RED HAT SUMMIT

Performance Analysis and Tuning Red Hat Enterprise Linux （RHEL8.2, 7.7, Virt and Podman improvements)

D. John Shakshober
Sr Distinguished Eng
Tech Director RH Perf+Scale

Larry Woodman
Sr Distinguished Eng
R H Kernel Eng

July 2020

Agenda: Red Hat Performance Analysis Tuning 2020

- **RHEL Evolution 5->6->7-8 , What’s new for perf in RHEL8!**
  - Red Hat Perf Lab results
  - New IO and Network Improvements
- **Networking**
  - Low Latency Network (cpu_partitioning tuned), CVEs
  - XDP, eBPF denial of service
- **Disk IO**
  - Database / File system improvements w/ RHEL8
- **Virtual Memory**
  - Non-Uniform Memory Access (NUMA)
  - HugePages
  - 5 Level Page Tables - NvDIMM persistent memory.
- **Tools** Perf, PCP, Pbench, eBPF, and Insight collaboration

RHEL Performance Evolution

RHEL5

Static Hugepages

CPU Sets

Ktune on/off

CPU Affinity (taskset)

NUMA Pinning (numactl)

irqbalance

RHEL6

Transparent Hugepages

Tuned - Choose Profile

NUMAD - userspace

cgroups

irqbalance - NUMA enhanced

RHEL7

Tuned - throughput-performance (default)

Automatic NUMA-balancing

Containers/OCI - CRI-O (podman)

irqbalance - NUMA enhanced

RHEL8

5 level PTEs
(THP cont)

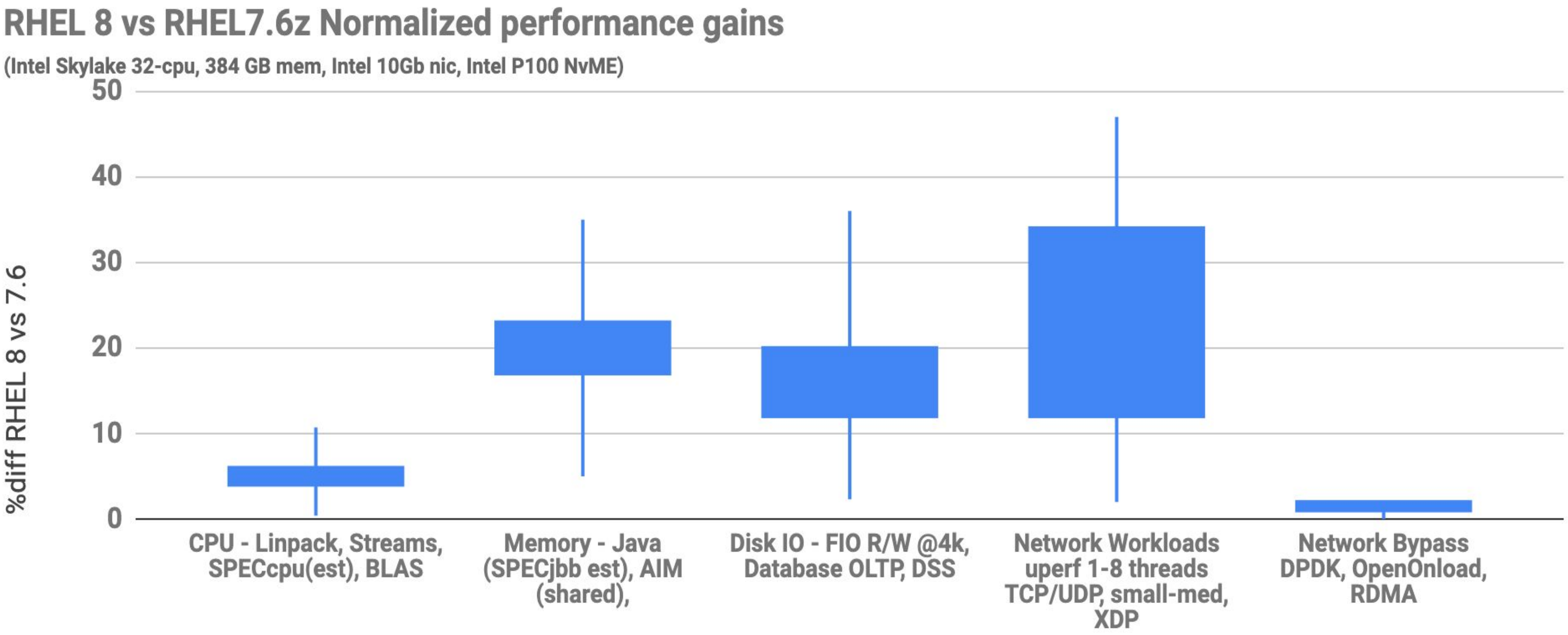
Tuned: Throughput/
Lat - SSD/Nvdimm

Multi-Arch:
Intel/ AMD/
ARM/ Power

Networking:
XDP and eBPF

Acceleration
GPU/FPGA/Offloads

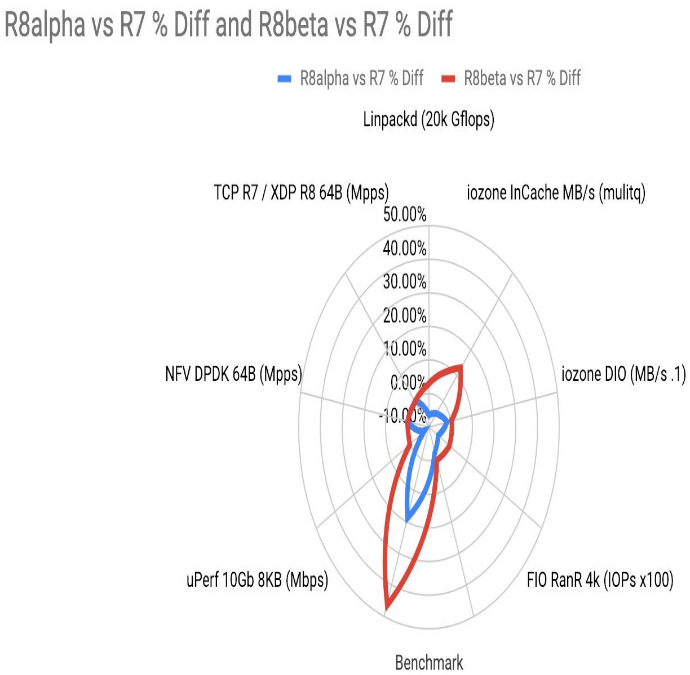
RHEL 8 vs RHEL 7 Workload Performance Gains



RHEL 8 Performance Coverage

Benchmarks – code path coverage

- CPU – linpack, lmbench
- Memory – lmbench, McCalpin STREAM
- Disk IO – iozone, fio – SCSI, FC, iSCSI
- Filesystems – iozone, ext3/4, xfs, gfs2, gluster, ceph
- Networks – netperf – 10/40/100 Gbit, Infiniband/RoCE, Bypass, DPDK
- Bare Metal, KVM, Containers
- White box AMD/Intel/Arm / (Power TBD)
- HW OEM partners



RHEL 8 Performance improvements w/ AIM7

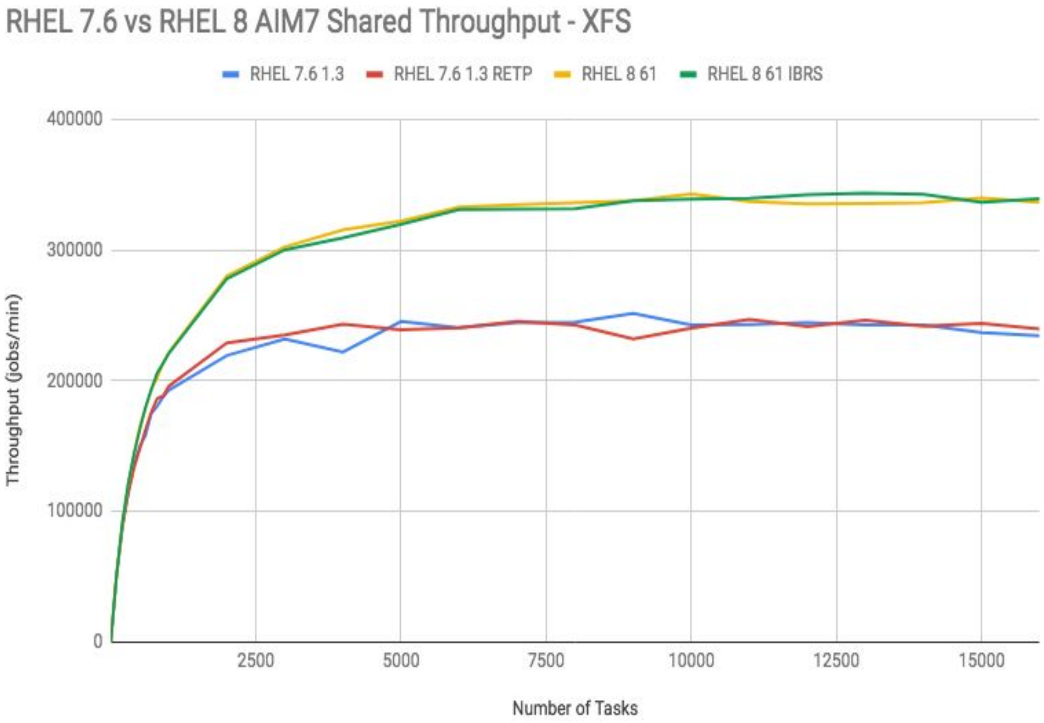
AIM7 Shared User Mix - multiuser benchmark, throughput in jobs/min +35.6%

RHEL 7.6, page fault stack not present.

raw_spin_unlock_irqrestore
_raw_spin_unlock_irqrestore
__wake_up
xlog_state_do_callback
xlog_state_done_syncing
xlog_iodone
xfs_buf_ioend
Xfs_buf_ioend_work

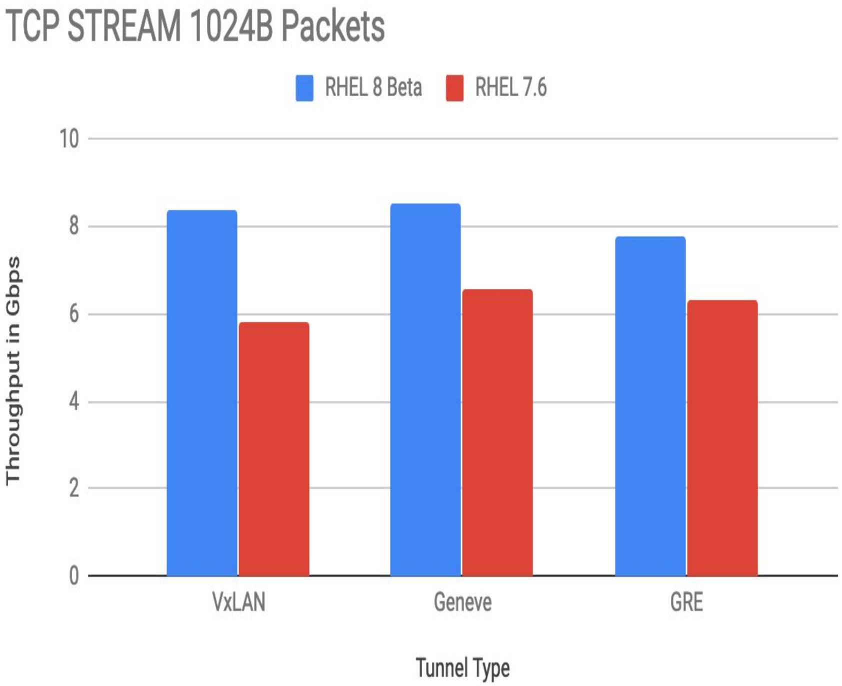
RHEL 8

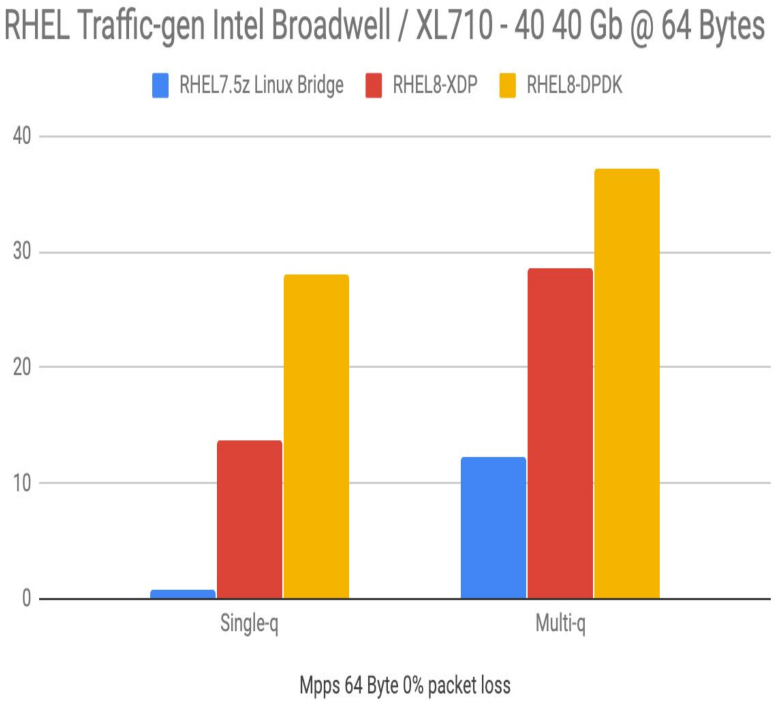
filemap_map_pages+187
handle_pte_fault+2406
__handle_mm_fault+1066
handle_mm_fault+218
__do_page_fault+586
do_page_fault+50
page_fault+30



RHEL 8 Network Performance TCP / XDP

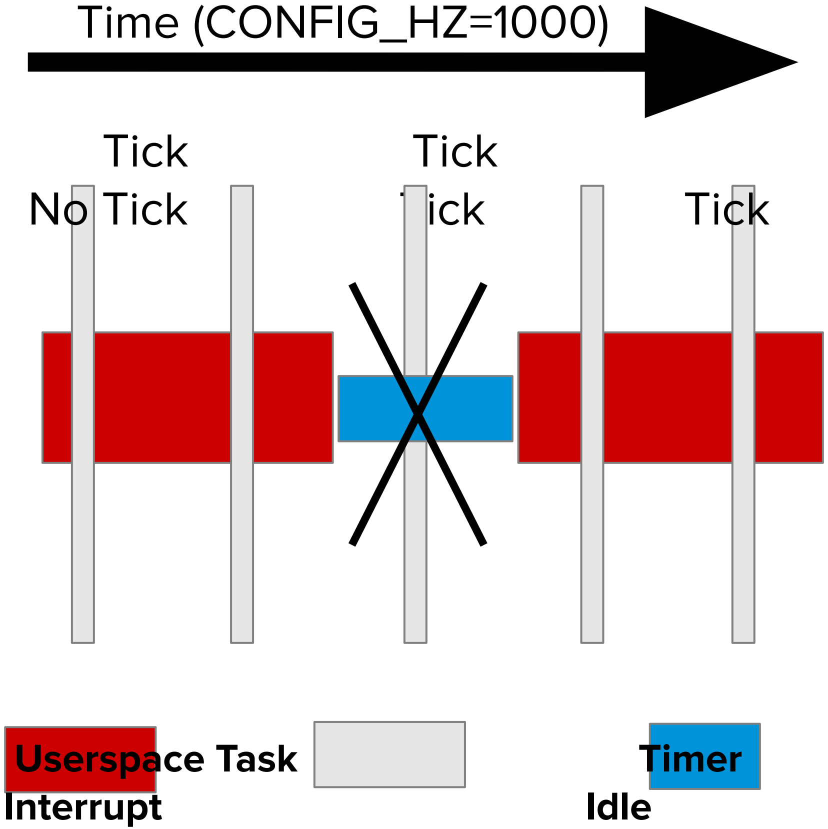
RHEL8 Network Performance out-of-the-box - 10 Gb @ 1k, 40Gb @ 64b Intel Nics





RHEL Tickless

User tasks interupted 1000x/sec



Time (CONFIG_HZ=1000)

Tick

No Tick

Tick

Tick

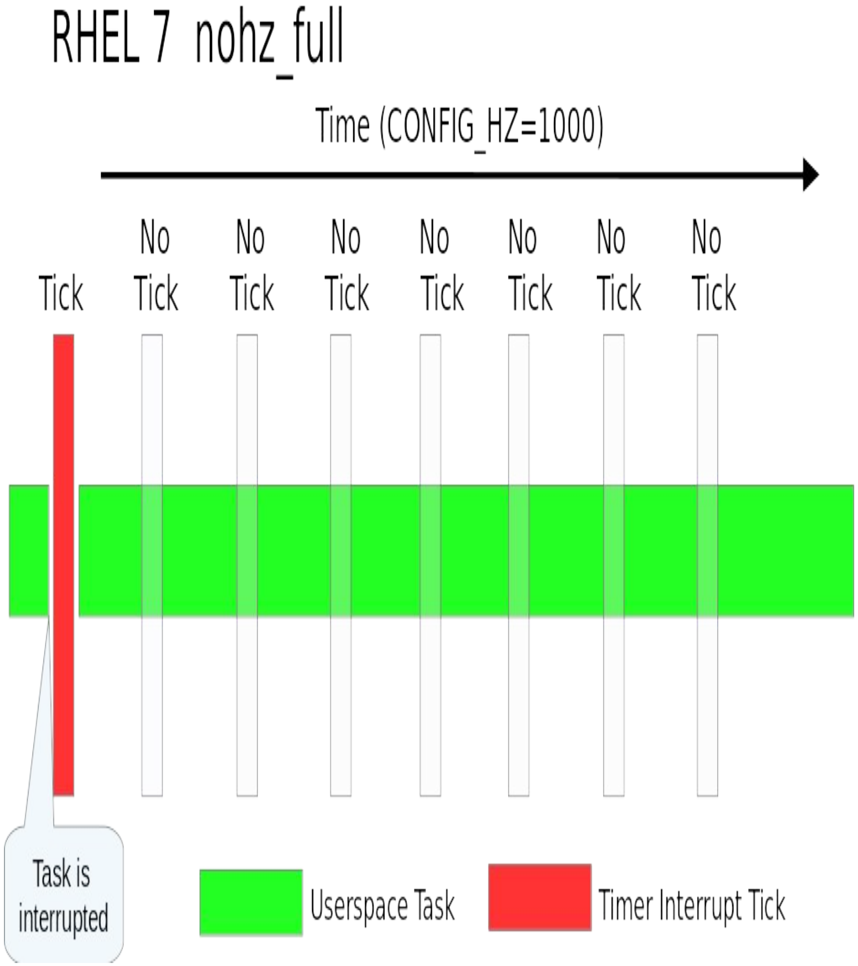
Tick

Userspace Task

Interrupt

Timer

Idle



Tuned Profiles throughout Red Hat's Product Line

RHEL7/8 Laptop/Workstation

balanced

RHEL7/8 Server/HPC

throughput-performance

RHEL7/8 KVM Host, Guest

virtual-host/guest

RHV/OSP

virtual-host

Red Hat Storage

rhs-high-throughput

RHEL OSP (compute node)

Virtual-host/guest

Open Shift Platform

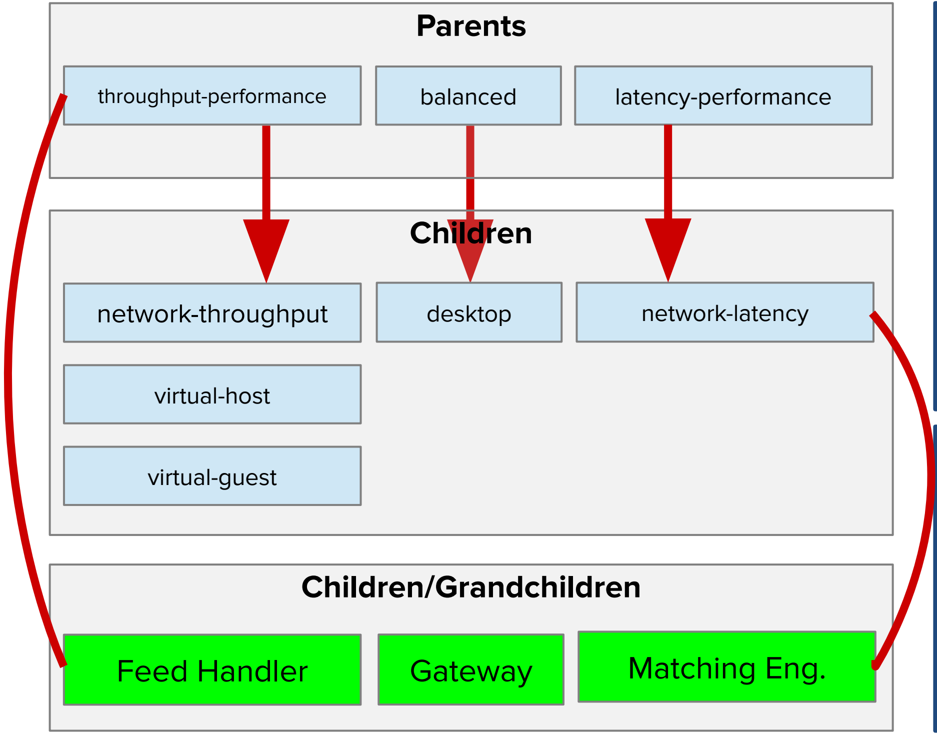
control-plane/node

NFV / RT

cpu_partitioning/rt

Tuned 8.2 supports **arch-specific tuning**

Examples Cascadelake N (cstates), AMD (Epyc numa/scheduler), Power (cstates)



latency-performance

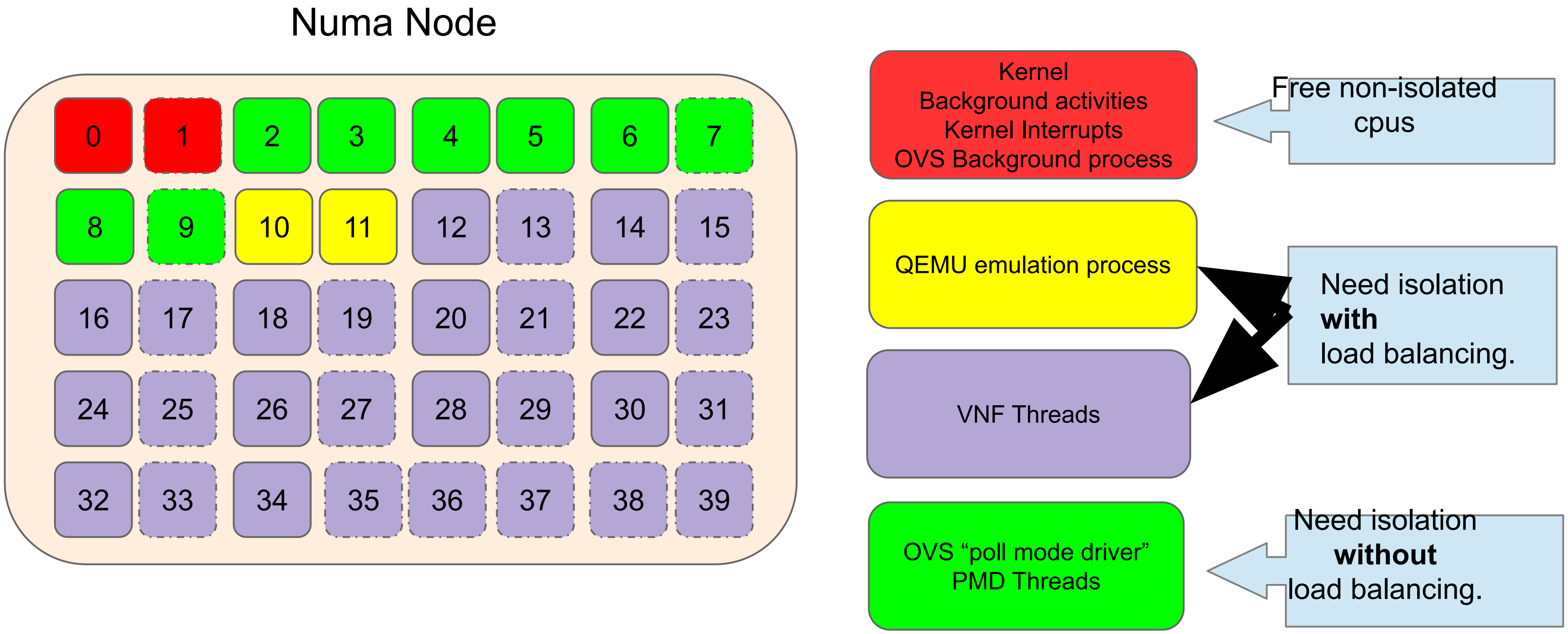
```
force_latency=1
governor=performance
energy_perf_bias=performance
min_perf_pct=100
vm.dirty_ratio=10
vm.dirty_background_ratio=3
vm.swappiness=10
kernel.sched_min_granularity_ns=10000000
kernel.sched_migration_cost_ns=5000000
```

network-latency

```
include=latency-performance
transparent_hugepages=never
net.core.busy_read=50
net.core.busy_poll=50
net.ipv4.tcp_fastopen=3
kernel.numa_balancing=0
```

CPU Partitioning tuned profile

Simple, flexible low-latency cpu isolation tuning.



Numa Node

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

Kernel
Background activities
Kernel Interrupts
OVS Background process

Free non-isolated cpus

QEMU emulation process

VNF Threads

Need isolation **with** load balancing.

OVS “poll mode driver” PMD Threads

Need isolation **without** load balancing.

“cpu-partitioning” tuned profile

For latency sensitive applications needing kernel scheduler load balancing.

Does all the “heavy lifting” for you.

1) Just edit *<u>/etc/tuned/cpu-partitioning-variables.conf</u>*

```
# Isolated CPUs with kernel load balancing:
isolated_cores=10-39
# Isolated CPUs without kernel load balancing:
no_balance_cores=2-9
```

1) Set the cpu-partitioning tuned profile.

```
# tuned-adm profile cpu-partitioning
```

1) Then reboot!

- After a reboot you should have the following to the kernel boot line:

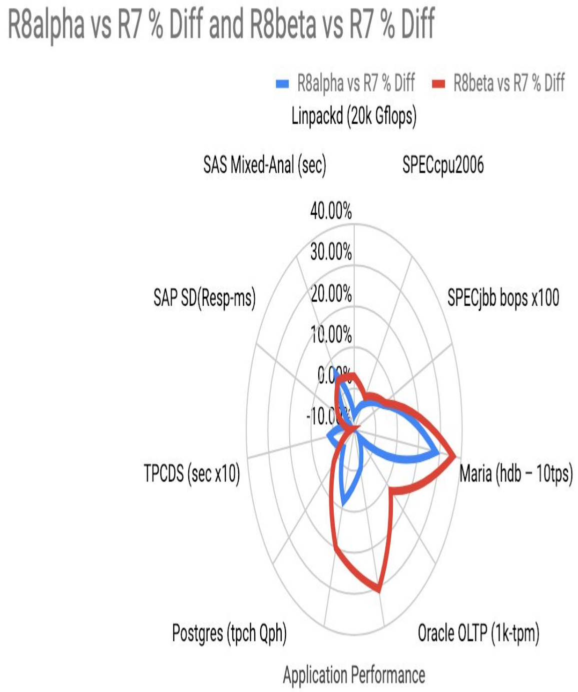
```
skew_tick=1
nohz=on
nohz_full=2-39
rcu_nocbs=2-39
tuned.non_isolcpus=0000000003
intel_pstate=disable
Nosoftlockup
```

- Moves all users tasks off the isolated cpus
  - Including all children of systemd (pid 1)
  - All future processes too, as default system cpu affinity is changed.

RHEL8 Application Performance Coverage

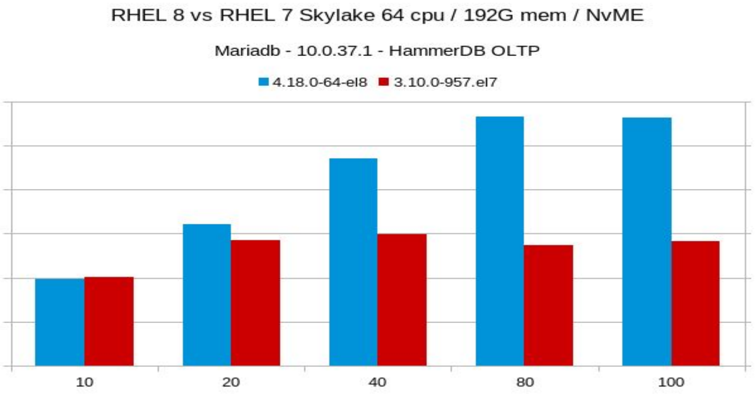
Application Performance

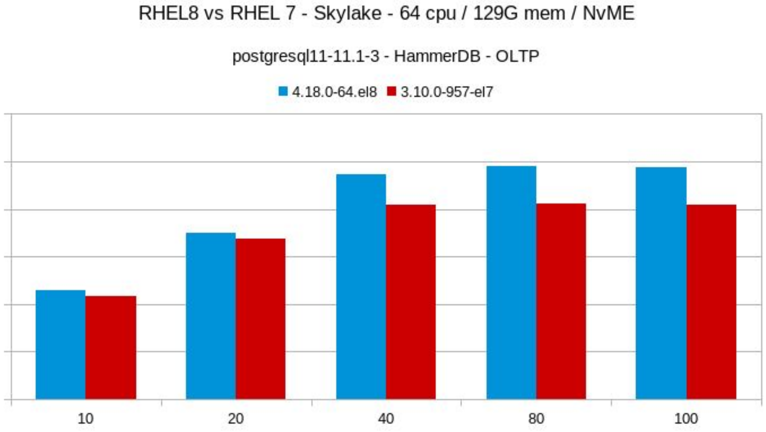
- Linpack HPC
- SPECcpu2006, SPECjbb2005
- Database: Oracle 12, SQLserver, MariaDB, PostgreSQL
    - OLTP – BM, KVM, RHV – TPC-C/E
    - DSS – BM, KVM, RHV - TPC-H-DS
- AIM 7 – shared, compute, high-cont
- SAP – ERP (SD), HANA (pboffline)
- SAS – Mixed Analytics, Grid
- STAC - N (nic lat), A2 (GPU accel)



RHEL 8 - Database tuning tips

- **MariaDB**
  - Huge pages
    - Reduce TLB misses
    - For wiring down database pages
    - Prevent swapping
  - Lower dirty background ratio / Increase dirty ratio
    - To start early reclaim of dirty blocks
  - Size buffer pool based on user connections
    - To prevent memory pressure
- **Postgres**
  - Use Huge pages
    - Reduce TLB misses
    - For wiring down database pages
    - Prevent swapping
  - Lower dirty background ratio / Increase dirty ratio
    - To start early reclaim of dirty blocks
  - Configure Shared buffers as well as effective cache
  - size to avoid memory pressure

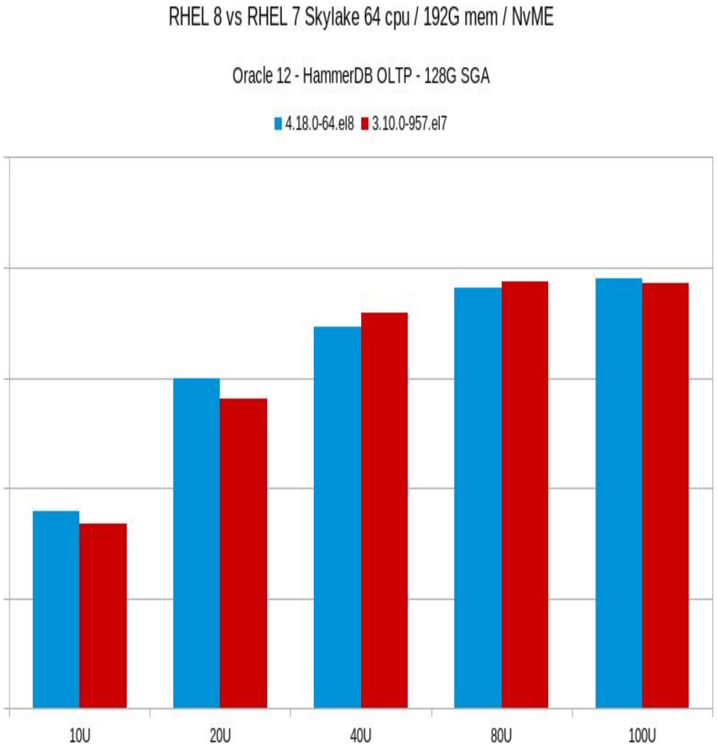




RHEL 8 - Database tuning tips

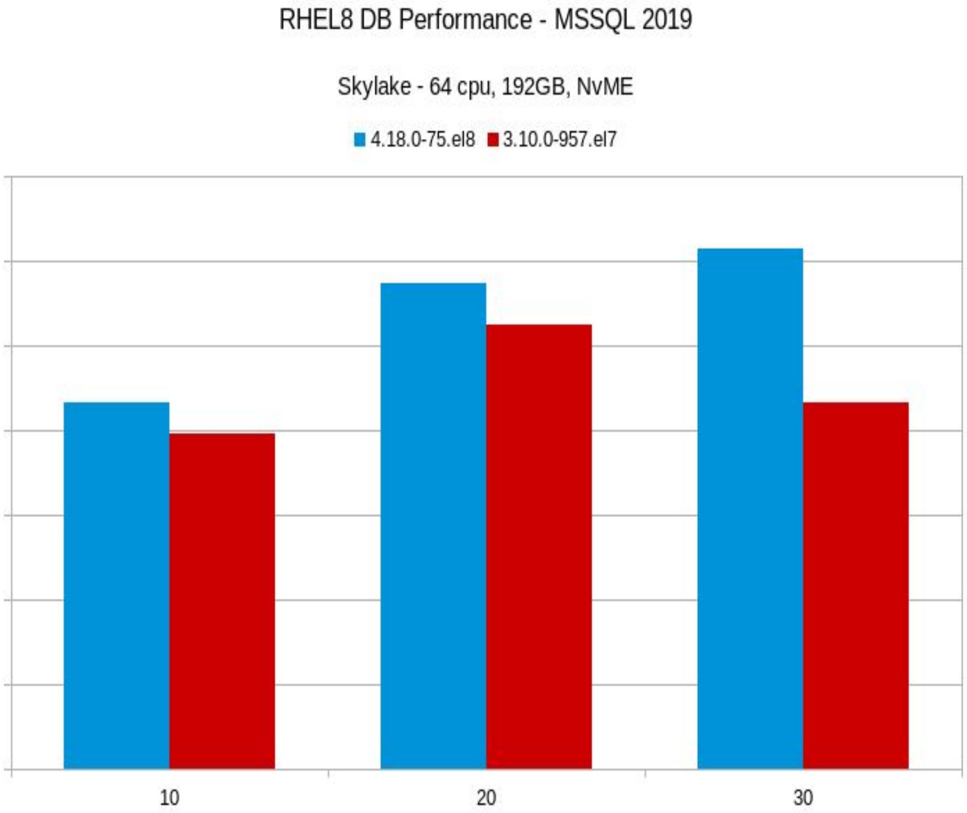
Oracle 12c

- Implement huge pages
  - Reduce TLB misses
  - For wiring down database pages
  - Prevent swapping
- Turn off Auto numa
  - To prevent conflict with Oracle NUMA optimization
- Turn of transparent huge pages
  - To reduce CPU overhead of THP scan
- Lower dirty background ratio
  - Start flushing dirty blocks and reclaim
- Increase dirty ratio
  - Delay the process of hitting dirty blocks threshold
- Use numa pinning in multiple instance environments
- To take advantage of NUMA localization
- Size SGA based on user connections
- To prevent memory pressure



RHEL 8 with Microsoft SQLServer19 Increased Performance

- Updates to the mssql tuned profile optimize tuning for decision support workloads
- New TCP/IP stack delivers increased performance and BBR congestion control
- Storage block devices now use multiqueue scheduling to make the best use of bandwidth available from modern flash-based storage devices



Microsoft redhat.

CVE Insights can detect, perf override for experiments

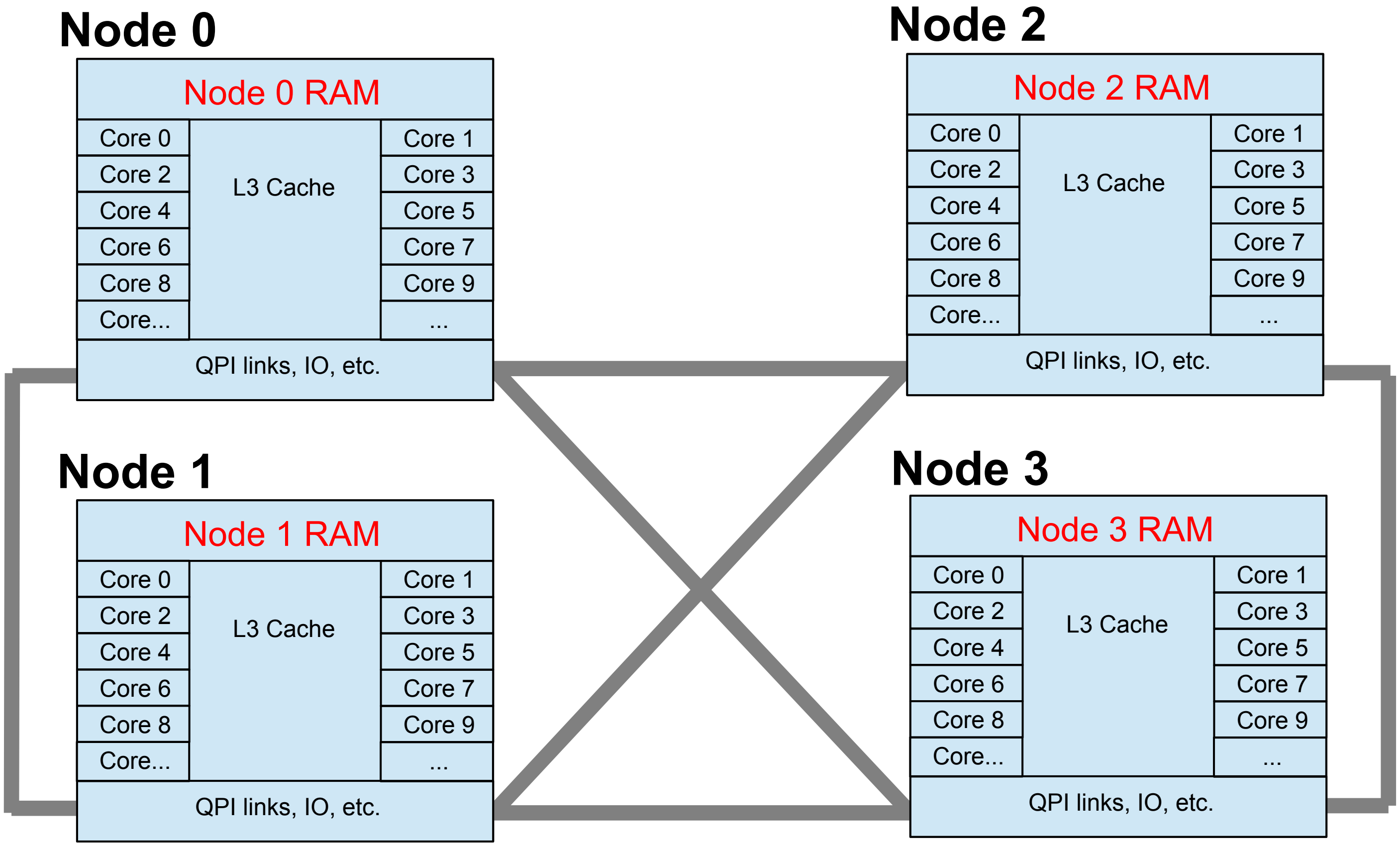
- Official Red Hat Security pages
  https://access.redhat.com/security/
- To disable CVE on RHEL-{6,7,8}, add the following to the boot grub line\
  spectre_v2=off spec_store_bypass_disable=off nopti l1tf=off mds=off
- **(New to RHEL7.7/8.1 - add mitigations=off to disable all, experiment only)**

Your resulting vulnerabilities files should then look something like these:

```
# grep . /sys/devices/system/cpu/vulnerabilities/*
     /sys/devices/system/cpu/vulnerabilities/l1tf:Mitigation: PTE Inversion; VMX: vulnerable
     /sys/devices/system/cpu/vulnerabilities/meltdown:Vulnerable
     /sys/devices/system/cpu/vulnerabilities/spec_store_bypass:Vulnerable
     /sys/devices/system/cpu/vulnerabilities/spectre_v1:Mitigation: __user pointer sanitization
     /sys/devices/system/cpu/vulnerabilities/spectre_v2:Vulnerable, IBPB: disabled, STIBP:
     disabled
```

Numa and Memory Perf Tuning

Typical Four-Node NUMA System



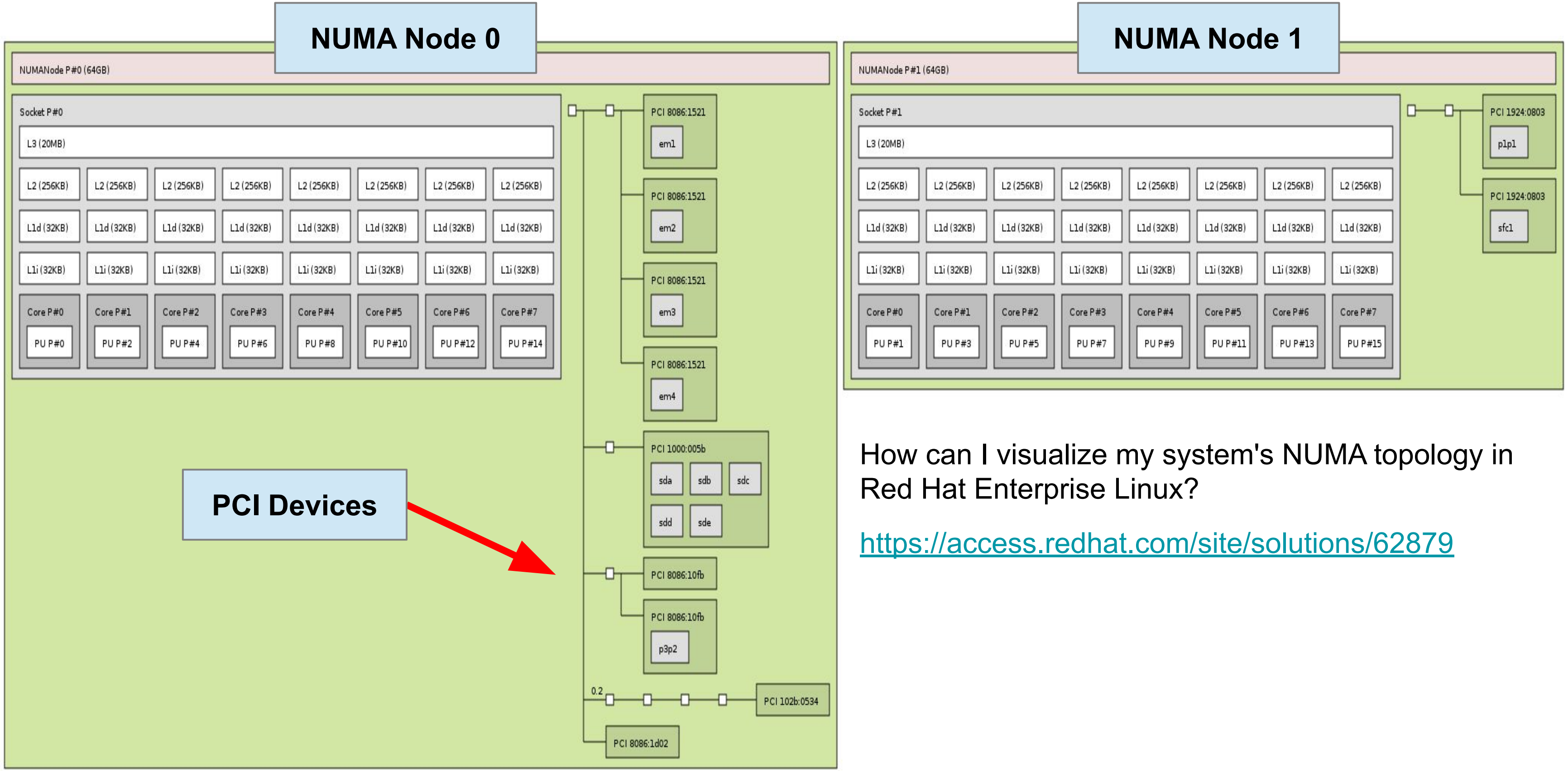
Tools to display CPU and Memory (NUMA)

```
# numactl --hardware
available: 4 nodes (0-3)
node 0 cpus: 0 4 8 12 16 20 24 28 32 36
node 0 size: 65415 MB
node 0 free: 63482 MB
node 1 cpus: 2 6 10 14 18 22 26 30 34 38
node 1 size: 65536 MB
node 1 free: 63968 MB
node 2 cpus: 1 5 9 13 17 21 25 29 33 37
node 2 size: 65536 MB
node 2 free: 63897 MB
node 3 cpus: 3 7 11 15 19 23 27 31 35 39
node 3 size: 65536 MB
node 3 free: 63971 MB
node distances:
node   0   1   2   3
  0:  10  21  21  21
  1:  21  10  21  21
  2:  21  21  10  21
  3:  21  21  21  10
```

cpus & memory for each node

Relative “node-to-node” latency costs.

Visualize NUMA Topology: lstopo


NUMA Node 0
NUMANode P#0 (64GB)
Socket P#0
L3 (20MB)
L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB)
L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB)
L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#0 PU P#2 PU P#4 PU P#6 PU P#8 PU P#10 PU P#12 PU P#14
PCI 8086:1521
em1
PCI 8086:1521
em2
PCI 8086:1521
em3
PCI 8086:1521
em4
PCI 1000:005b
sda sdb sdc
sdd sde
PCI 8086:10fb
PCI 8086:10fb
p3p2
0.2
PCI 102b:0534
PCI 8086:1d02
PCI Devices
NUMA Node 1
NUMANode P#1 (64GB)
Socket P#1
L3 (20MB)
L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB) L2 (256KB)
L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB) L1d (32KB)
L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB) L1i (32KB)
Core P#0 Core P#1 Core P#2 Core P#3 Core P#4 Core P#5 Core P#6 Core P#7
PU P#1 PU P#3 PU P#5 PU P#7 PU P#9 PU P#11 PU P#13 PU P#15
PCI 1924:0803
p1p1
PCI 1924:0803
sfc1

How can I visualize my system's NUMA topology in Red Hat Enterprise Linux?

https://access.redhat.com/site/solutions/62879

Numactl

- The numactl command can launch commands with **static** NUMA memory and execution thread alignment
  - # numactl -m <NODES> -N <NODES> <Workload>
- Can specify devices of interest to process instead of explicit node list
- Numactl can interleave memory for large monolithic workloads
  - # numactl --interleave=all <Workload>

```
# numactl   -m 6-7 -N 6-7    numactl --show
policy: bind
preferred node: 6
physcpubind: 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
cpubind: 6 7
nodebind: 6 7
membind: 6 7
```

```
# numactl   -m netdev:ens6f2 -N netdev:ens6f2    numactl --show
policy: bind
preferred node: 2
physcpubind: 20 21 22 23 24 25 26 27 28 29
cpubind: 2
nodebind: 2
membind: 2
```

```
# numactl   -m file:/data -N file:/data    numactl --show
policy: bind
preferred node: 0
physcpubind: 0 1 2 3 4 5 6 7 8 9
cpubind: 0
nodebind: 0
membind: 0
```

```
# numactl   --interleave=4-7  -N 4-7    numactl --show
policy: interleave
preferred node: 5 (interleave next)
interleavemask: 4 5 6 7
interleavenode: 5
physcpubind: 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79
cpubind: 4 5 6 7
nodebind: 4 5 6 7
membind: 0 1 2 3 4 5 6 7
```

numastat shows need for NUMA management

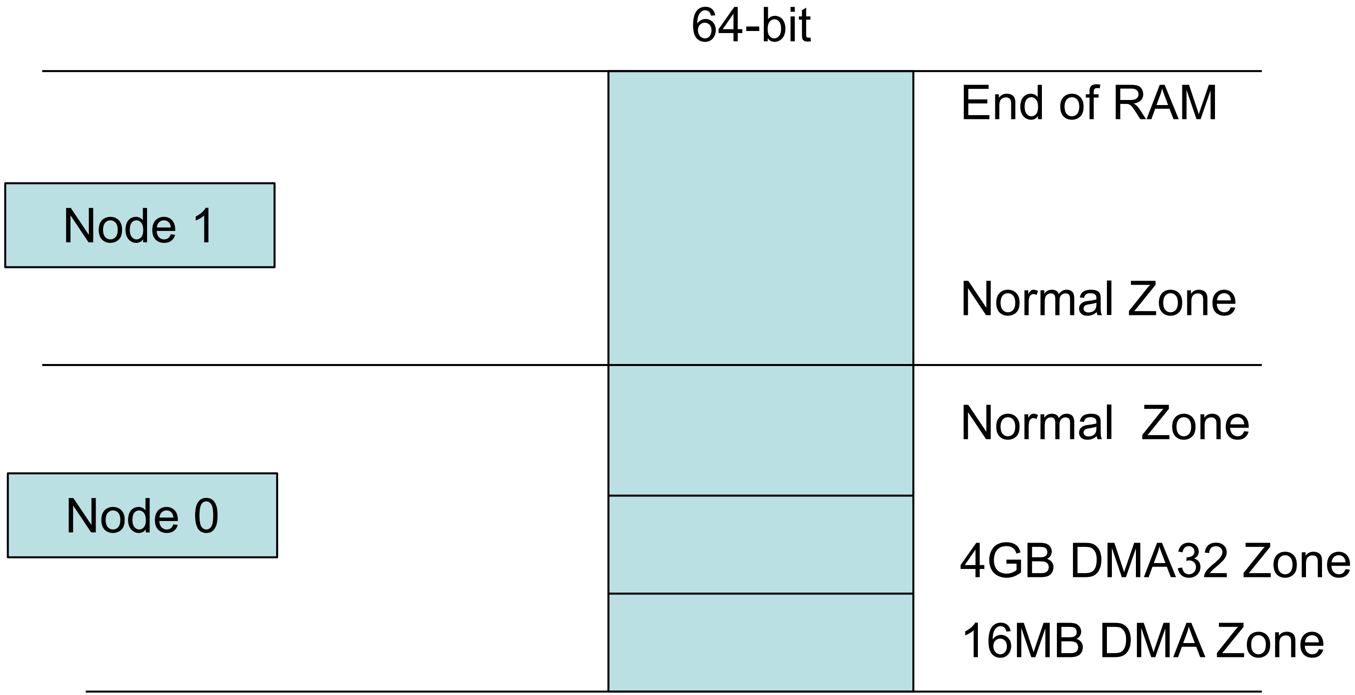
```
# numastat -c qemu  Per-node process memory usage (in Mbs)

PID              Node 0 Node 1 Node 2 Node 3 Total
---------------  ------ ------ ------ ------ -----
10587 (qemu-kvm)   1216   4022   4028   1456 10722
10629 (qemu-kvm)   2108     56    473   8077 10714
10671 (qemu-kvm)   4096   3470   3036    110 10712
10713 (qemu-kvm)   4043   3498   2135   1055 10730
---------------  ------ ------ ------ ------ -----
Total             11462  11045   9672  10698 42877
```

unaligned

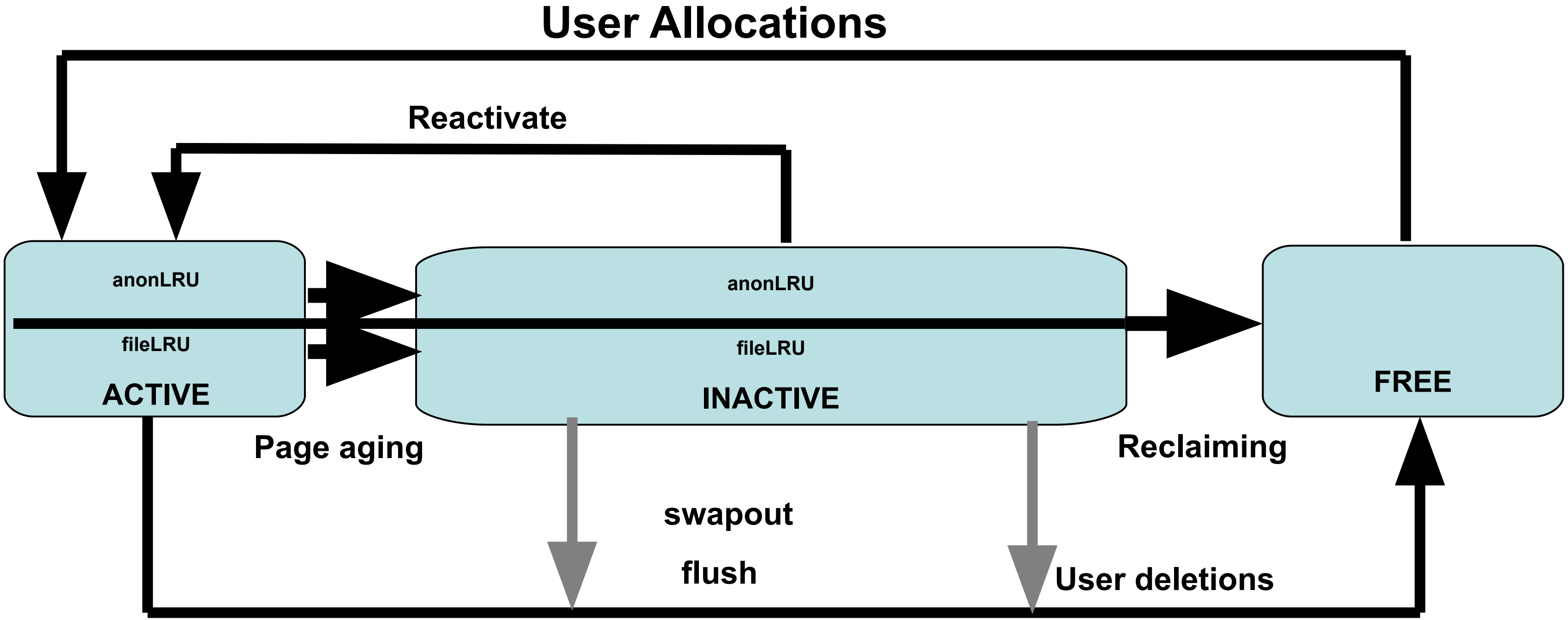
```
# numastat -c qemu

Per-node process memory usage (in Mbs)

PID              Node 0 Node 1 Node 2 Node 3 Total
---------------  ------ ------ ------ ------ -----
10587 (qemu-kvm)      0  10723      5      0 10728
10629 (qemu-kvm)      0      0      5  10717 10722
10671 (qemu-kvm)      0      0  10726      0 10726
10713 (qemu-kvm)  10733      0      5      0 10738
---------------  ------ ------ ------ ------ -----
Total             10733  10723  10740  10717 42913
```

aligned

NUMA Nodes and Zones



Per Node / Zone split LRU Paging Dynamics



HugePages

Hugepages in RHEL

- **X86_64 supports 3 page sizes:**
  - 4KB, 2MB, 1GB
- **Standard HugePages 2MB**
  - Reserve/free via
    - /proc/sys/vm/nr_hugepages
    - /sys/devices/node/*/hugepages/*/nrhugepages
  - Used via hugetlbfs
- **GB Hugepages 1GB**
  - Prior to RHEL7 - Reserved at boot time/no freeing
  - RHEL7&8 allows runtime allocation & freeing
  - Used via hugetlbfs
- **Transparent HugePages 2MB**
  - On by default via boot args or /sys
  - Used for anonymous memory

2MB standard and 1GB Hugepages

```
# echo 2000 > /proc/sys/vm/nr_hugepages
# cat /proc/meminfo
MemTotal:        16331124 kB
MemFree:         11788608 kB

HugePages_Total:    2000
HugePages_Free:     2000
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB

# ./hugeshm 1000

# cat /proc/meminfo
MemTotal:        16331124 kB
MemFree:         11788608 kB

HugePages_Total:    2000
HugePages_Free:     1000
HugePages_Rsvd:     1000
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

```
hugepagesz=1G, hugepagesz=1G, hugepages=8


# cat /proc/meminfo | grep HugePages
HugePages_Total:       8
HugePages_Free:        8
HugePages_Rsvd:        0
HugePages_Srp:         0


#mount -t hugetlbfs none /mnt
# ./mmapwrite /mnt/junk 33
writing 2097152 pages of random junk to /mnt/junk
wrote 8589934592 bytes to file /mnt/junk


# cat /proc/meminfo | grep
HugePages
HugePages_Total:       8
HugePages_Free:        0
HugePages_Rsvd:        8
HugePages_Srp:         0
```

Transparent Hugepages

```
- Disable transparent_hugepages
    #echo never > /sys/kernel/mm/transparent_hugepages=never
    #time ./memory 15 0
    real   0m12.434s
    user   0m0.936s
    sys    0m11.416s

     # cat /proc/meminfo
     MemTotal:       16331124 kB
     AnonHugePages:  0 kB
```

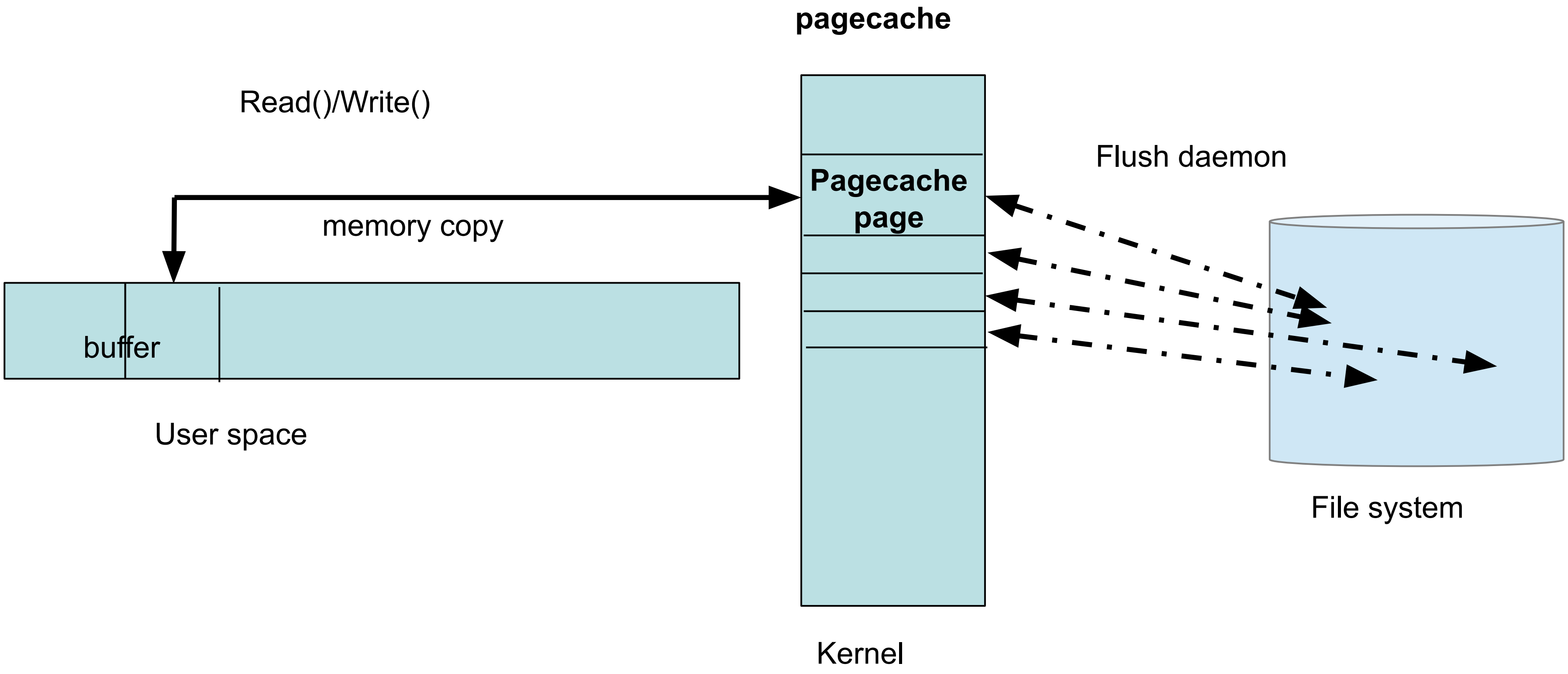
- Boot argument: transparent_hugepages=always (enabled by default)

```
-    #echo always > /sys/kernel/mm/redhat_transparent_hugepage/enabled
     #time ./memory 15GB
     real    0m7.024s
     user    0m0.073s
     sys     0m6.847s

    #cat /proc/meminfo
    MemTotal:       16331124 kB
    AnonHugePages:  15590528 kB
```

SPEEDUP 12.4/7.0 = 1.77x, 56%

RHEL Disk I/O and I/O Elevators

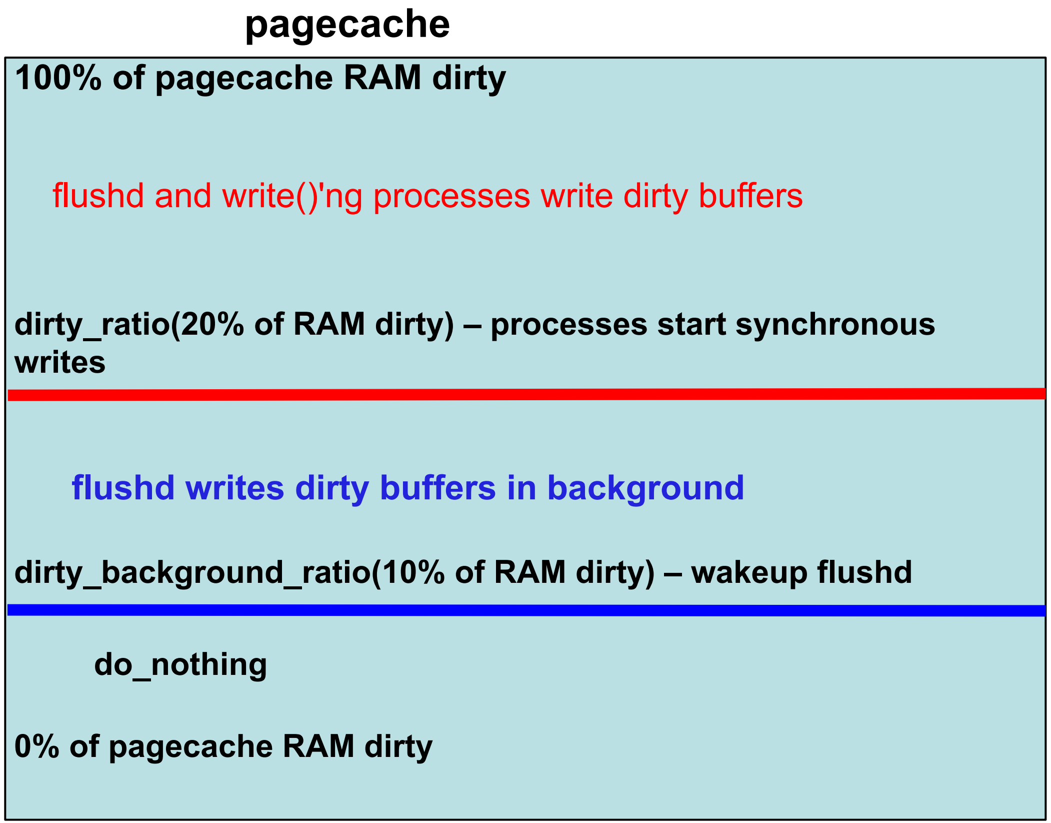
Per file system flush daemon



Virtual Memory Manager (VM) Tunables

- **Reclaim Ratios**
  - /proc/sys/vm/swappiness
  - /proc/sys/vm/vfs_cache_pressure
  - /proc/sys/vm/min_free_kbytes
  -
- **Writeback Parameters**
  - /proc/sys/vm/dirty_background_ratio
  - /proc/sys/vm/dirty_ratio
  -
- **Readahead parameters**
  - /sys/block/<bdev>/queue/read_ahead_kb

dirty_ratio and dirty_background_ratio



pagecache

100% of pagecache RAM dirty

flushd and write()'ng processes write dirty buffers

dirty_ratio(20% of RAM dirty) – processes start synchronous writes

flushd writes dirty buffers in background

dirty_background_ratio(10% of RAM dirty) – wakeup flushd

do_nothing

0% of pagecache RAM dirty

If there is a lot of pagecache pressure one would want to start background flushing sooner and delay the synchronous writes. This can be done by

- **Lowering the dirty_background_ratio**
- **Increasing the dirty_ratio**

On very large memory systems, consider using more granularity by using

- **dirty_background_bytes**
- **dirty_bytes**

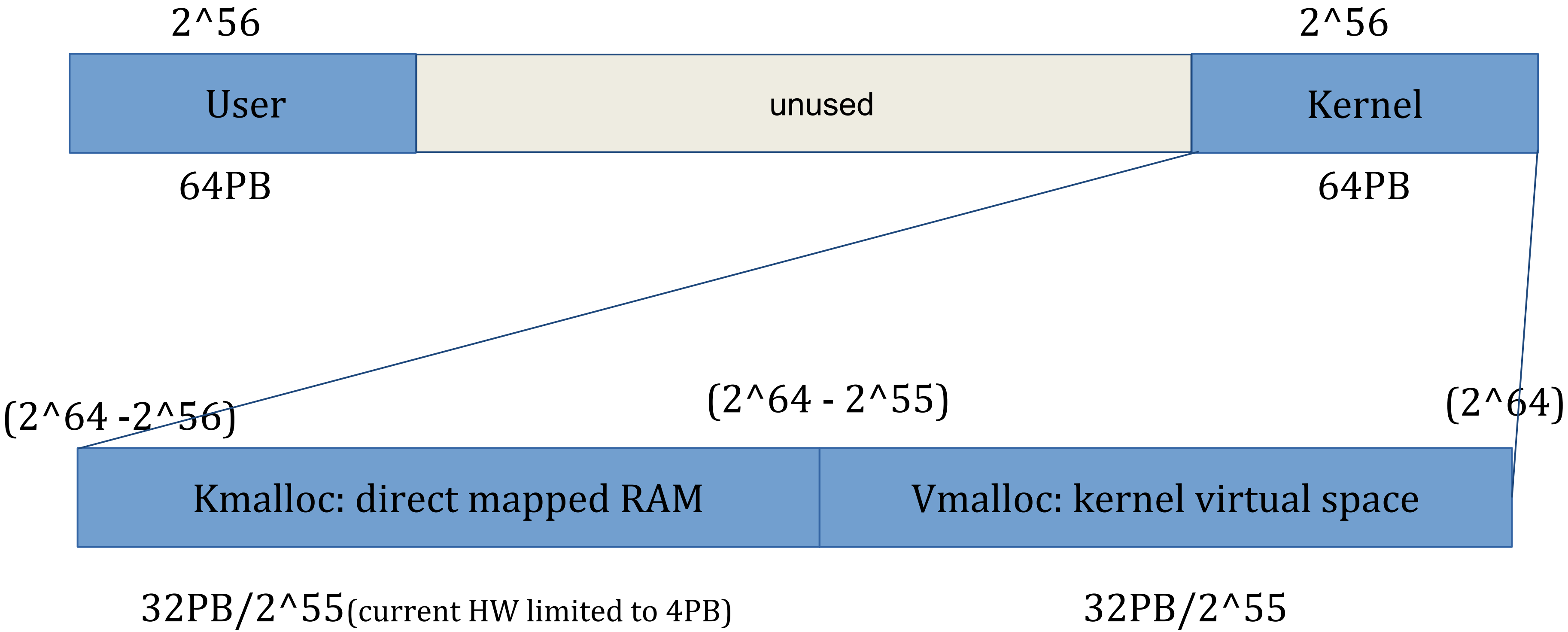
New to RHEL8:

X86_64 5-level page table/57-bit memory support

and

Persistent memory/NvDIMM support

57 bit address space/5-level page tables

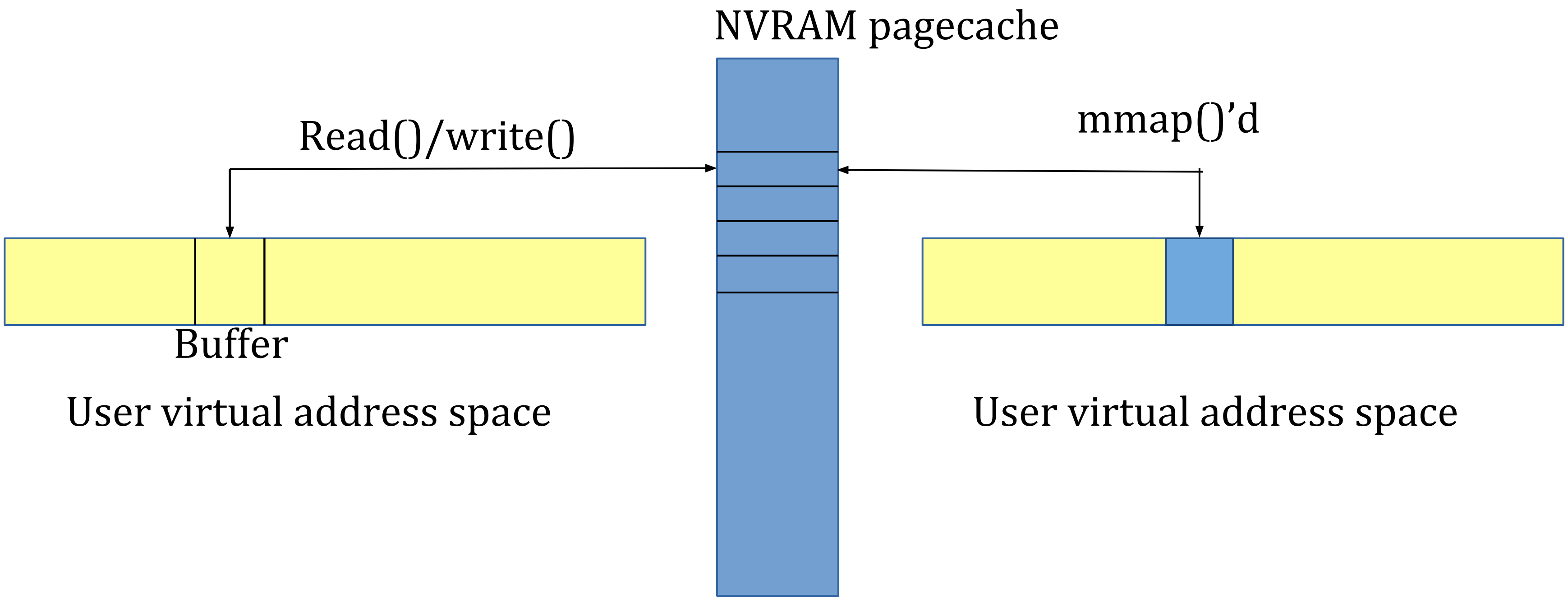


2^56
2^56
User
unused
Kernel
64PB
64PB
(2^64 -2^56)
(2^64 - 2^55)
(2^64)
Kmalloc: direct mapped RAM
Vmalloc: kernel virtual space
32PB/2^55(current HW limited to 4PB)
32PB/2^55

Persistent Memory/NvDIMM Support in RHEL

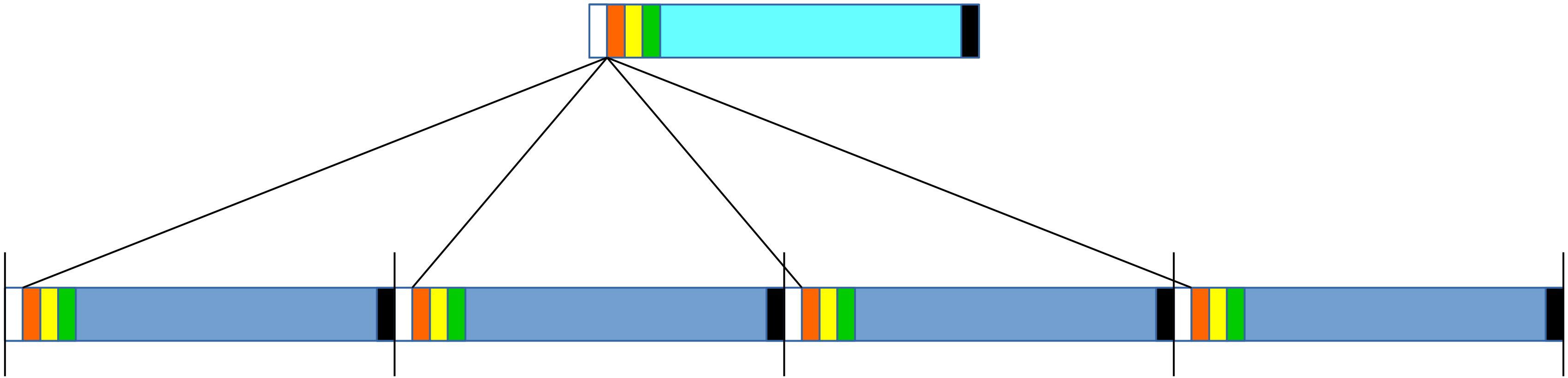
- Persistent memory is non-volatile memory NVDIMMs(aka NVRAM) that can be plugged into the DRAM slots.
  - Can/will be VERY large(need 5-page table support)
- NVRAM can not be accessed via the PCI interface like SSDs.
- NVRAM is accessed via the memory bus, its in the physical address space just like RAM
- NVRAM is primarily used for storage but can be configured as RAM(systems with NVDIMMs must also have DRAM).
  - Choosing if you want the NVDIMMs to be used as storage or RAM is controlled via BIOS settings.
  - In storage mode the DRAM is the system memory and the NVRAM is the storage.
  - In memory mode the NVDIMMs are the system memory and the DRAM is a cache for NVDIMMs.
- DAX – Direct Access File System: allows pages of NVRAM to be mapped directly in the pagecache.
  - Eliminates multiple copies of data
  - Reduces memory demand.
  - Eliminates need for pagecache write-back operations needed for disks and SSDs.

Storage Mode: DAX uses NVRAM for pagecache



Memory Mode: DRAM cache for NVDIMM

DRAM used as cache(direct mapped)



Banks of NVDIMM Memory used as RAM

Summary - RHEL Performance Tech/Tunables

- **RHEL 8 improvements**
    - Multiq SCSI - direct attached and fiberchannel, iozone, fio
    - Network – Netperf/Uperf (TCP/UDP) - improved sm/med packet
    - AIM multiuser (shared, db, fileserver) - lower syscall overhead, VM changes.
    - CVE impacts, use retpoline for spectre Intel (on Skylake vs IBRS)

- **RHEL Performance Tools**
    - **Tuned** - arch specific capable, CascadelakeN, AMD Epyc, ARM
        - Open Shift OCP enhanced for NFV and RT cpu-partioning
    - **AutoNUMA -** improved for BM, KVM and container workloads
        - SPECjbb multi-instance, CNV
    - **HugePages**
        - Control w/ tuned, wired-down, THP for VM/pods, DB/Java 2MB or 1GB
    - **Top Tools**
        - op, *stat, PCP, Perf (c-2-c), tuna, Pbench (new consult w/ SAs)

Red Hat Performance Whitepapers

- Red Hat Performance Tuning Guide
- Red Hat Low Latency Tuning Guide
- Red Hat Virtualization Tuning Guide
- RHEL Blog / Developer Blog

RHEL tuned parameters that affect performance (sysctls)

CPU Scheduler tunables

Throughput Performance
Scheduler quantum (default 4/10 ms,-> 10/15 ms)

- kernel.sched_min_granularity_ns=10000000
- kernel_sched_wakeup _granularity_ns = 15000000

Weight function on how often to migrate - 5ms -> 50ms

- kernel.sched_migration_cost_ns=50000000

Latency Performance tuning

- Decrease quantum above to 4 /10 ms

Adjust power management - BIOS OS controlled

- pstates - governor=performance
- energy_perf_bias=performance
- cstate - force_latency=1

Disable scaning tools for better determinism

- Disable numa balance
  - kernel.numa_balancing = 0
- Disable Transparent HugePages
  - mm.redhat_transparent_hugepage never

VM Tunables

Reclaim Ratios

- vm.swappiness
- vm.vfs_cache_pressure
- vm.min_free_kbytes

Writeback Parameters 30/10 -> 10/3

- vm.dirty_background_ratio
- vm.dirty_ratio

Readahead parameters per device 512-> 4k

- /sys/block/<bdev>/queue/read_ahead_kb

Non-Uniform Memory Access (NUMA) Hugepages

Auto numa balancing at scheduling time

- kernel.numa_balancing = 1
- Adjust numa scan interval 1000 ms -> 100 ms
- vm.zone_reclaim_mode = 1 (reclaim local node vs spill)

Transparent HugePages

- mm.redhat_transparent_hugepage enabled

RHEL8 eBPF Tech preview Denial Of Service (DoS)

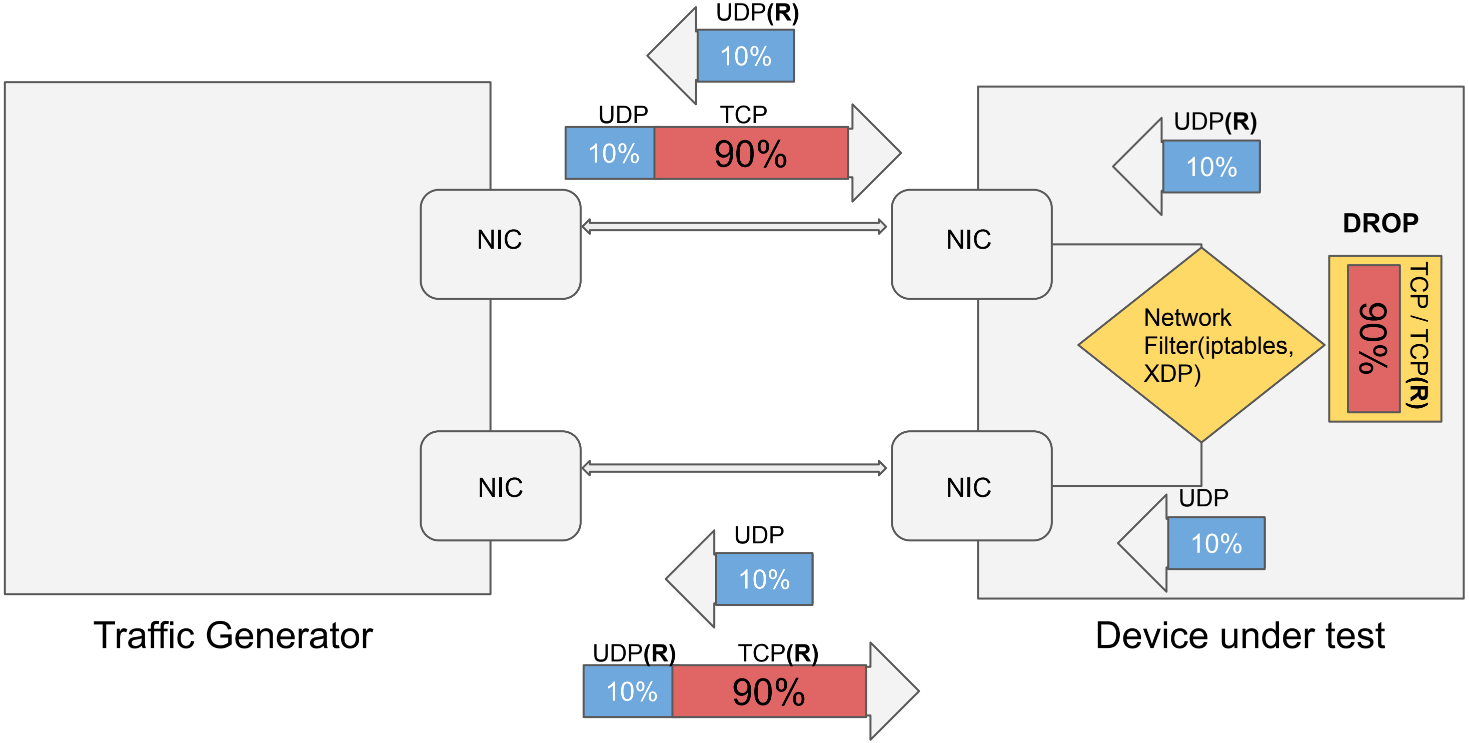
- The traffic flow is unidirectional from both interfaces.
- The packets are routed between the two DUT interfaces using kernel routing table and forwarded to the other traffic generator port respectively.
- A binary search is done to find the max packet rate till the test passes.
- The test is passed when:
  - **No TCP packet is received on both interfaces**
  - **0.002% of UDP packets drop threshold is maintained.**

- Iptables filter and drops TCP port 80 packets:
  - Rules are added once in **filter** table and then in **raw** table for performance comparison
- For XDP, we are using xdp_ddos_blacklist[1] program which is loaded on both DUT interfaces and drops packets arriving on TCP port 80.

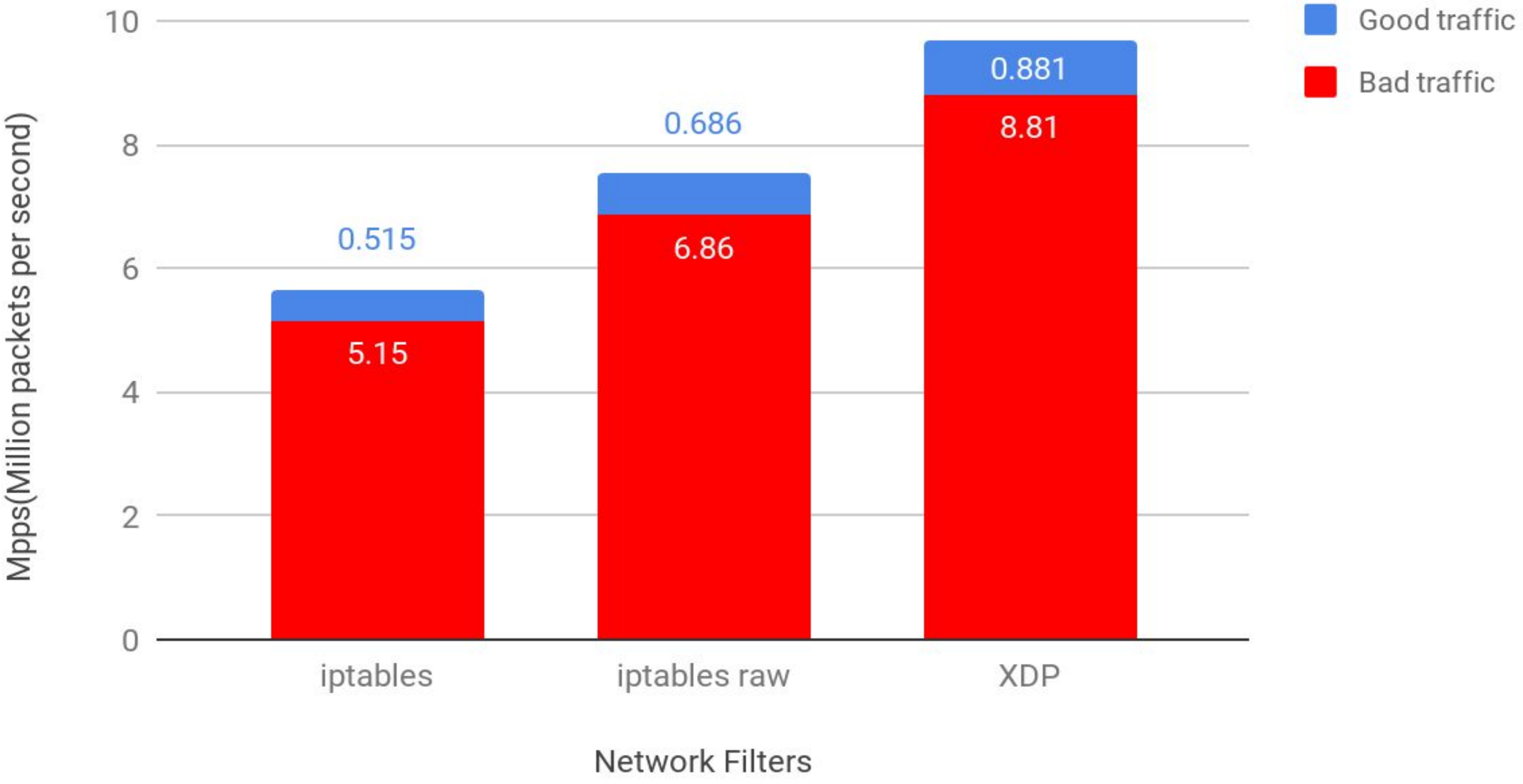
[1]: https://github.com/netoptimizer/prototype-kernel/blob/master/kernel/samples/bpf/xdp_ddos01_blacklist_kern.c

Test setup

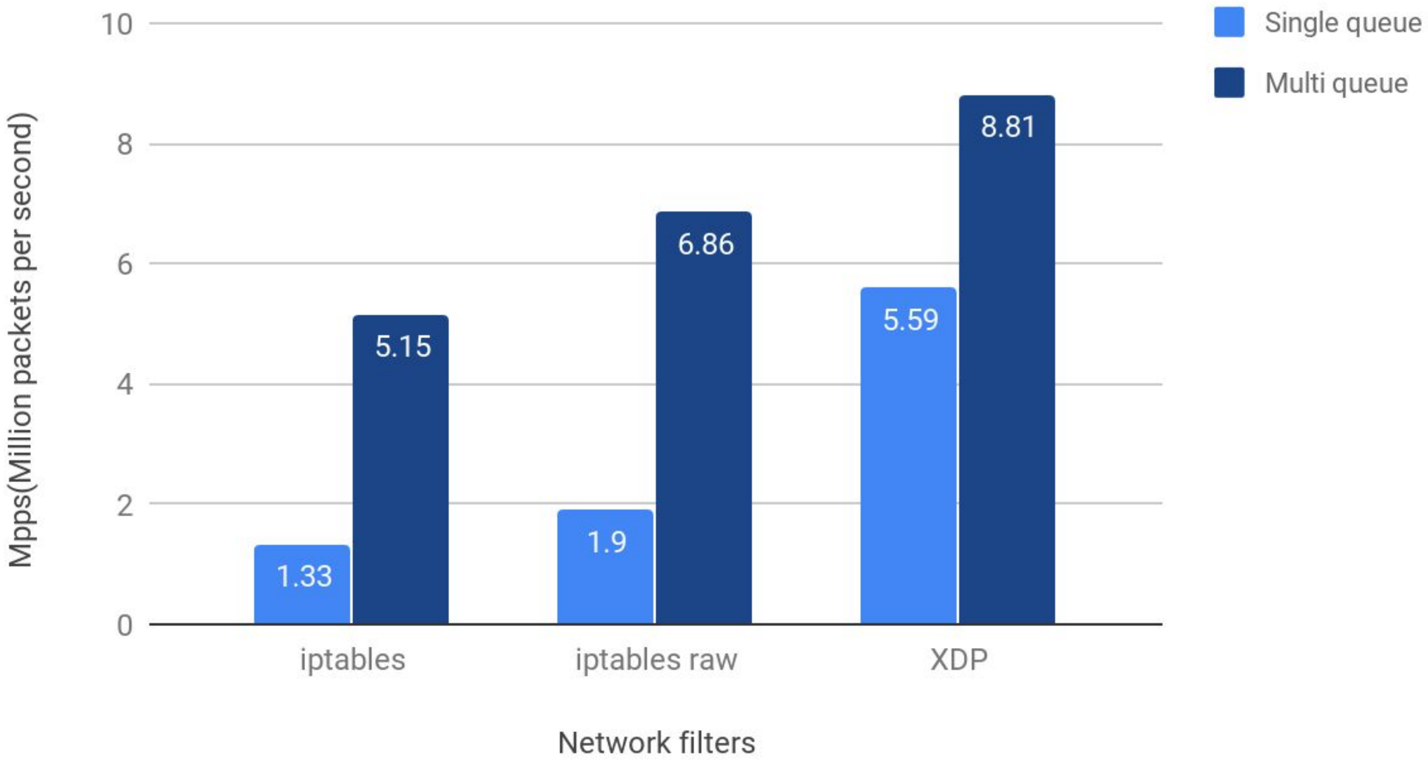
*** (R): Reverse Direction**



DDoS scenario(Ratio of bad to good traffic is 9:1)



DDoS scenario(Single vs Multi Queue)



perf c2c for cpu cacheline false sharing detection

Works on R8

Critical for:

- Shared memory applications
- Multi-threaded apps spanning multiple numa nodes

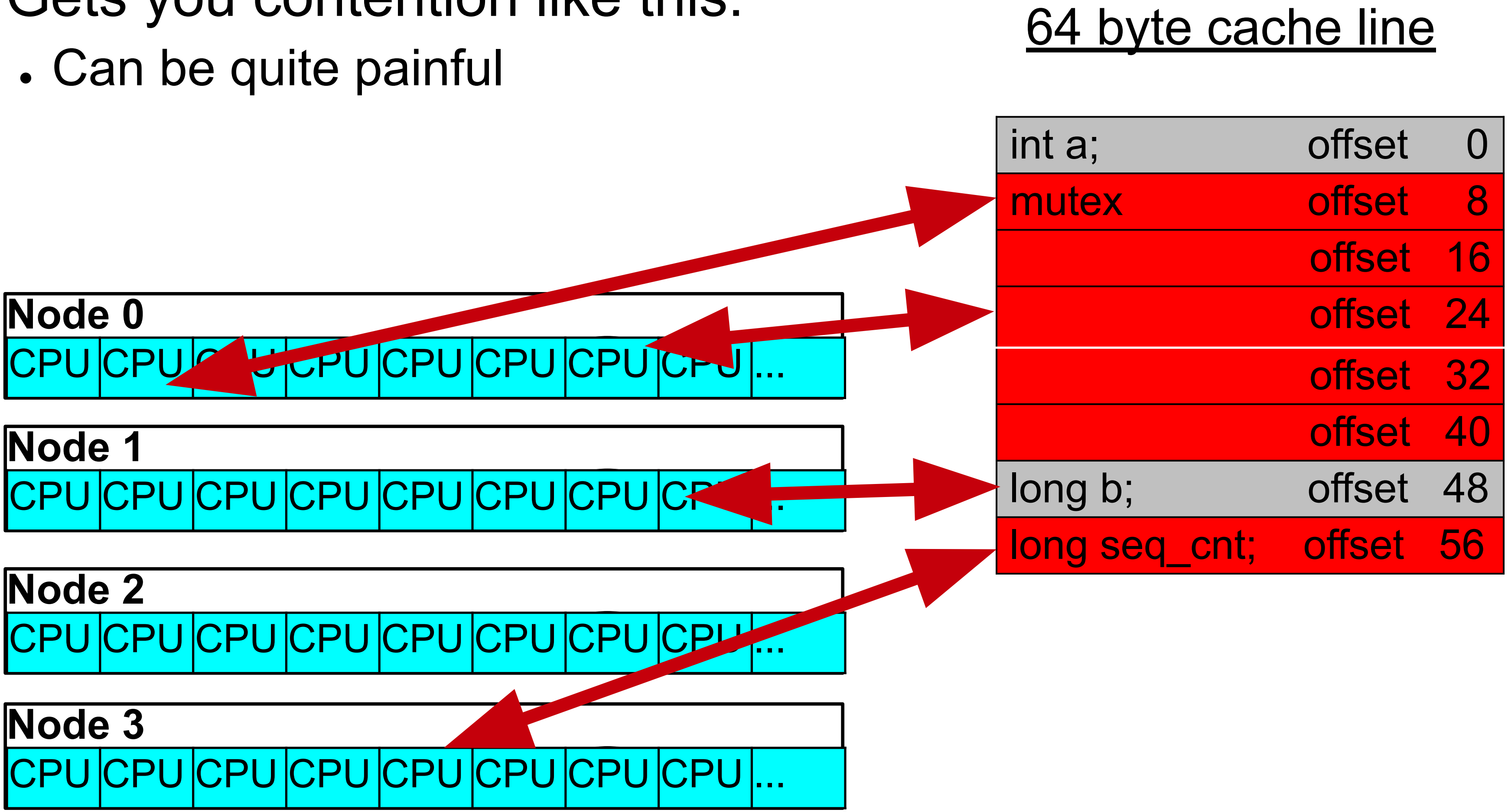
Shows everything needed to find false sharing:

- All readers and writers contending for hottest cachelines.
- The cpus and nodes they executed on.
- Process names, data addr, ip, pids, tids, src file and line number.
- Where hot variables are sharing cachelines, (like locks).
- Where hot structs are spanning cachelines, (like an unaligned mutex).

Detailed blog: https://joemario.github.io/blog/2016/09/01/c2c-blog/

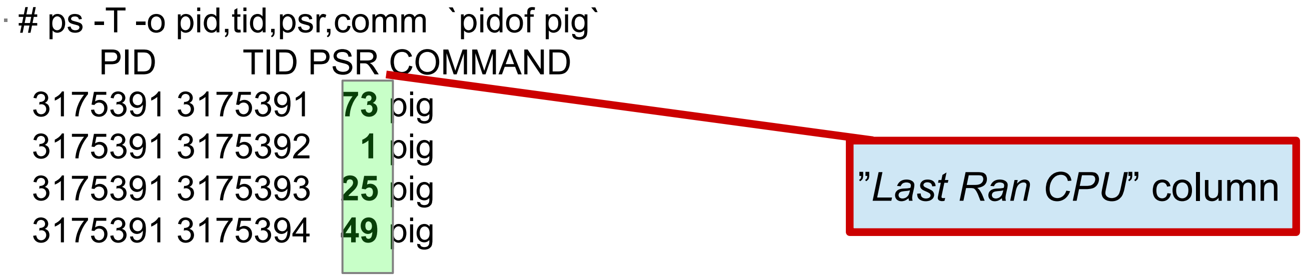
Gets you contention like this:

- Can be quite painful



Where are my processes and threads running?
Two ways to see “where it last ran”.

1) *ps -T -o pid,tid,psr,comm <pid>*



```
· # ps -T -o pid,tid,psr,comm  `pidof pig`
        PID        TID PSR COMMAND
  3175391 3175391   73 pig
  3175391 3175392    1 pig
  3175391 3175393   25 pig
  3175391 3175394   49 pig
```

”*Last Ran CPU*” column

2) *Run “top”, then enter “f”, then select “Last used cpu” field*

Are my threads and data aligned on same numa node?

Use perf (soon to report node & phys addr info where data resides)

perf mem record -- --sample-cpu foo_exe
perf mem report -F mem,cpu,dcacheline,snoop,symbol -s dcacheline --stdio

Tuna: command line or gui

Fine grained process view & control

- Adjust scheduler tunables, (sched policy, RT priority and CPU affinity)
- See results instantly
- Tune threads and IRQ handlers.
- Isolate CPU cores and sockets,

Examples:

Move an irq to cpu 5

```
# tuna -c5 -q eth4-rx-4 –move
```

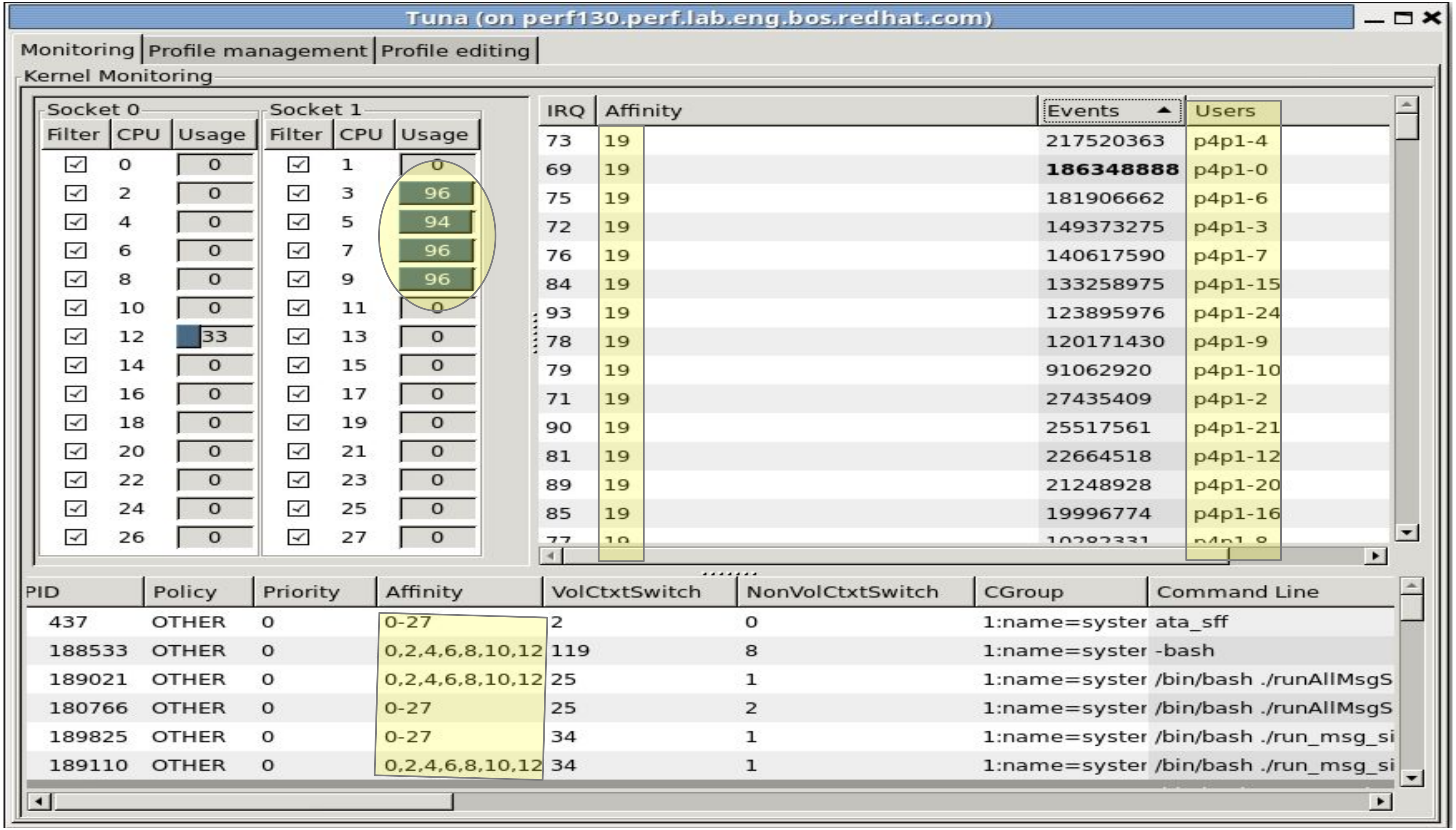
Move all irqs named “eth4*” away from numa node 1

```
# tuna -S 1 -i -q ‘eth4*’
```

Move all rcu kernel threads to cpus 1 and 3

```
# tuna -c1,3 -t ‘*rcu*’ --move
```

Tuna example



Socket 0

Filter	CPU	Usage
☑	0	0
☑	2	0
☑	4	0
☑	6	0
☑	8	0
☑	10	0
☑	12	33
☑	14	0
☑	16	0
☑	18	0
☑	20	0
☑	22	0
☑	24	0
☑	26	0

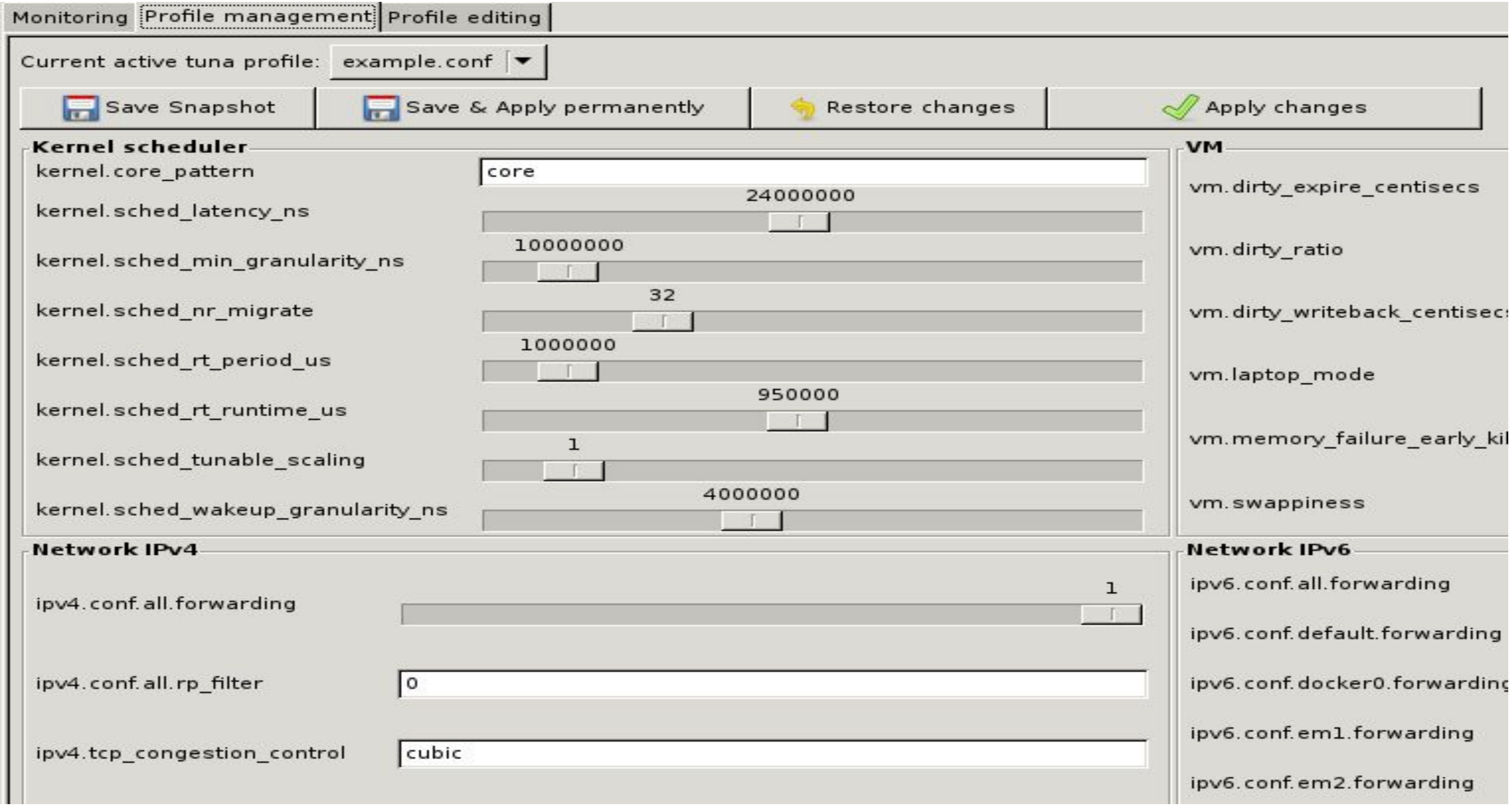
Socket 1

Filter	CPU	Usage
☑	1	0
☑	3	96
☑	5	94
☑	7	96
☑	9	96
☑	11	0
☑	13	0
☑	15	0
☑	17	0
☑	19	0
☑	21	0
☑	23	0
☑	25	0
☑	27	0

IRQ	Affinity	Events ▲	Users
73	19	217520363	p4p1-4
69	19	**186348888**	p4p1-0
75	19	181906662	p4p1-6
72	19	149373275	p4p1-3
76	19	140617590	p4p1-7
84	19	133258975	p4p1-15
93	19	123895976	p4p1-24
78	19	120171430	p4p1-9
79	19	91062920	p4p1-10
71	19	27435409	p4p1-2
90	19	25517561	p4p1-21
81	19	22664518	p4p1-12
89	19	21248928	p4p1-20
85	19	19996774	p4p1-16
77	19	10282331	p4p1-8

PID	Policy	Priority	Affinity	VolCtxtSwitch	NonVolCtxtSwitch	CGroup	Command Line
437	OTHER	0	0-27	2	0	1:name=syster	ata_sff
188533	OTHER	0	0,2,4,6,8,10,12	119	8	1:name=syster	-bash
189021	OTHER	0	0,2,4,6,8,10,12	25	1	1:name=syster	/bin/bash ./runAllMsgS
180766	OTHER	0	0-27	25	2	1:name=syster	/bin/bash ./runAllMsgS
189825	OTHER	0	0-27	34	1	1:name=syster	/bin/bash ./run_msg_si
189110	OTHER	0	0,2,4,6,8,10,12	34	1	1:name=syster	/bin/bash ./run_msg_si

Tuna GUI Capabilities Updated for RHEL7



Current active tuna profile: example.conf

Kernel scheduler

kernel.core_pattern: core
kernel.sched_latency_ns: 24000000
kernel.sched_min_granularity_ns: 10000000
kernel.sched_nr_migrate: 32
kernel.sched_rt_period_us: 1000000
kernel.sched_rt_runtime_us: 950000
kernel.sched_tunable_scaling: 1
kernel.sched_wakeup_granularity_ns: 4000000

VM

vm.dirty_expire_centisecs
vm.dirty_ratio
vm.dirty_writeback_centisec
vm.laptop_mode
vm.memory_failure_early_kil
vm.swappiness

Network IPv4

ipv4.conf.all.forwarding: 1
ipv4.conf.all.rp_filter: 0
ipv4.tcp_congestion_control: cubic

Network IPv6

ipv6.conf.all.forwarding
ipv6.conf.default.forwarding
ipv6.conf.docker0.forwarding
ipv6.conf.em1.forwarding
ipv6.conf.em2.forwarding

CVE Performance overrides

To disable CVE on RHEL-{6,7,8}, add the following to the boot grub line

spectre_v2=off spec_store_bypass_disable=off nopti l1tf=off mds=off

(New to RHEL7.7 and 8.1 - add mitigations=off to disable all, experiment only)

Your resulting vulnerabilities files should then look something like these:

```
# grep . /sys/devices/system/cpu/vulnerabilities/*
    /sys/devices/system/cpu/vulnerabilities/l1tf:Mitigation: PTE Inversion; VMX: vulnerable
    /sys/devices/system/cpu/vulnerabilities/meltdown:Vulnerable
    /sys/devices/system/cpu/vulnerabilities/spec_store_bypass:Vulnerable
    /sys/devices/system/cpu/vulnerabilities/spectre_v1:Mitigation: __user pointer sanitization
    /sys/devices/system/cpu/vulnerabilities/spectre_v2:Vulnerable, IBPB: disabled, STIBP: disabled
```

CVE Performance Defaults w/ SkyLake

```
# grep . /sys/devices/system/cpu/vulnerabilities/*
/sys/devices/system/cpu/vulnerabilities/l1tf:Mitigation: PTE
Inversion; VMX: conditional cache flushes, SMT vulnerable

/sys/devices/system/cpu/vulnerabilities/meltdown:Mitigation: PTI

/sys/devices/system/cpu/vulnerabilities/spec_store_bypass:Mitigation:
Speculative Store Bypass disabled via prctl and seccomp

/sys/devices/system/cpu/vulnerabilities/spectre_v1:Mitigation: __user
pointer sanitization

/sys/devices/system/cpu/vulnerabilities/spectre_v2:Mitigation: Full
generic retpoline, IBPB: conditional, IBRS_FW, STIBP: conditional,
RSB filling
```

RED HAT
SUMMIT

THANK YOU

- plus.google.com/+RedHat
- linkedin.com/company/red-hat
- youtube.com/user/RedHatVideos
- facebook.com/redhatinc
- twitter.com/redhat