Performance Analysis and Tuning – Part I

Containers are Linux, run/optimized and tuned just like Linux.

D. John Shakshober (Shak) – Tech Director Performance Engineering

Larry Woodman - Senior Consulting Engineer

Bill Gray – Senior Principal Performance Engineer

Joe Mario - Senior Principal Performance Engineer
Agenda: Performance Analysis Tuning Part I

• Part I - Containers are Linux, run/optimized and tuned just like Linux.
  • RHEL Evolution 5->6->7 – Hybrid Clouds Atomic / OSE / RHOP
  • System Performance/Tools Tuned profiles
  • NonUniform Memory Access (NUMA)
    • What is NUMA, RHEL Architecture, Auto-NUMA-Balance
• Network Performance – noHZ, Throughput vs Latency-performance
  • Tuna – IRQ pinning, alter priorities, monitor
  • NFV w/ DPDK fastdata path
  • Perf advanced features, BW monitoring, Cache-line tears C-2-C
• “Performance + Scale Experts” - 205C - 5:30-7 PM
• Free - Soda/Beer/Wine
Red Hat Enterprise Linux Performance Evolution

- **RHEL5**
  - 1000 Hz, CFQ IO elevator, ktune to change to deadline
  - Numactl, taskset affinity, static hugepages, IRQ balance, oprofile

- **RHEL 6**
  - Tickless scheduler CFS, islocpus, userspace NUMAD tool
  - Transparent hugepages (THP), numa-IRQ balance, cGroups
  - Tuna, Perf, PCP ship w/ OS

- **RHEL 7**
  - NoHZ_full for CFQ, islocpu, realtime ship same RHEL7 kernel
  - AutoNuma balance, THP, systemd – Atomic containers
RHEL Performance Workload Coverage

(bare metal, KVM virt w/ RHEV and/or OSP, LXC Kube/OSE and Industry Standard Benchmarks)

Benchmarks – code path coverage

- CPU – linpack, Imbench
- Memory – Imbench, McCalpin STREAM
- Disk IO – iozone, fio – SCSI, FC, iSCSI
- Filesystems – iozone, ext3/4, xfs, gfs2, gluster
- Networks – netperf – 10/40Gbit, Infiniband/RoCE, Bypass
- Bare Metal, RHEL6/7 KVM, Atomic Containers
- White box AMD/Intel, with our OEM partners

Application Performance

- Linpack MPI, HPC workloads
- AIM 7 – shared, filesystem, db, compute
- Database: DB2, Oracle 11/12, Sybase 15.x, MySQL, MariaDB, Postgrs, MongoDB
- OLTP – TPC-C, TPC-VMS
- DSS – TPC-H/xDS
- Big Data – TPCx-HS, Bigbench
- SPEC cpu, jbb, sfs, virt, cloud
- SAP – SLCS, SD, Hana
- STAC = FSI (STAC-N)
- SAS mixed Analytic, SAS grid (gfs2)
RHEL / Intel Benchmarks Broadwell EP/EX

Red Hat delivers high-performance on critical enterprise workloads with the latest Intel Xeon E7 v4 processor family.

Benchmark publications using Red Hat Enterprise Linux over past 24 months

Industry Benchmarks June 2016

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Percent Using Red Hat Enterprise Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPCx-BB</td>
<td>100%</td>
</tr>
<tr>
<td>TPCx-HS</td>
<td>89%</td>
</tr>
<tr>
<td>SPEC CPU2006</td>
<td>81%</td>
</tr>
<tr>
<td>SPECjbb2015</td>
<td>67%</td>
</tr>
<tr>
<td>SPECvirt_sc2013</td>
<td>63%</td>
</tr>
<tr>
<td>SPEC OMP2012</td>
<td>39%</td>
</tr>
</tbody>
</table>
Key performance takeaways

- SQL Server 2017 on Red Hat Enterprise Linux surpasses the previous #1 TPC-H@1000GB result achieved with SQL Server 2016
  - 6% higher performance
  - 5% better price/performance
- The first and only result with Microsoft SQL Server 2017 Enterprise Edition
- Results achieved on similarly configured servers with two Intel® Xeon® E5-2699 v4 processors

Higher performance

- HPE ProLiant DL380 Gen9 w/SQL Server 2016 on Microsoft Windows: 678,492 QphH

Better price/performance

- HPE ProLiant DL380 Gen9 w/SQL Server 2016 on Microsoft Windows: $0.64/QphH
  - System availability as of 07-31-2016 (results published 03-24-2016)
- HPE ProLiant DL380 Gen9 w/SQL Server 2017 on Red Hat Enterprise Linux: $0.61/QphH

Read the performance brief at hpe.com/servers/benchmarks.

© Copyright 2017 Hewlett Packard Enterprise Development LP. Microsoft and Windows are U.S. registered trademarks of Microsoft Corporation. Red Hat, Red Hat Enterprise Linux, and the Shadowman logo are registered trademarks of Red Hat, Inc. Linux is a registered trademark of Linus Torvalds. Intel and Xeon are trademarks of Intel Corporation in the U.S. and other countries. TPC and TPC-H are trademarks of the Transaction Processing Performance Council. TPC-H results show the HPE ProLiant DL380 Gen9 w/SQL Server 2016 on Microsoft Windows with a result of 717,101 QphH @ 1000GB with system availability as of 10-19-2017 (results published 04-19-2017), the HPE ProLiant DL380 Gen9 w/SQL Server 2017 on Red Hat Enterprise Linux with a result of 678,492 QphH @ 1000GB and $0.64/QphH with system availability as of 07-31-2016 (results published 03-24-2016). The TPC believes that comparisons of TPC-H results published with different scale factors are misleading and discourages such comparisons. Please see tpc.org for up-to-date information. Competitive claims valid as of 04-19-2017.
Pointers – Benchmarks / Partner Results

• SPEC – Systems Performance Evaluation Committee
  • http://www.spec.org
• TPC – Transaction Processing Council
  • http://www.tpc.org
• STAC – Security Technology Analysis Center
  • https://stacresearch.com/
• HP
  • http://hpe.com/servers/benchmarks New World Record RH/SQLserver2017
Performance Tools - Tuned
tuned is a tool to dynamically tune Red Hat Enterprise Linux.

You could improve workload performance by applying one of the predefined profiles or use those that you’ve written yourself.
Tuned: Your Custom Profiles

Parents
- throughput-performance
- balanced
- latency-performance

Children
- network-throughput
- desktop
- network-latency
- virtual-host
- virtual-guest

Children/Grandchildren
- Your Web Profile
- Your Database Profile
- Your Middleware Profile
Performance Metrics - Latency == Speed - Throughput == Bandwidth

**Latency – Speed Limit**
- Ghz of CPU, Memory PCI
- Small transfers, disable aggregation – TCP nodelay
- Dataplane optimization DPDK

**Throughput – Bandwidth** - # lanes in Highway
- Width of data path / cachelines
- Bus Bandwidth, QPI links, PCI 1-2-3
- Network 1 / 10 / 40 Gb – aggregation, NAPI
- Fiberchannel 4/8/16, SSD, NVME Drivers
Tuned Profile Examples

**throughput-performance**

- governor=performance
- energy_perf_bias=performance
- min_perf_pct=100
- transparent_hugepages=always
- readahead=>4096
- sched_min_granularity_ns = 10000000
- sched_wakeup_granularity_ns = 15000000
- vm.dirty_ratio = 40
- vm.dirty_background_ratio = 10
- vm.swappiness=10

**latency-performance**

- force_latency=1
- governor=performance
- energy_perf_bias=performance
- min_perf_pct=100
- kernel.sched_min_granularity_ns=10000000
- vm.dirty_ratio=10
- vm.dirty_background_ratio=3
- vm.swappiness=10
- kernel.sched_migration_cost_ns=5000000
Tuned: Storage Performance Boost: throughput-performance (default in RHEL7)

64KB Read Untuned vs "Tuned" throughput-performance

Larger is better

Throughput (IOPS)

Untuned               Tuned

0                   5000
1250               3750
2500               5000

64KB Read
Map tuned profiles to Red Hat's product portfolio:

<table>
<thead>
<tr>
<th>RHEL Desktop/Workstation</th>
<th>balanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>RHV Host, Guest</td>
<td>virtual-host/guest</td>
</tr>
<tr>
<td>Red Hat Storage</td>
<td>rhs-high-throughput, virt</td>
</tr>
<tr>
<td>RHEL Atomic</td>
<td>atomic-host, atomic-guest</td>
</tr>
<tr>
<td>RHEL Server/HPC</td>
<td>throughput-performance</td>
</tr>
<tr>
<td>RHV</td>
<td>virtual-host</td>
</tr>
<tr>
<td>RHOSP (compute node)</td>
<td>virtual-host</td>
</tr>
<tr>
<td>OpenShift</td>
<td>openshift-master, node</td>
</tr>
<tr>
<td>RHEL for Real Time</td>
<td>realtime</td>
</tr>
<tr>
<td>RHEL for Real Time KVM/NFV</td>
<td>realtime-virtual-host/guest</td>
</tr>
<tr>
<td>RHEL + SAP</td>
<td>sap / sap-hana</td>
</tr>
<tr>
<td>RHOP - NFV (compute node)</td>
<td>cpu-partitioning</td>
</tr>
</tbody>
</table>

New in 7.4
RHEL 6/7 Non-Uniform Memory Access (NUMA)
Typical Four-Node NUMA System

Node 0

Node 0 RAM

Core 0
Core 2
Core 4
Core 6
Core 8
Core...

L3 Cache

Core 1
Core 3
Core 5
Core 7
Core 9

QPI links, IO, etc.

Node 1

Node 1 RAM

Core 0
Core 2
Core 4
Core 6
Core 8
Core...

L3 Cache

Core 1
Core 3
Core 5
Core 7
Core 9

QPI links, IO, etc.

Node 2

Node 2 RAM

Core 0
Core 2
Core 4
Core 6
Core 8
Core...

L3 Cache

Core 1
Core 3
Core 5
Core 7
Core 9

QPI links, IO, etc.

Node 3

Node 3 RAM

Core 0
Core 2
Core 4
Core 6
Core 8
Core...

L3 Cache

Core 1
Core 3
Core 5
Core 7
Core 9

QPI links, IO, etc.
NUMA Especially Important When...

• Server consolidation / replicated processes / virtual guests / containers
  • Multiple processes (re)using mostly local data
  • Multiple workloads / threads consuming fractional subsets of system resources
• Resource access patterns can be private, localized or contained
• Ideally, these workloads / threads can be sized to fit within NUMA nodes!
  • Align CPUs, Memory, Devices, and Interrupts for workloads that can be localized to minimize latency, and isolated to avoid interference!
• System-wide monolithic processes with poor data locality are different...
Want to align process memory and CPU threads within NUMA nodes

No NUMA management

With NUMA management
Non-optimal numa setup

Process 1 in red, 5 threads
Optimal numa setup

Process 1 in green, 4 threads
Process 2 in red, 5 threads

Numa node 0

Numa node 1
Are my processes doing that?

• Variety of commands available to help:
  • lscpu
  • numactl
  • lstopo
  • numastat
  • ps
  • top
Tools to display CPU and Memory (NUMA)

# lscpu

Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                40
On-line CPU(s) list:   0-39
Thread(s) per core:    1
Core(s) per socket:    10
CPU socket(s):         4
NUMA node(s):          4

<table>
<thead>
<tr>
<th>NUMA node0 CPU(s):</th>
<th>NUMA node1 CPU(s):</th>
<th>NUMA node2 CPU(s):</th>
<th>NUMA node3 CPU(s):</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 4, 8, 12, 16, 20, 24, 28, 32, 36</td>
<td>2, 6, 10, 14, 18, 22, 26, 30, 34, 38</td>
<td>1, 5, 9, 13, 17, 21, 25, 29, 33, 37</td>
<td>3, 7, 11, 15, 19, 23, 27, 31, 35, 39</td>
</tr>
</tbody>
</table>

The cpu numbers for each node

cpu, core, socket, node info
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:

```
<table>
<thead>
<tr>
<th>node</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
<td>10</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>21</td>
<td>10</td>
<td>21</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>21</td>
<td>21</td>
<td>10</td>
</tr>
</tbody>
</table>
```

Relative “node-to-node” latency costs.
How can I visualize my system's NUMA topology in Red Hat Enterprise Linux?

https://access.redhat.com/site/solutions/62879
Numastat Shows Process Memory Locations

• Enhanced by Red Hat (since Red Hat Enterprise Linux 6.4) with helpful and informative new memory display features.

• Numastat shows per-NUMA-node memory statistics for processes and the operating system.

• By default, numastat displays per-node kernel memory allocator hit and miss statistics.

• Any command line arguments to numastat will invoke enhanced behavior to show per-node distribution of memory.

• Typical usage: “numastat -cm <workload>"
numastat shows need for NUMA management

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

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

```bash
# numastat -c qemu

Per-node process memory usage (in Mbs)

<table>
<thead>
<tr>
<th>PID</th>
<th>Node 0</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>10587 (qemu-kvm)</td>
<td>0</td>
<td>10723</td>
<td>5</td>
<td>0</td>
<td>10728</td>
</tr>
<tr>
<td>10629 (qemu-kvm)</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>10717</td>
<td>10722</td>
</tr>
<tr>
<td>10671 (qemu-kvm)</td>
<td>0</td>
<td>0</td>
<td>10726</td>
<td>0</td>
<td>10726</td>
</tr>
<tr>
<td>10713 (qemu-kvm)</td>
<td>10733</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>10738</td>
</tr>
<tr>
<td>Total</td>
<td>10733</td>
<td>10723</td>
<td>10740</td>
<td>10717</td>
<td>42913</td>
</tr>
</tbody>
</table>
```
What about my processes and threads?
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`

<table>
<thead>
<tr>
<th>PID</th>
<th>TID</th>
<th>PSR</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>3175391</td>
<td>3175391</td>
<td>73</td>
<td>pig</td>
</tr>
<tr>
<td>3175391</td>
<td>3175392</td>
<td>1</td>
<td>pig</td>
</tr>
<tr>
<td>3175391</td>
<td>3175393</td>
<td>25</td>
<td>pig</td>
</tr>
<tr>
<td>3175391</td>
<td>3175394</td>
<td>49</td>
<td>pig</td>
</tr>
<tr>
<td>3175391</td>
<td>3175395</td>
<td>74</td>
<td>pig</td>
</tr>
<tr>
<td>3175391</td>
<td>3175396</td>
<td>2</td>
<td>pig</td>
</tr>
<tr>
<td>3175391</td>
<td>3175397</td>
<td>26</td>
<td>pig</td>
</tr>
<tr>
<td>3175391</td>
<td>3175398</td>
<td>50</td>
<td>pig</td>
</tr>
<tr>
<td>3175391</td>
<td>3175399</td>
<td>75</td>
<td>pig</td>
</tr>
<tr>
<td>3175391</td>
<td>3175400</td>
<td>3</td>
<td>pig</td>
</tr>
</tbody>
</table>
```

"Last Ran CPU" column

2) Run “`top`”, then enter “f”, then select “Last used cpu” field
Tips for Good NUMA Performance

• Never disable NUMA in the BIOS. Keep BIOS interleaved memory OFF (which should be the system BIOS default)
  • Else OS will see only 1-NUMA node!
• Understand your system hardware NUMA topology, and basic operation and implications of NUMA
  • (e.g. per-node resources, and zone_reclaim_mode setting)
• Know your workload resource consumption. If possible, size parallel jobs to fit entirely in NUMA nodes.
• Use appropriate tuning if necessary to control placement.
zone_reclaim_mode

• Controls NUMA specific memory allocation policy
• To see current setting: cat /proc/sys/vm/zone_reclaim_mode
  • Turn ON: echo 1 > /proc/sys/vm/zone_reclaim_mode
    • Reclaim memory from local node rather than allocating from next node
  • Turn OFF: echo 0 > /proc/sys/vm/zone_reclaim_mode
    • Allocate from all nodes before reclaiming memory
• Default is set at boot time based on NUMA factor
• In Red Hat Enterprise Linux 6.6+ and 7+, the default is usually OFF – because this is better for many applications
Low-memory SPEC CPU loses huge performance with wrong zone reclaim mode setting! Several benchmarks off more than 40%.

(BTW, Don't run SPEC CPU with low memory!!)
Techniques to control placement:

**numactl:**
- Control NUMA policy for processes or shared memory:

**taskset:**
- Retrieve or set a process's CPU affinity

**sched_getaffinity(), sched_setaffinity()**
- for process affinity from within program

**mbind(), get_mempolicy(), set_mempolicy()**
- set default NUMA memory policy for a process children.
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
cpupbind: 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
cpupbind: 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
cpupbind: 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
cpupbind: 4 5 6 7
nodebind: 4 5 6 7
membind: 0 1 2 3 4 5 6 7
```
Techniques to control placement (cont):

numad:
• User-mode daemon.
• Attempts to locate processes for efficient NUMA locality and affinity.
• Dynamically adjusting to changing system conditions.
• Available in RHEL 6 & 7.

Auto-Numa-Balance kernel scheduler:
• Automatically run programs near their memory, and moves memory near the programs using it.
• Default enabled. Available in RHEL 7+
• Great video on how it works:
  • https://www.youtube.com/watch?v=mjVw_oe1hEA
NUMA Alignment Makes SPECjbb2005 2x Faster

Multi-instance Java Workload

- No NUMA
- Autonuma (Default)
- Numad
- Numactl
NUMA tuning for KVM / Containers is the same!

- Best performance is achieved if the size of the guest/container can fit into a single NUMA node.
  - In RHEL7, auto-numa kernel scheduler will try to move guest to one node.

- Great doc with numerous examples: See the NUMA chapter in: Red Hat Virtualization Tuning and Optimization Guide
RHEL Network Performance
Network Tuned Profiles

**For throughput**

- **Parents**
  - throughput-performance

- **Children**
  - network-throughput

**Network-throughput:**
- Inherits system tunables for throughput.
- Bumps network tunables for increased network throughput.

**For latency**

- **Parents**
  - latency-performance

- **Children**
  - network-latency

**Network-latency:**
- Inherits system tunables for latency
- Bumps network tunables for latency
- Disables:
  - transparent hugepages
  - kernel numa balancing.
Network Performance Tuning
Red Hat Enterprise Linux 7

• Tactical tuning overview for latency-sensitive workloads.
• Emphasizes impactful new features included in RHEL7:
  • CPU/power management
  • NUMA
  • tuned profiles
  • scheduling
  • network tunables
  • kernel timers.
  • "de-jittering" CPU cores
  • tracing techniques

https://access.redhat.com/articles/1323793
Performance Optimizations in RHEL7

• busy_poll – new default

• tcp_fastopen
  • Reduce 1 round trip of handshake setting up TCP connection.

• nohz_full (tickless while active)
  • Timer ticks only on boot cpu or selected cpus

• Byte Queue Limits
  • Control bufferbloat in network queues
  • Helps tune high prio packets to get delivered w/reasonable latency
RHEL7 BUSY_POLL Socket Options

- Socket-layer code polls receive queue of NIC
- Replaces interrupts and NAPI
- Retains full capabilities of kernel network stack
- Set globally by tuned network-latency

To test if your device supports it:

```
# ethtool -k device | grep "busy-poll"
```

If this returns busy-poll: on [fixed], busy polling is available on the device.
Tuned: Network-Latency Performance Boost

C-state lock improves determinism, reduces jitter
Turbostat: Idle States and Frequencies on Intel CPUs

```
# tuned-adm profile throughput-performance
# turbostat sleep 5

<table>
<thead>
<tr>
<th>Bzy_MHz</th>
<th>TSC_MHz</th>
<th>SMI</th>
<th>CPU%c1</th>
<th>CPU%c3</th>
<th>CPU%c6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1866</td>
<td>2600</td>
<td>0</td>
<td>0.22</td>
<td>0.01</td>
<td>99.71</td>
</tr>
</tbody>
</table>
```

```
# tuned-adm profile network-latency
# turbostat sleep 5

<table>
<thead>
<tr>
<th>Bzy_MHz</th>
<th>TSC_MHz</th>
<th>SMI</th>
<th>CPU%c1</th>
<th>CPU%c3</th>
<th>CPU%c6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3108</td>
<td>2600</td>
<td>0</td>
<td>99.99</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
```
RHEL7 nohz_full

• Patchset Goal:
  – Stop interrupting userspace tasks
  – Move timekeeping to non-latency-sensitive cores
  – If nr_running=1, then scheduler/tick can avoid that core
• Default disabled...Opt-in via nohz_full cmdline option

• Kernel Tick:
  • timekeeping (gettimeofday)
  • Scheduler load balancing
  • Memory statistics (vmstat)
RHEL 7 nohz_full

Task is interrupted

Userspace Task

Timer Interrupt Tick

Time (CONFIG_HZ=1000)
RHEL Network NFV – DPDK + OVS
DPDK+OVS – kernel bypass, poll mode driver
RHEL 7.3 vhostuser – OVS 2.6 multi-queue

Upstream ovs-dpdk (2.6), Intel 10Gb
64-byte frames
22.7 Million packets per second!
OVS: 8 cores (16 threads), 2 bridges, each using 4 i40e PMD threads + 4 vhostuser PMD threads
VM 4: cores (8 threads), 2 vhostuser interfaces, each using 4 virtio PMD thread
WHY DPDK+OVS? RHEL7.3 KVM kernel vs DPDK

(Intel Haswell EP, 20c, 256 MB mem, 2-10Gb Nic, uni-directional .002% loss)

- Vihost-user
- DPDK Mpps
- Kernel Mpps
- DPDK cores
- Kernel cores

#redhat #rhsummit
New “cpu-partitioning” tuned profile

Do you have a latency sensitive application to tune for?

If so:

- Decide which cpus you want to allocated to it.
- Add those cpus to a tuned configuration file.
- Then reboot!

A highly tuned low latency system with cores dedicated to your application.

Numerous tedious tuning steps now handled by tuned!
VNF Mobile Network - Graphical CPU Partitioning

System Partitioning

- Physical core
- Kernel
  - Background activities
  - Kernel Interrupts
  - OVS Background process
- CPU Thread
- QEMU emulation process
- PMD Threads
- VNF NUMA 0 Threads
- VNF NUMA 1 Threads
Cpu-partitioning – What the profile does for you.

• After reboot you have:

1) Cpus dedicated to your application are isolated (not via the isolcpus boot flag)
2) IRQs, RCU callbacks, kernel dirty page threads - all moved off the isolated cpus
3) nohz_full set on the isolated cpus
4) CPU frequency scaling set to maximum allowed
5) MCE interrupts disabled
6) Kernel workqueue requests moved off isolated cpus
7) Kernel numa_balance and transparent hugepages disabled
8) Various KVM options set to reduce latency and to remove unwanted VM Exits and interrupts
9) Numerous SYSCTL parameters set to optimal low latency values

• Repeatable, automated, and very cool!
RHOP 10 w/ DPDK17.01 + OVS 2.6 L2 Forward tuned-adm profile=cpu-partitioning – 0.002% packet loss

(Intel Haswell EP, 20c, 256 MB mem, 2-10Gb Nic, uni-directional .002% loss)

<table>
<thead>
<tr>
<th>Frame size</th>
<th>Mpps @0.002% loss</th>
<th>Gbps @0.002% loss</th>
<th>Mpps/core @0.002% loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>22.93[1]</td>
<td>15.41</td>
<td>5.73</td>
</tr>
<tr>
<td>256</td>
<td>9.04</td>
<td>19.96</td>
<td>2.26</td>
</tr>
<tr>
<td>1024</td>
<td>2.39</td>
<td>19.96</td>
<td>0.59</td>
</tr>
<tr>
<td>1500</td>
<td>1.63</td>
<td>19.88</td>
<td>0.40</td>
</tr>
</tbody>
</table>

[1] Dual-port Intel 82599 based adapters are hardware limited to ~23Mpps
Noisy Cacheline Neighbor
Cache Allocation & Cache Monitoring Technology

Hi Priority Program

CPU 0

LO Priority Program

CPU 1

Shared Last Level Cache:
(Low Priority Program gets more cache)

VS:

Hi Priority Program

CPU 0

LO Priority Program

CPU 1

Shared Last Level Cache:
(HIGH Priority Program gets more cache)

Available in RHEL 7.4 via the intel-cmt-cat-*:el7 package.
See ‘man pqos’
Intel only. Recent CPU models.
Memory latency testing using CAT

Imbench lat_mem_rd speedup with CAT

With background "noisy neighbor" memory dirtying processes.

Memory read latency in nanosec

Memory array size in MB

Without CAT
Using CAT
System Tuning Tool - tuna

- Tool for fine grained control
- Display applications / processes
- Displays CPU enumeration
- Socket (useful for NUMA tuning)
- Dynamic control of tuning
  - Process affinity
  - Parent & threads
  - Scheduling policy
- Device IRQ priorities, etc
# tuna --help

- `h, --help`  Give this help list
- `a, --config_file_apply=profilename`  Apply changes described in profile
- `l, --config_file_list`  List preloaded profiles
- `g, --gui`  Start the GUI
- `G, --cgroup`  Display the processes with the type of cgroups they are in
- `c, --cpus=CPU-LIST`  CPU-LIST affected by commands
- `C, --affect_children`  Operation will affect children threads
- `f, --filter`  Display filter the selected entities
- `I, --isolate`  Move all threads away from CPU-LIST
- `i, --include`  Allow all threads to run on CPU-LIST
- `K, --no_kthreads`  Operations will not affect kernel threads
- `m, --move`  Move selected entities to CPU-LIST
- `N, --nohz_full`  CPUs in nohz_full= kernel command line will be affected by operations
- `p, --priority=[POLICY:]RTPRIO`  Set thread scheduler tunables: POLICY and RTPRIO
- `P, --show_threads`  Show thread list
- `Q, --show_irqs`  Show IRQ list
- `q, --irqs=IRQ-LIST`  IRQ-LIST affected by commands
- `r, --run=COMMAND`  fork a new process and run the COMMAND
- `s, --save=FILENAME`  Save kthreads sched tunables to FILENAME
- `S, --sockets=CPU-SOCKET-LIST`  CPU-SOCKET-LIST affected by commands
- `t, --threads=THREAD-LIST`  THREAD-LIST affected by commands
- `U, --no_uthreads`  Operations will not affect user threads
- `v, --version`  Show version
- `W, --what_is`  Provides help about selected entities
- `x, --spread`  Spread selected entities over CPU-LIST
Tuna – command line 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 GUI Capabilities Updated for RHEL7
Performance Tools - Perf
Partial list of available pre-defined perf events

# perf list
branch-instructions OR branches [Hardware event]
branch-misses [Hardware event]
cache-misses [Hardware event]
cpu-cycles OR cycles [Hardware event]
instructions [Hardware event]
context-switches OR cs [Software event]
page-faults OR faults [Software event]
LLC-load-misses [Hardware cache event]
LLC-loads [Hardware cache event]
LLC-store-misses [Hardware cache event]
LLC-stores [Hardware cache event]
dTLB-load-misses [Hardware cache event]
dTLB-loads [Hardware cache event]
dTLB-store-misses [Hardware cache event]
dTLB-stores [Hardware cache event]
node-load-misses [Hardware cache event]
node-store-misses [Hardware cache event]
intel_cqm/l1c occupancy/ [Kernel PMU event]
mem-loads OR cpu/mem-loads/ [Kernel PMU event]
mem-stores OR cpu/mem-stores/ [Kernel PMU event]
power/energy-pkg/ [Kernel PMU event]
tx-abort OR cpu/tx-abort/ [Kernel PMU event]
tx-capacity OR cpu/tx-capacity/ [Kernel PMU event]
tx-commit OR cpu/tx-commit/ [Kernel PMU event]
uncore_imc_0/cas_count_read/ [Kernel PMU event]
uncore_imc_0/cas_count_write/ [Kernel PMU event]

block:block_bio_backmerge [Tracepoint event]
block:block_bio_bounce [Tracepoint event]

#redhat #rhsummit
Most commonly used perf commands:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotate</td>
<td>Read perf.data (created by perf record) and display annotated code</td>
</tr>
<tr>
<td>archive</td>
<td>Create archive with object files with build-ids found in perf.data file</td>
</tr>
<tr>
<td>bench</td>
<td>General framework for benchmark suites</td>
</tr>
<tr>
<td>buildid-cache</td>
<td>Manage build-id cache.</td>
</tr>
<tr>
<td>buildid-list</td>
<td>List the buildids in a perf.data file</td>
</tr>
<tr>
<td>config</td>
<td>Get and set variables in a configuration file.</td>
</tr>
<tr>
<td>data</td>
<td>Data file related processing</td>
</tr>
<tr>
<td>diff</td>
<td>Read perf.data files and display the differential profile</td>
</tr>
<tr>
<td>evlist</td>
<td>List the event names in a perf.data file</td>
</tr>
<tr>
<td>inject</td>
<td>Filter to augment the events stream with additional information</td>
</tr>
<tr>
<td>kmem</td>
<td>Tool to trace/measure kernel memory properties</td>
</tr>
<tr>
<td>kvm</td>
<td>Tool to trace/measure kvm guest os</td>
</tr>
<tr>
<td>list</td>
<td>List all symbolic event types</td>
</tr>
<tr>
<td>lock</td>
<td>Analyze lock events</td>
</tr>
<tr>
<td>mem</td>
<td>Profile memory accesses</td>
</tr>
<tr>
<td>record</td>
<td>Run a command and record its profile into perf.data</td>
</tr>
<tr>
<td>report</td>
<td>Read perf.data (created by perf record) and display the profile</td>
</tr>
<tr>
<td>sched</td>
<td>Tool to trace/measure scheduler properties (latencies)</td>
</tr>
<tr>
<td>script</td>
<td>Read perf.data (created by perf record) and display trace output</td>
</tr>
<tr>
<td>stat</td>
<td>Run a command and gather performance counter statistics</td>
</tr>
<tr>
<td>test</td>
<td>Runs sanity tests.</td>
</tr>
<tr>
<td>timechart</td>
<td>Tool to visualize total system behavior during a workload</td>
</tr>
<tr>
<td>top</td>
<td>System profiling tool.</td>
</tr>
<tr>
<td>probe</td>
<td>Define new dynamic tracepoints</td>
</tr>
<tr>
<td>trace</td>
<td>strace inspired tool</td>
</tr>
</tbody>
</table>
**Example: perf record**

- Record system-wide (-a)
  - `perf record -a sleep 10`
  - `perf record -a // Hit ctrl-c when done.`

- Or record a single command
  - `perf record myapp.exe`

- Or record an existing process (-p)
  - `perf record -p <pid>`

- Or add call-chain recording (-g)
  - `perf record -g ls -rl /root`

- Or only record specific events (-e)
  - `perf record -e branch-misses -p <pid>`

**Followed by: perf report**

- `perf report [options]`
perf report

<table>
<thead>
<tr>
<th># Overhead</th>
<th>Command</th>
<th>Shared Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>43.53%</td>
<td>dd</td>
<td>[kernel.kallsyms] [k] __clear_user</td>
</tr>
<tr>
<td></td>
<td></td>
<td>__clear_user</td>
</tr>
<tr>
<td></td>
<td></td>
<td>--99.75%-- read_zero.part.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>read_zero</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vfs_read</td>
</tr>
<tr>
<td></td>
<td></td>
<td>sys_read</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system_call_fastpath</td>
</tr>
<tr>
<td></td>
<td></td>
<td>__GI___libc_read</td>
</tr>
</tbody>
</table>
|           |         | --0.25%-- [...]
| 5.37%     | dd      | [kernel.kallsyms] [k] do_blockdev_direct_IO |
|           |         | do_blockdev_direct_IO |
|           |         | __blockdev_direct_IO |
|           |         | xfs_vm_direct_IO |
|           |         | generic_file_direct_write |
|           |         | xfs_file_dio_aio_write |
|           |         | xfs_file_aio_write |
|           |         | do_sync_write |

/dev/zero

oflag=direct
perf top

System-wide 'top' view of busy functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Time</th>
<th>File</th>
<th>Type</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>[k] avtab_search_node</td>
<td>34.35%</td>
<td>httpd</td>
<td>[kernel.kallsyms]</td>
<td>]</td>
</tr>
<tr>
<td>[k] _spin_lock</td>
<td>12.70%</td>
<td>httpd</td>
<td>[kernel.kallsyms]</td>
<td>[k]</td>
</tr>
<tr>
<td>[k] tg_load_down</td>
<td>8.61%</td>
<td>httpd</td>
<td>[kernel.kallsyms]</td>
<td>[k]</td>
</tr>
<tr>
<td>[k] _spin_lock_irq</td>
<td>7.42%</td>
<td>httpd</td>
<td>[kernel.kallsyms]</td>
<td>[k]</td>
</tr>
<tr>
<td>[k] intel_idle</td>
<td>5.79%</td>
<td>init</td>
<td>[kernel.kallsyms]</td>
<td>[k]</td>
</tr>
<tr>
<td>[k] _spin_lock_irqsave</td>
<td>3.92%</td>
<td>httpd</td>
<td>[kernel.kallsyms]</td>
<td>[k]</td>
</tr>
<tr>
<td>[k] sidtab_search_core</td>
<td>1.75%</td>
<td>httpd</td>
<td>[kernel.kallsyms]</td>
<td>[k]</td>
</tr>
<tr>
<td>[k] load_balance_fair</td>
<td>1.74%</td>
<td>httpd</td>
<td>[kernel.kallsyms]</td>
<td>[k]</td>
</tr>
<tr>
<td>[k] tg_nop</td>
<td>1.18%</td>
<td>httpd</td>
<td>[kernel.kallsyms]</td>
<td>[k]</td>
</tr>
<tr>
<td>[k] _spin_lock</td>
<td>1.13%</td>
<td>init</td>
<td>[kernel.kallsyms]</td>
<td>[k]</td>
</tr>
</tbody>
</table>
perf diff / sched

Compare 2 perf recordings

grep for something interesting, maybe to see what numabalance is doing?
perf `c2c` for cpu cacheline false sharing detection

Shows everything needed to find false sharing problems.

- All readers and writers contending for the hottest cachelines.
  - The cpus and nodes they executed on.
- The 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).

Tends to show up in shared memory and/or multi-threaded programs.

Detailed blog at: [https://joemario.github.io/blog/2016/09/01/c2c-blog/](https://joemario.github.io/blog/2016/09/01/c2c-blog/)
CPU Cacheline False Sharing – 101

Two numa node system

- Reader thread
  - CPU0: L1, L2
  - CPU1: L1, L2
  - CPU2: L1, L2
  - CPU3: L1, L2
  - Memory
  - LLC (last level cache)

- Writer thread
  - CPU4: L1, L2
  - CPU5: L1, L2
  - CPU6: L1, L2
  - CPU7: L1, L2
  - Memory
  - LLC (last level cache)

- 64 Bytes of data in memory.

Writer

Reader
If the “writer” is truly hot, break it up.

<table>
<thead>
<tr>
<th>CPU0</th>
<th>CPU1</th>
<th>CPU2</th>
<th>CPU3</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>L1</td>
<td>L1</td>
<td>L1</td>
</tr>
<tr>
<td>L2</td>
<td>L2</td>
<td>L2</td>
<td>L2</td>
</tr>
</tbody>
</table>

LLC (last level cache)

Memory

Reader thread

<table>
<thead>
<tr>
<th>CPU4</th>
<th>CPU5</th>
<th>CPU6</th>
<th>CPU7</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>L1</td>
<td>L1</td>
<td>L1</td>
</tr>
<tr>
<td>L2</td>
<td>L2</td>
<td>L2</td>
<td>L2</td>
</tr>
</tbody>
</table>

LLC (last level cache)

Memory

Reader

writer

pad

pad

pad

pad

pad

pad
We often see this:

64 byte cache line

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>int a;</td>
<td>0</td>
</tr>
<tr>
<td>mutex</td>
<td>8</td>
</tr>
<tr>
<td>mutex</td>
<td>16</td>
</tr>
<tr>
<td>mutex</td>
<td>24</td>
</tr>
<tr>
<td>mutex</td>
<td>32</td>
</tr>
<tr>
<td>mutex</td>
<td>40</td>
</tr>
<tr>
<td>long b;</td>
<td>48</td>
</tr>
<tr>
<td>long seq_cnt;</td>
<td>56</td>
</tr>
</tbody>
</table>
Summary - Red Hat Enterprise Linux – Part 1

- Red Hat Enterprise Linux – Containers are Linux
  - RHEL x86 evolution, autoNUMA, Hugepages, IRQ pin, tuned profiles
  - NonUniform Memory Access (NUMA), numactl, numad, autonuma
- Network Performance
  - RHEL 7 Network tuning via tuned, try noHZ_full to reduce jitter
  - RHEL/KVM DPDK + OVS 2.6 multiQ, 0% packet loss in OSP10.
  - Tuna - IRQ placement, alter device affinities, process priorities
  - Perf – monitoring BW, cache-line tears, C-2-C analysis

- Q+A at “Meet The Experts” - Free as in Soda/Beer/Wine
Agenda: Performance Analysis Tuning Part II

• Part II - Containers are Linux, run/optimized and tuned just like Linux.
  • What is NUMA, RHEL Architecture, Auto-NUMA-Balance
  • Disk IO – IO elevators, sawpiness, dirty ratios, readahead, multi-Q
  • Scheduler tunables – new features
  • HugePages - Static, Transparent, variable sized 4K/2MB/1GB
  • Cgroups cpuset, memory, network and IO
    • Use to prevent IO from consuming 95% of memory
  • R+D GPUoffload, OpenHPC, multi-arch – Bill
  • Containers / Other preso's
  • “Performance + Scale Experts” - Room 205C 5:30-7 PM
  • Free - Soda/Beer/Wine
Mapping *tuned* profiles to Red Hat's product portfolio:

<table>
<thead>
<tr>
<th>RHEL Desktop/Workstation</th>
<th>RHEL Server/HPC</th>
<th>RHEL for Real Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>balanced</td>
<td>throughput-performance</td>
<td>realtime</td>
</tr>
<tr>
<td>RHV Host, Guest</td>
<td>RHV</td>
<td>RHEL for Real Time KVM-RT</td>
</tr>
<tr>
<td>virtual-host/guest</td>
<td>virtual-host</td>
<td>realtime-virtual-host/guest</td>
</tr>
<tr>
<td>Red Hat Storage</td>
<td>RHOSP (compute node)</td>
<td>RHEL + SAP</td>
</tr>
<tr>
<td>rhs-high-throughput, virt</td>
<td>virtual-host</td>
<td>sap / sap-hana</td>
</tr>
<tr>
<td>RHEL Atomic</td>
<td>OpenShift</td>
<td>RHOP - NFV (compute node)</td>
</tr>
<tr>
<td>atomic-host, atomic-guest</td>
<td>openshift-master, node</td>
<td>cpu-partitioning</td>
</tr>
</tbody>
</table>

New in 7.4
<table>
<thead>
<tr>
<th>64-bit</th>
<th>Node 1</th>
<th>Node 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>End of RAM</td>
<td>4GB DMA32 Zone</td>
<td>16MB DMA Zone</td>
</tr>
<tr>
<td>Normal Zone</td>
<td>Normal Zone</td>
<td></td>
</tr>
</tbody>
</table>
Per Node / Zone split LRU Paging Dynamics

User Allocations

- Reactivate
- Page aging
- swapout
- flush
- Reclaiming
- User deletions
Interaction between VM Tunables and NUMA

• Dependent on NUMA: Reclaim Ratios
  /proc/sys/vm/swappiness
  /proc/sys/vm/min_free_kbytes
  /proc/sys/vm/zone_reclaim_mode

• Independent of NUMA: Reclaim Ratios
  /proc/sys/vm/vfs_cache_pressure

• Writeback Parameters
  /proc/sys/vm/dirty_background_ratio
  /proc/sys/vm/dirty_ratio

• Readahead parameters
  /sys/block/<bdev>/queue/read_ahead_kb
swappiness

• Controls how aggressively the system reclaims anonymous memory versus pagecache memory:
  • Anonymous memory – swapping and freeing
  • File pages – writing if dirty and freeing
  • System V shared memory – swapping and freeing
• Default is 60
• Decrease: more aggressive reclaiming of pagecache memory
• Increase: more aggressive swapping of anonymous memory
• Can effect Numa nodes differently.
• Tuning not as necessary on RHEL7 than RHEL6 and even less than RHEL5
Memory reclaim Watermarks

Free memory list

All of RAM
Do nothing

Pages High – kswapd sleeps above High
kswapd reclaims memory

Pages Low – kswapd wakes up at Low
Wakeup kswapd and it reclaims memory

Pages Min – all memory allocators reclaim at Min
user processes/kswapd reclaim memory

0
min_free_kbytes

Directly controls the page reclaim watermarks in KB
Distributed between the Numa nodes
Defaults are higher when THP is enabled

```
# cat /proc/sys/vm/min_free_kbytes
90100
-----------------------------------------------------------
Node 0 DMA   min:80 low:100kB high:120kB
Node 0 DMA32 min:15312kB low:19140kB high:22968kB
Node 0 Normal min:29600kB low:37000kB high:44400kB
Node 1 Normal min:45108kB low:56384kB high:67660kB
-----------------------------------------------------------
```

```
echo 180200 > /proc/sys/vm/min_free_kbytes
```

```
-----------------------------------------------------------
Node 0 DMA   min:160kB low:200kB high:240kB
Node 0 DMA32 min:30624kB low:38280kB high:45936kB
Node 0 Normal min:59200kB low:74000kB high:88800kB
Node 1 Normal min:90216kB low:112768kB high:135320kB
-----------------------------------------------------------
```
RHEL Disk IO
I/O Tuning – Understanding I/O Elevators

- Deadline – new RHEL7 default for all profiles
  - Two queues per device, one for read and one for writes
  - I/Os dispatched based on time spent in queue
  - CFQ – used for system disks off SATA/SAS controllers
    - Per process queue
    - Each process queue gets fixed time slice (based on process priority)
- NOOP – used for high-end SSDs (Fusion IO etc)
  - FIFO
  - Simple I/O Merging
  - Lowest CPU Cost
Tuned: Profile throughput-performance (RHEL7 default)

```
throughput-performance

governor=performance
energy_perf_bias=performance
min_perf_pct=100
readahead=4096
kernel.sched_min_granularity_ns = 10000000
kernel.sched_wakeup_granularity_ns = 15000000
vm.dirty_background_ratio = 10
vm.swappiness=10
```
Iozone Performance effect of “tuned” EXT4/XFS/GFS

RHEL 7.1 3.10.0-253 File System In Cache Performance

Intel I/O (iozone - geoM 1m-4g, 4k-1m)

<table>
<thead>
<tr>
<th>File System</th>
<th>Throughput in MB/Sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext3</td>
<td>not tuned</td>
</tr>
<tr>
<td>ext4</td>
<td>not tuned</td>
</tr>
<tr>
<td>xfs</td>
<td>not tuned</td>
</tr>
<tr>
<td>gfs2</td>
<td>not tuned</td>
</tr>
</tbody>
</table>

RHEL 7 3.10.0-253 File System Out of Cache Performance

Intel I/O (iozone - geoM 1m-4g, 4k-1m)

<table>
<thead>
<tr>
<th>File System</th>
<th>Throughput in MB/Sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext3</td>
<td>not tuned</td>
</tr>
<tr>
<td>ext4</td>
<td>not tuned</td>
</tr>
<tr>
<td>xfs</td>
<td>not tuned</td>
</tr>
<tr>
<td>gfs2</td>
<td>not tuned</td>
</tr>
</tbody>
</table>
SAS Application on Standalone Systems

RHEL 7 limits

**xfs**  most recommended
- Max file system size 500TB
- Max file size 100 TB
- Best performing

**ext4** recommended
- Max file system size 50 TB
- Max file size 16 TB

**ext3** not recommended
- Max file system size 16TB
- Max file size 2TB

![SAS Mixed Analytics (RHEL6 vs RHEL7)](chart.png)

perf 32 (2 socket) 16cpu x 48GB

<table>
<thead>
<tr>
<th>File system tested</th>
<th>TOTAL Time</th>
<th>System Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>xfs-rhel6</td>
<td>6.18</td>
<td>-4.05</td>
</tr>
<tr>
<td>ext3-rhel6</td>
<td>4.94</td>
<td>9.59</td>
</tr>
<tr>
<td>ext4-rhel6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gfs2-rhel6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tuning Memory – **Flushing Caches**

- Drop unused Cache – to control pagecache dynamically
  - ✔ Frees most pagecache memory
  - ✔ File cache
  - ✗ If the DB uses cache, may notice slowdown

- NOTE: Use for benchmark environments.

- **Free pagecache**
  - # sync; echo 1 > /proc/sys/vm/drop_caches

- **Free slabcache**
  - # sync; echo 2 > /proc/sys/vm/drop_caches

- **Free pagecache and slabcache**
  - # sync; echo 3 > /proc/sys/vm/drop_caches
Per file system flush daemon

**User space**

- `buffer`
- `read()/write()`
- `memory copy`

**Kernel**

- `pagecache`
- `flush daemon`
- `file system`

**Diagram**:

- `buffer` connected to `read()/write()`
- `read()/write()` connected to `memory copy`
- `memory copy` connected to `pagecache`
- `pagecache` connected to `flush daemon`
- `flush daemon` connected to `file system`
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_background_ratio, dirty_background_bytes

- Controls when dirty pagecache memory starts getting written.
- Default is 10%
- Lower
  - flushing starts earlier
  - less dirty pagecache and smaller IO streams
- Higher
  - flushing starts later
  - more dirty pagecache and larger IO streams
- dirty_background_bytes over-rides when you want < 1%
dirty_ratio and dirty_background_ratio

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
RHEL CFS Scheduler
RHEL Scheduler Tunables

Implements multiple red/black trees as run queues for sockets and cores (as opposed to one run queue per processor or per system)

RHEL tunables
- sched_min_granularity_ns
- sched_wakeup_granularity_ns
- sched_migration_cost
- sched_child_runs_first
- sched_latency_ns
Finer Grained Scheduler Tuning

- RHEL6/7 Tuned-adm will increase quantum on par with RHEL5
  - echo 10000000 > /proc/sys/kernel/sched_min_granularity_ns
    - Minimal preemption granularity for CPU bound tasks. See sched_latency_ns for details. The default value is 4000000 (ns).
  - echo 15000000 > /proc/sys/kernel/sched_wakeup_granularity_ns
    - The wake-up preemption granularity.
    - Increasing this variable reduces wake-up preemption, reducing disturbance of compute bound tasks.
    - Decreasing it improves wake-up latency and throughput for latency critical tasks, particularly when a short duty cycle load component must compete with CPU bound components. The default value is 5000000 (ns).
Load Balancing

• Scheduler tries to keep all CPUs busy by moving tasks from overloaded CPUs to idle CPUs

• Detect using “perf stat”, look for excessive “migrations”

• `/proc/sys/kernel/sched_migration_cost`
  – Amount of time after the last execution that a task is considered to be “cache hot” in migration decisions. A “hot” task is less likely to be migrated, so increasing this variable reduces task migrations. The default value is 500000 (ns).
  – If the CPU idle time is higher than expected when there are runnable processes, try reducing this value. If tasks bounce between CPUs or nodes too often, try increasing it.

• Rule of thumb – increase by 2-10x to reduce load balancing (tuned does this)

• Use 10x on large systems when many CGROUPs are actively used (ex: RHEV/KVM/RHOS)
fork() behavior

sched_child_runs_first
• Controls whether parent or child runs first
• Default is 0: parent continues before children run.
• Default is different than RHEL5

RHEL6 Effect of sched_migration cost on fork/exit

Intel Westmere EP 24cpu/12core, 24 GB mem

usec/call default 500us
usec/call tuned 4ms
percent improvement
RHEL7.4 Core Kernel Features

- Deadline Scheduler
  - allows process to define when it must run.
- Persistent Memory
  - supports huge amounts of non-volatile RAM
- Qspinlock
  - prevents cacheline contention causes locking contention
- RWsemaphore locking
  - performance improvement to kernel r/w semaphores
- KASLR
  - kernel addr space randomization provide better security

New in 7.4
RHEL VM HugePages
RHEL Hugepages/ VM Tuning

• Standard HugePages 2MB
  - Reserve/free via
  • /proc/sys/vm/nr_hugepages
  • /sys/devices/node/*/hugepages/*/nrhugepages
  - Used via hugetlbfs

• GB Hugepages 1GB
  - Reserved at boot time/no freeing
  - RHEL7 allows runtime allocation & freeing
  - Used via hugetlbfs

• Transparent HugePages 2MB
  - On by default via boot args or /sys
  - Used for anonymous memory
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%
2MB standard Hugepages

```bash
# 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
```
# echo 0 > /proc/sys/vm/nr_hugepages
# cat /proc/meminfo | grep HugePages_Free
HugePages_Free: 0

# echo 1000 > /proc/sys/vm/nr_hugepages
# cat /proc/meminfo | grep HugePages_Free
HugePages_Free: 1000
# cat /sys/devices/system/node/node*/hugepages/hugepages-2048kB/nr_hugepages
500
500

# echo 0 > /proc/sys/vm/nr_hugepages
# echo 1000 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
# cat /proc/meminfo | grep HugePages_Free
HugePages_Free: 1000
# cat /sys/devices/system/node/node*/hugepages/hugepages-2048kB/nr_hugepages
1000
0
Database Performance OLTP (Higher = Better)
huge pages on Bare Metal

Oracle OLTP tran/min

The effect of hugepages are more pronounced as system drive to saturation
Boot-time allocated 1GB Hugepages

Boot arguments

- default_hugepagesz=1G, hugepagesz=1G, hugepages=8

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

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

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

Dynamic per-node allocation/deallocation of 1GB Hugepages

```bash
# cat /sys/devices/system/node/node*/hugepages/hugepages-1048576kB/nr_hugepages
0
0

# echo 8 > /sys/devices/system/node/node0/hugepages/hugepages-1048576kB/nr_hugepages
# cat /proc/meminfo | grep HugePages_Free
HugePages_Free:     8
# cat /sys/devices/system/node/node*/hugepages/hugepages-1048576kB/nr_hugepages
8
0

# echo 0 > /sys/devices/system/node/node0/hugepages/hugepages-1048576kB/nr_hugepages
# cat /proc/meminfo | grep HugePages_Free
HugePages_Free:     0
# cat /sys/devices/system/node/node*/hugepages/hugepages-1048576kB/nr_hugepages
0
0
```
SAP Performance w/ Hana

RHEL7.3 RHV4.x - SAP HANA OLTP Cert (250 test) - Baseline vs. Tuned Guest

Intel Haswell EX, 144 CPU, 4 socket, 512 GB memory, 2 PCI NVME cards

<table>
<thead>
<tr>
<th>% Tests Passing</th>
<th>Baseline Large 136 vcpu Guest</th>
<th>Vcpupin, numa, 1GB Huge, LLC patch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update</td>
<td>50</td>
<td>90</td>
</tr>
<tr>
<td>Delete</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>Insert</td>
<td>50</td>
<td>90</td>
</tr>
<tr>
<td>Select</td>
<td>20</td>
<td>70</td>
</tr>
</tbody>
</table>

New
RHEL Control Group - Cgroups
Cgroup default mount points

RHEL6

```bash
# cat /etc/cgconfig.conf

mount {
    cpuset = /cgroup/cpuset;
    cpu = /cgroup/cpu;
    cpuacct = /cgroup/cpuacct;
    memory = /cgroup/memory;
    devices = /cgroup/devices;
    freezer = /cgroup/freezer;
    net_cls = /cgroup/net_cls;
    blkio = /cgroup/blkio;
}
```

RHEL7

```bash
# ls -l /sys/fs/cgroup/

blkio   drwxr-xr-x. 2 root root  0 Mar 20 16:40
cpu,cpuacct drwxr-xr-x. 2 root root  0 Mar 20 16:40
hsset      drwxr-xr-x. 2 root root  0 Mar 20 16:40
devices   drwxr-xr-x. 2 root root  0 Mar 20 16:40
freezer   drwxr-xr-x. 2 root root  0 Mar 20 16:40
hugetlb   drwxr-xr-x. 3 root root  0 Mar 20 16:40
memory    drwxr-xr-x. 2 root root  0 Mar 20 16:40
net_cls   drwxr-xr-x. 2 root root  0 Mar 20 16:40
perf_event drwxr-xr-x. 4 root root  0 Mar 20 16:40
systemd   drwxr-xr-x. 4 root root  0 Mar 20 16:40
```
Cgroup how-to

Create a 2GB/4CPU subset of a 16GB/8CPU system

# numactl --hardware
# mount -t cgroup xxx /cgroups
# mkdir -p /cgroups/test
# cd /cgroups/test
# echo 0 > cpuset.mems
# echo 0-3 > cpuset.cpus
# echo 2G > memory.limit_in_bytes
# echo $$ > tasks
## cgroups

```bash
# echo 0-3 > cpuset.cpus
# runmany 20MB 110procs &
# top -d 5

top - 12:24:13 up  1:36,  4 users, load average: 22.70, 5.32, 1.79
Tasks: 315 total,  93 running, 222 sleeping,   0 stopped,   0 zombie

<table>
<thead>
<tr>
<th>CPU</th>
<th>us %</th>
<th>sy %</th>
<th>ni %</th>
<th>id %</th>
<th>wa %</th>
<th>hi %</th>
<th>si %</th>
<th>st %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cpu0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Cpu1</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Cpu2</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Cpu3</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Cpu4</td>
<td>0.4</td>
<td>0.6</td>
<td>0.0</td>
<td>98.8</td>
<td>0.0</td>
<td>0.0</td>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td>Cpu5</td>
<td>0.4</td>
<td>0.0</td>
<td>0.0</td>
<td>99.2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.4</td>
<td>0.0</td>
</tr>
<tr>
<td>Cpu6</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Cpu7</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>99.8</td>
<td>0.0</td>
<td>0.0</td>
<td>0.2</td>
<td>0.0</td>
</tr>
</tbody>
</table>
```

#rhsummit
### Correct NUMA bindings

```
# echo 0 > cpuset.mems
# echo 0-3 > cpuset.cpus
# numastat

<table>
<thead>
<tr>
<th></th>
<th>node0</th>
<th>node1</th>
</tr>
</thead>
<tbody>
<tr>
<td>numa_hit</td>
<td>1648772</td>
<td>438778</td>
</tr>
<tr>
<td>numa_miss</td>
<td>23459</td>
<td>2134520</td>
</tr>
<tr>
<td>local_node</td>
<td>1648648</td>
<td>423162</td>
</tr>
<tr>
<td>other_node</td>
<td>23583</td>
<td>2150136</td>
</tr>
</tbody>
</table>

# /common/lwoodman/code/memory 4G
faulting took 1.616062s
touching took 0.364937s
```

```
# numastat

<table>
<thead>
<tr>
<th></th>
<th>node0</th>
<th>node1</th>
</tr>
</thead>
<tbody>
<tr>
<td>numa_hit</td>
<td>2700423</td>
<td>439550</td>
</tr>
<tr>
<td>numa_miss</td>
<td>23459</td>
<td>2134520</td>
</tr>
<tr>
<td>local_node</td>
<td>2700299</td>
<td>423934</td>
</tr>
<tr>
<td>other_node</td>
<td>23583</td>
<td>2150136</td>
</tr>
</tbody>
</table>
```

### Incorrect NUMA bindings

```
# echo 1 > cpuset.mems
# echo 0-3 > cpuset.cpus
# numastat

<table>
<thead>
<tr>
<th></th>
<th>node0</th>
<th>node1</th>
</tr>
</thead>
<tbody>
<tr>
<td>numa_hit</td>
<td>1623318</td>
<td>434106</td>
</tr>
<tr>
<td>numa_miss</td>
<td>23459</td>
<td>1082458</td>
</tr>
<tr>
<td>local_node</td>
<td>1623194</td>
<td>418490</td>
</tr>
<tr>
<td>other_node</td>
<td>23583</td>
<td>1098074</td>
</tr>
</tbody>
</table>

# /common/lwoodman/code/memory 4G
faulting took 1.976627s
touching took 0.454322s
```

```
# numastat

<table>
<thead>
<tr>
<th></th>
<th>node0</th>
<th>node1</th>
</tr>
</thead>
<tbody>
<tr>
<td>numa_hit</td>
<td>1623341</td>
<td>434147</td>
</tr>
<tr>
<td>numa_miss</td>
<td>23459</td>
<td>2133738</td>
</tr>
<tr>
<td>local_node</td>
<td>1623217</td>
<td>418531</td>
</tr>
<tr>
<td>other_node</td>
<td>23583</td>
<td>2149354</td>
</tr>
</tbody>
</table>
# cat cpu.shares
1024

# echo 10 > cpu.shares

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>20104</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>360</td>
<td>284</td>
<td>R</td>
<td>99.4</td>
<td>0.0</td>
<td>12:35.83 useless</td>
</tr>
<tr>
<td>20103</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>356</td>
<td>284</td>
<td>R</td>
<td>91.4</td>
<td>0.0</td>
<td>12:34.78 useless</td>
</tr>
<tr>
<td>20105</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>360</td>
<td>284</td>
<td>R</td>
<td>90.4</td>
<td>0.0</td>
<td>12:33.08 useless</td>
</tr>
<tr>
<td>20106</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>360</td>
<td>284</td>
<td>R</td>
<td>88.4</td>
<td>0.0</td>
<td>12:32.81 useless</td>
</tr>
<tr>
<td>20102</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>360</td>
<td>284</td>
<td>R</td>
<td>86.4</td>
<td>0.0</td>
<td>12:35.29 useless</td>
</tr>
<tr>
<td>20109</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>360</td>
<td>284</td>
<td>R</td>
<td>85.4</td>
<td>0.0</td>
<td>12:33.51 useless</td>
</tr>
<tr>
<td>20107</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>360</td>
<td>284</td>
<td>R</td>
<td>84.8</td>
<td>0.0</td>
<td>12:31.87 useless</td>
</tr>
<tr>
<td>20108</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>360</td>
<td>284</td>
<td>R</td>
<td>82.1</td>
<td>0.0</td>
<td>12:30.55 useless</td>
</tr>
</tbody>
</table>
| 20410 | root | 20 | 0   | 4160  | 360  | 284 | R  | 91.4 | 0.0  | 0:18.51 useful   

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>20102</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>360</td>
<td>284</td>
<td>R</td>
<td>100.0</td>
<td>0.0</td>
<td>0:17.45 useless</td>
</tr>
<tr>
<td>20103</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>356</td>
<td>284</td>
<td>R</td>
<td>100.0</td>
<td>0.0</td>
<td>0:17.03 useless</td>
</tr>
<tr>
<td>20107</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>360</td>
<td>284</td>
<td>R</td>
<td>100.0</td>
<td>0.0</td>
<td>0:15.57 useless</td>
</tr>
<tr>
<td>20104</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>360</td>
<td>284</td>
<td>R</td>
<td>99.8</td>
<td>0.0</td>
<td>0:16.66 useless</td>
</tr>
<tr>
<td>20105</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>360</td>
<td>284</td>
<td>R</td>
<td>99.8</td>
<td>0.0</td>
<td>0:16.31 useless</td>
</tr>
<tr>
<td>20108</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>360</td>
<td>284</td>
<td>R</td>
<td>99.8</td>
<td>0.0</td>
<td>0:15.19 useless</td>
</tr>
<tr>
<td>20110</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>360</td>
<td>284</td>
<td>R</td>
<td>99.4</td>
<td>0.0</td>
<td>0:14.74 useless</td>
</tr>
<tr>
<td>20109</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>360</td>
<td>284</td>
<td>R</td>
<td>99.1</td>
<td>0.0</td>
<td>0:15.87 useless</td>
</tr>
<tr>
<td>20111</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>4160</td>
<td>356</td>
<td>284</td>
<td>R</td>
<td>1.0</td>
<td>0.0</td>
<td>0:00.08 useful</td>
</tr>
</tbody>
</table>
C-group Dynamic resource control

Dynamic CPU Change
Oracle OLTP Workload

Transactions Per Minute

<table>
<thead>
<tr>
<th>Control Group CPU Count</th>
<th>Instance 1</th>
<th>Instance 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>cgrp 1 (4), cgrp 2 (32)</td>
<td>200K</td>
<td>200K</td>
</tr>
<tr>
<td>cgrp 1 (32), cgrp 2 (4)</td>
<td>80K</td>
<td>80K</td>
</tr>
</tbody>
</table>
cpu.cfs_quota_us unlimited

```
# cat cpu.cfs_period_us
100000
# cat cpu.cfs_quota_us
-1
```

top - 10:11:33 up 13 days, 17:31, 11 users, load average: 6.21, 7.78, 6.80

```
PID USER        PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
20614 root         20   0    4160      360    284  R 100.0 0.0  0:30.77 useful
```

```
# echo 1000 > cpu.cfs_quota_us
```

top - 10:16:55 up 13 days, 17:36, 11 users, load average: 0.07, 2.87, 4.93

```
PID USER        PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
20645 root        20   0      4160    360      284 R  1.0  0.0  0:01.54 useful
```

#redhat #rhsummit
Cgroup OOMkills

# mkdir -p /sys/fs/cgroup/memory/test
# echo 1G > /sys/fs/cgroup/memory/test/memory.limit_in_bytes
# echo 2G > /sys/fs/cgroup/memory/test/memory.memswo.limit_in_bytes
# echo $$ > /sys/fs/cgroup/memory/test/tasks

# ./memory 16G
size = 10485760000
touching 2560000 pages
Killed
# vmstat 1
...
Cgroup OOMkills (continued)

```
# vmstat 1
... 
0  0  52224 1640116      0 3676924    0    0       0     0   202  487  0  0 100  0  0  
1  0  52224 1640116      0 3676924    0    0       0     0   162  316  0  0 100  0  0   
0  1 248532 587268       0 3676948   32 196312    32 196372  912  974  1  4 88   7  0   
0  1 406228 586572       0 3677308   0 157696     0 157704  624  696  0  1 87  11  0   
0  1 568532 585928       0 3676864   0 162304     0 162312  722 1039  0  2 87  11  0   
0  1 729300 584744       0 3676840   0 160768     0 160776  719 1161  0  2 87  11  0   
1  0 885972 585404       0 3677008   0 156844     0 156852  754 1225  0  2 88  10  0   
0  1 1042644 587128      0 3676784   0 156500     0 156508  747 1146  0  2 86  12  0   
0  1 1169708 587396      0 3676748   0 127064     4 127836  702 1429  0  2 88  10  0   
0  0  86648 1607092      0 3677020  144  0     148   0  491 1151  0  1 97   1  0   
... 
```

```
# dmesg
... 
[506858.413341]  Task in /test killed as a result of limit of /test
[506858.413342] memory: usage 1048460kB, limit 1048576kB, failcnt 295377
[506858.413343] memory+swap: usage 2097152kB, limit 2097152kB, failcnt 74
[506858.413344] kmem: usage 0kB, limit 9007199254740991kB, failcnt 0
[506858.413345] Memory cgroup stats for /test: cache:0KB rss:1048460KB rss_huge:10240KB mapped_file:0KB swap:1048692KB inactive_anon:524372KB active_anon:524084KB inactive_file:0KB active_file:0KB unevictable:0KB
```
Even though the “RED” application does not have resources and starts swapping, The other applications are not affected.
Red Hat R+D HPC, GPU, Multi-Arch
HPC R&D: OpenHPC

- A Linux Foundation project to collect and distribute a standardized set of HPC tools and libraries
- OpenHPC Mission: to provide a reference collection of open-source HPC software components and best practices, lowering barriers to deployment, advancement, and use of modern HPC methods and tools.
- Red Hat joined OpenHPC in 2016
- Would like to hear from any of you interested in this topic.
Multiple-Architecture R&D:

- Red Hat Enterprise Linux: multiple architectures, one experience.
- Driving standardized interfaces and consistent user experiences
- Open, flexible, familiar across common architectures
- Would like to hear from any of you interested in this topic.
GPU / Accelerator R&D:

• Various research efforts exploring offloading and acceleration technologies spanning bare metal, virtualization, and containers
• Looking for input to prioritize important use cases such as:
  • AI / ML (Tensorflow, Theano, Torch, etc.)
  • Image analysis
  • Scientific applications
  • Accelerated libraries
  • Software development tools for accelerated applications
  • Other applications
RHEL Atomic Containers
RED HAT ENTERPRISE LINUX
ATOMIC HOST

MINIMAL, SECURE FOOTPRINT
- Minimal host provides “just enough” to support apps.

RAPID PROVISIONING
- Apps can be provisioned and started in milliseconds.

SIMPLIFIED MAINTENANCE
- Atomic updates are quick, reliable, and can be rolled back.

Atomic Tuned Profile Inheritance

Parents

throughput-performance

Children

virtual-host  virtual-guest

Children/Grandchildren

atomic-openshift-node-guest
Capacity Planning / Memory Usage

Application Memory Usage (Baseline vs Loaded (100tps))

MB RSS

Golang  |  PHP  |  Django  |  NodeJS

Baseline  |  Under Load

#redhat #rhsummit
OSE 3.4 Cloud Native Storage w/ Gluster

OSE 3.2, Gluster FIO (5 nodes)

- Pods / Pods (Hyperconverged)
- Pods / Pods (Separate)
- Bare Metal / Bare Metal (Separate)

Throughput (MB/s)

- seq.read
- seq.write
Using Cgroups – Docker, Kubernetes

/sys/fs/cgroup/blkio/system.slice/docker-0d4aeda3.scope
/sys/fs/cgroup/cpu,cpuacct/system.slice/docker-0d4aeda3.scope
/sys/fs/cgroup/cpuset/system.slice/docker-0d4aeda3.scope
/sys/fs/cgroup/devices/system.slice/docker-0d4aeda3.scope
/sys/fs/cgroup/freezer/system.slice/docker-0d4aeda3.scope
/sys/fs/cgroup/hugetlb/system.slice/docker-0d4aeda3.scope
/sys/fs/cgroup/memory/system.slice/docker-0d4aeda3.scope
/sys/fs/cgroup/net_cls,net_prio/system.slice/docker-0d4aeda3.scope
/sys/fs/cgroup/perf_event/system.slice/docker-0d4aeda3.scope
Using Cgroups – cgconfig.conf

```plaintext
group app1 {
  cpuset {
    cpuset.cpus = "2,4,6,8,10";
    cpuset.mems = 0;
  }
}
group app2 {
  cpuset {
    cpuset.cpus = "1,3,5,7,9";
    cpuset.mems = 1;
  }
}
```
RHEL7 Performance Tuning Summary – Part II

• NonUniform Memory Access (NUMA) – kernel details / tuning
• Scheduler tunables adjust quantum, forking, migration cost, new deadline scheduler in 7.x
• HugePages - static, THP, variable sized 4K/2MB/1GB
• Cgroups cpuset, memory, network and IO
  • use to prevent IO from consuming 95% of memory
• Disk IO - IO elevators, sawpiness, dirty ratios, readahead, multi-Q
• R+D GPUoffload, OpenHPC, multi-arch
• Containers
  • Containers are Linux, run/optimized and tuned just like Linux.

• Q+A in “Meet The Experts” - Free as in Soda/Beer/Wine
Performance Whitepapers

- Performance Tuning of Satellite 6.1 and Capsules
  https://access.redhat.com/articles/2356131
- OpenShift v3 Scaling, Performance and Capacity Planning
  https://access.redhat.com/articles/2191731
- Performance and Scaling your RHEL OSP 7 Cloud
  https://access.redhat.com/articles/2165131
  - Update guides to perf / virt to rhel7, add containers
- Red Hat Enterprise Linux Tuning Guide RHEL7
- Red Hat Virtualization Tuning Guide
- Comprehensive Overview of Storage Scalability in Docker
- RHEL Blog / Developer Blog
THANK YOU
LEARN. NETWORK. EXPERIENCE OPEN SOURCE.