performance_Tuning_Guide/index.html#s-cpu-numa-topology

- NUMA Topology Visualization Tool, hwloc:
  https://access.redhat.com/knowledge/solutions/62879

- Hugepages Overview:
  https://access.redhat.com/knowledge/sources/source_rpms/kernel-2.6.32-279.el6/tree/Documentation/vm/hugetlbpage.txt

- Hugepages Howto:

  https://access.redhat.com/knowledge/solutions/33613

- Hugepages/Transparent Hugepages Overview:

  https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Linux/6/html-single/Performance_Tuning_Guide/index.html#s-memory-transhuge

- Optimizing Red Hat Enterprise Linux Performance Tuning IRQ Affinity:
  https://access.redhat.com/knowledge/techbriefs/optimizing-red-hat-enterprise-linux-performance-tuning-irq-affinity

- Receive Flow Steering (RFS) Howto:
  https://access.redhat.com/knowledge/solutions/62885

- C-state Lock Example Script ****unsupported****:
  http://git.fedorahosted.org/cgit/tuned.git/tree/libexec/pmqos-static.py?h=1.0

- /dev/cpu_dma_latency documentation:
  https://access.redhat.com/knowledge/sources/source_rpms/kernel-2.6.32-279.el6/tree/Documentation/power/pm_qos_interface.txt

- turbostat ****unsupported****:

  http://git.kernel.org/?p=linux/kernel/git/torvalds/linux.git;a=tree;f=tools/power/x86/turbostat;h=f74ba384500e5fbc9a5091e7e31341d87a99c094;hb=HEAD

- Kernel command line parameter documentation:
  https://access.redhat.com/knowledge/sources/source_rpms/kernel-2.6.32-279.el6/tree/Documentation/x86/x86_64/boot-options.txt

- SELinux Documentation:

  https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Linux/6/html-single/Security-Enhanced_Linux/index.html#sect-Security-Enhanced_Linux-Enabling_and_Disabling_SELinux-Disabling_SELinux

8 Revision History

Revision 1.0	Thursday, September 27, 2012	Jeremy Eder
Initial Release		