Lessons Learned:
Optimizing KVM Performance for EHR Systems

Jon Kohler
Principal Solutions Architect, Nutanix

jon@nutanix.com

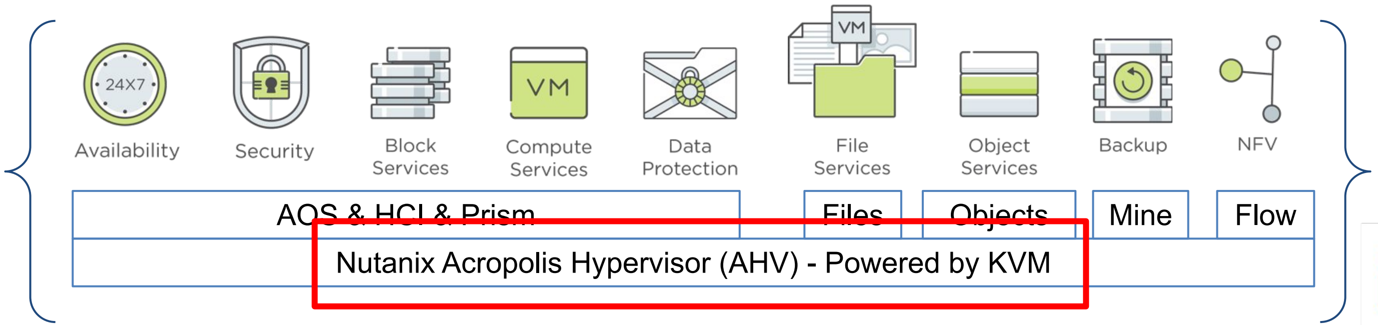
Agenda

- Quick Context: EHR Systems, Challenges?
- Diving in: debugging a real EHR perf issue
  - Practical Example
  - Optimization Results
- Zooming out: Related Ecosystem Enhancements
- Appendix:
  - Debugging Deep Dive Content
  - New to Flamegraphs?

Catering for EHR systems requires diligence

Epic
MEDITECH
ChipSoft
Allscripts
eClinicalWorks
InterSystems

- ISV mandates application architecture and core systems behavior.
- Provisioning and Scaling are usually in-elastic, systems rarely get smaller, and they live-on for a very long time (decades).
- End-users of all abilities and tolerance levels touch system 24/7.
- Some use cases may be life-critical, tolerance for system unavailability and poor performance is close to zero.
- Many users simply can't "come back later" when system is working better
- All comes together to keep Infrastructure layer "on its toes".
- "It takes the whole village" across both KVM and the broader ecosystem to optimize and sustain performance.



Diving In
A Practical Example

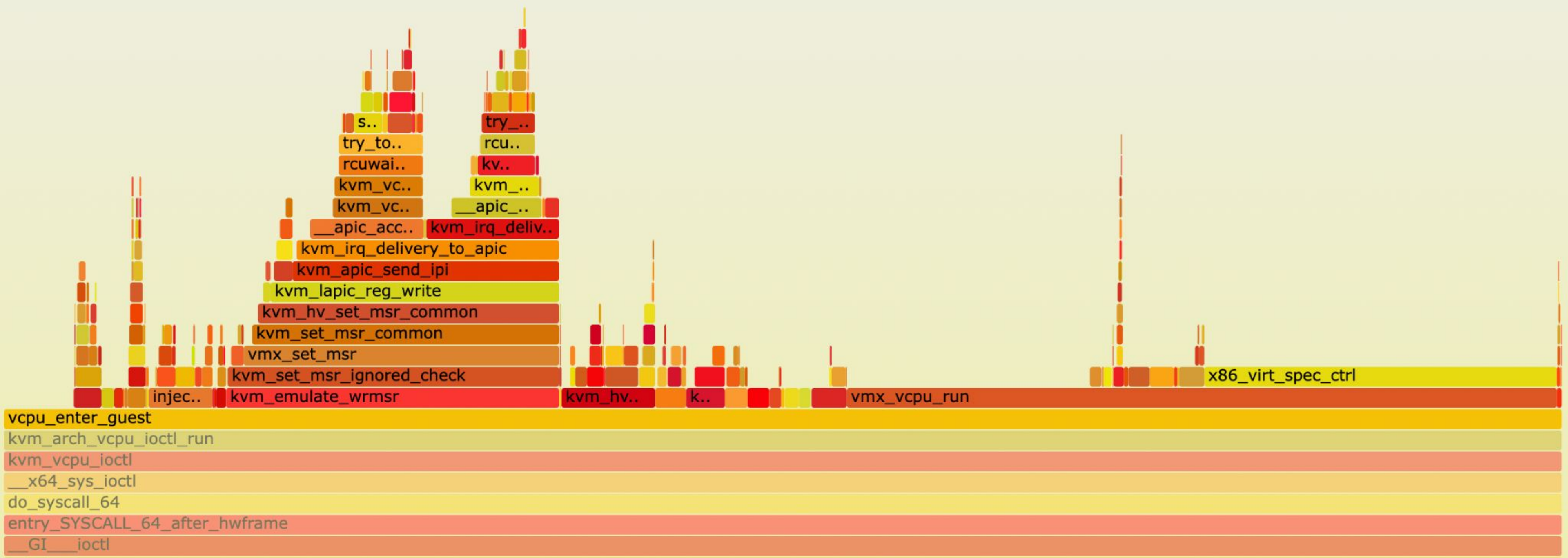
Practical Example
Reported Scaling Issue on EHR* App Layer

- Windows based “front end” EHR workload
- ISV Benchmark reported degradation in scaling vs competitive platform
- Nutanix AHV 8.0 pre-GA, Kernel 5.10.y based
- Intel Ice Lake hardware
- Particular EHR App “front end” is *particularly* latency sensitive to CPU perf
- Benchmark loads up synthetic users with increasing density/core and measures response time (think load-runner esque), has SLA “fault” line where response time will be unacceptable.
- Multiple Windows VM’s/host, lightly oversubscribed

THE LINUX FOUNDATION

* [redacted]

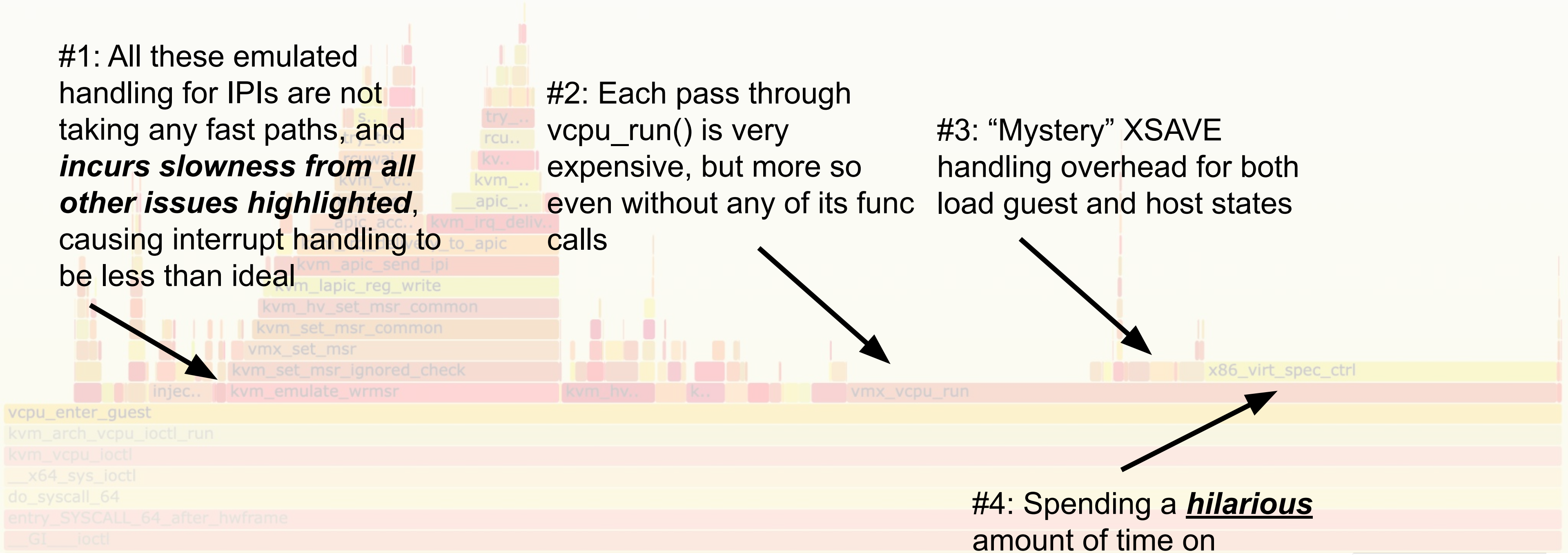
Diving In: Reported Competitive Scaling Issue
What’s wrong with this picture?



New to FlameGraphs? See copy-paste example in Slide Appendix

THE LINUX FOUNDATION

Diving In: Reported Competitive Scaling Issue
What’s wrong with this picture?



#1: All these emulated handling for IPIs are not taking any fast paths, and ***incurs slowness from all other issues highlighted***, causing interrupt handling to be less than ideal

#2: Each pass through vcpu_run() is very expensive, but more so even without any of its func calls

#3: “Mystery” XSAVE handling overhead for both load guest and host states

#4: Spending a ***<u>hilarious</u>*** amount of time on speculation control

Diving In: Reported Competitive Scaling Issue
Issue 1 and 2 Summary

1. IPI handling overhead, Windows Specific!
    a. HyperV SynIC enabled with AutoEOI (default), which disables hardware (Intel APICv) handling
    b. Even with (new) hv-apicv, Windows exits with APIC_WRITE rather than MSR_WRITE. APIC_WRITE's do not have exit fastpath handler
    c. **Fix 1a**: Switch to hv-apicv
    d. **Fix 1b**: Introduce new APIC_WRITE fastpath handler
2. vmx_vcpu_run() overhead (~10% FG Samples)
    a. Guest enabled eIBRS, each exit spams rdmsr SPEC_CTRL (expensive!), caused by MSR bitmap interception from eIBRS enablement
    b. Also, host debugctl msr update spamming unnecessary (expensive!)
    c. **Fix 2a:** Fixup enablement for eIBRS so kernel does not disable interception
    d. **Fix 2b:** Revert offending commit for debugctl msr issue

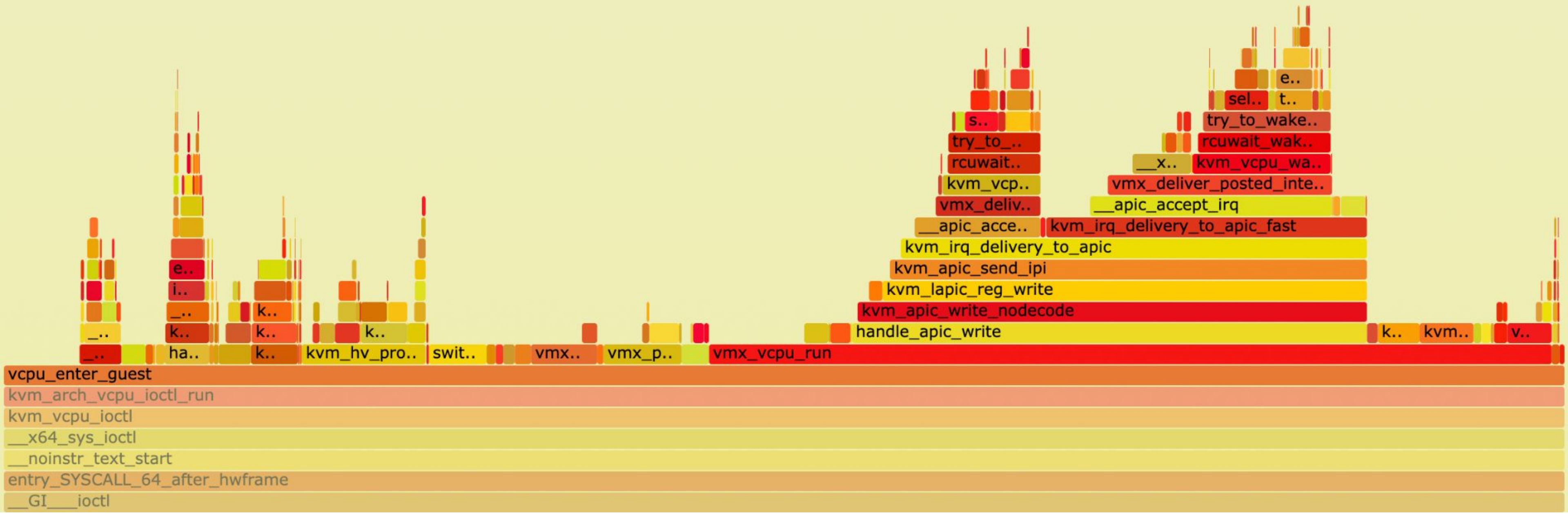
Diving In: Reported Competitive Scaling Issue
Issue 3 and 4 Summary

1. XSAVE overhead (~1% FG Samples)
    a. Due to Guest vs Host xfeatures state mismatch, each entry/exit spams xsetbv (expensive!)
    b. Mismatch is caused by control plane feature auto-masking MPX and PKU away from guest.
    c. **Fix 3**: WIP - Fixup early initialization code in FPU to properly compile out MPX and PKU
2. x86_virt_spec_ctrl overhead (~8% FG Samples)
    a. Due to Guest vs Host SPEC_CTRL mismatch, each entry/exit spams wrmsr SPEC_CTRL
    b. wrmsr to SPEC_CTRL stalls pipeline completely (**very expensive**)
    c. Mismatch is (unintentionally?) forced by qemu seccomp config vs kernel spectre_v2=auto
    d. **Fix 4**: Backport default change in bugs.c to depessmize seccomp

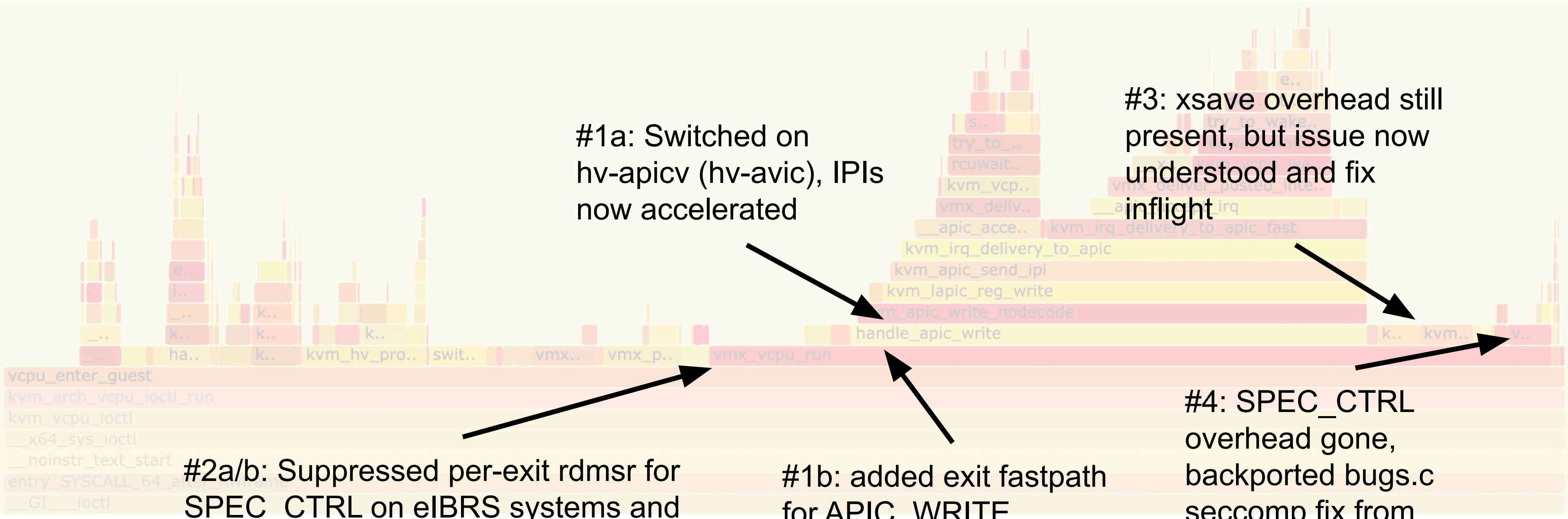
THE LINUX FOUNDATION

Diving In
Optimization Results

Optimization Results: Improved Flamegraph



Optimization Results: Improved Flamegraph



#1a: Switched on hv-apicv (hv-avic), IPIs now accelerated

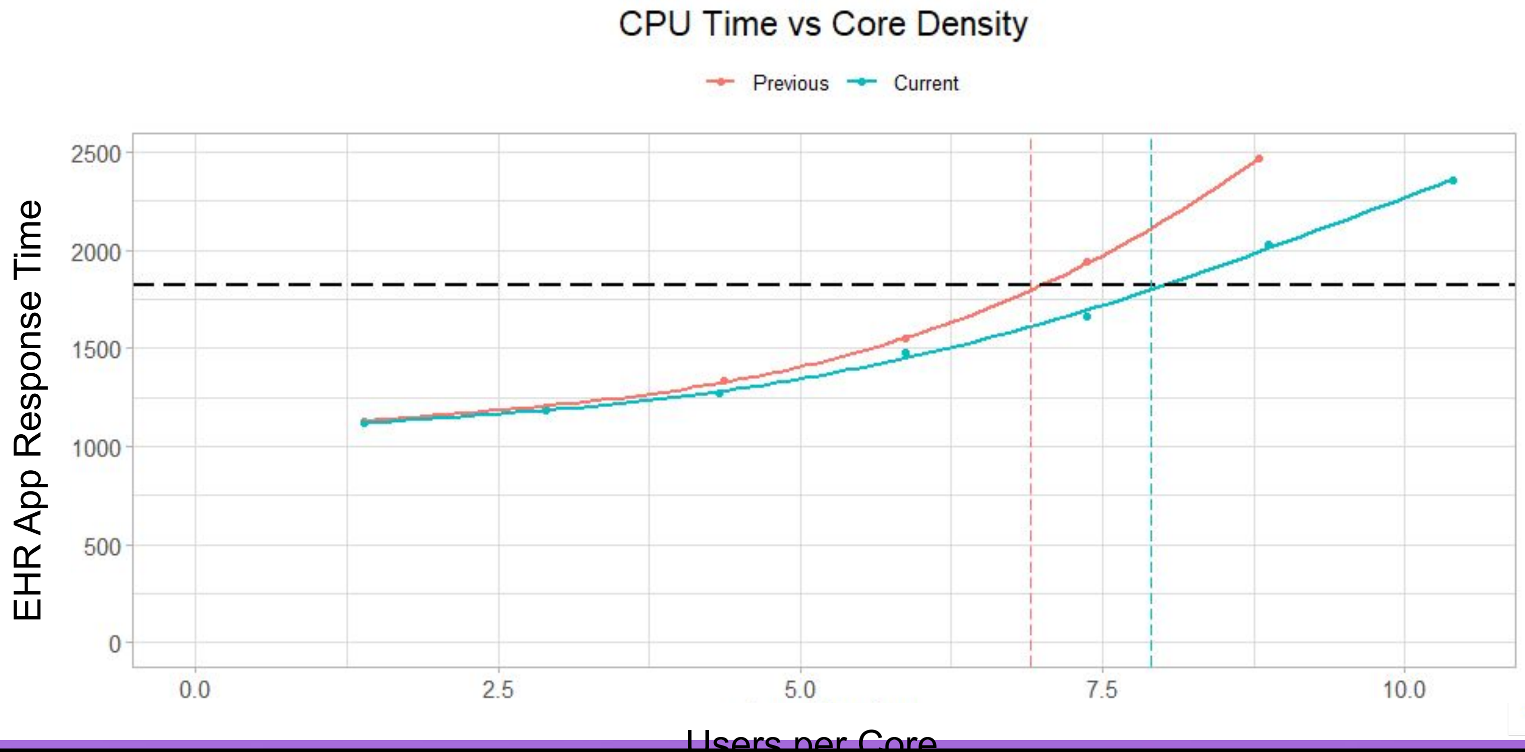
#3: xsave overhead still present, but issue now understood and fix inflight

#2a/b: Suppressed per-exit rdmsr for SPEC_CTRL on eIBRS systems and reverted backport causing host_debugctrlmsr to spam restore

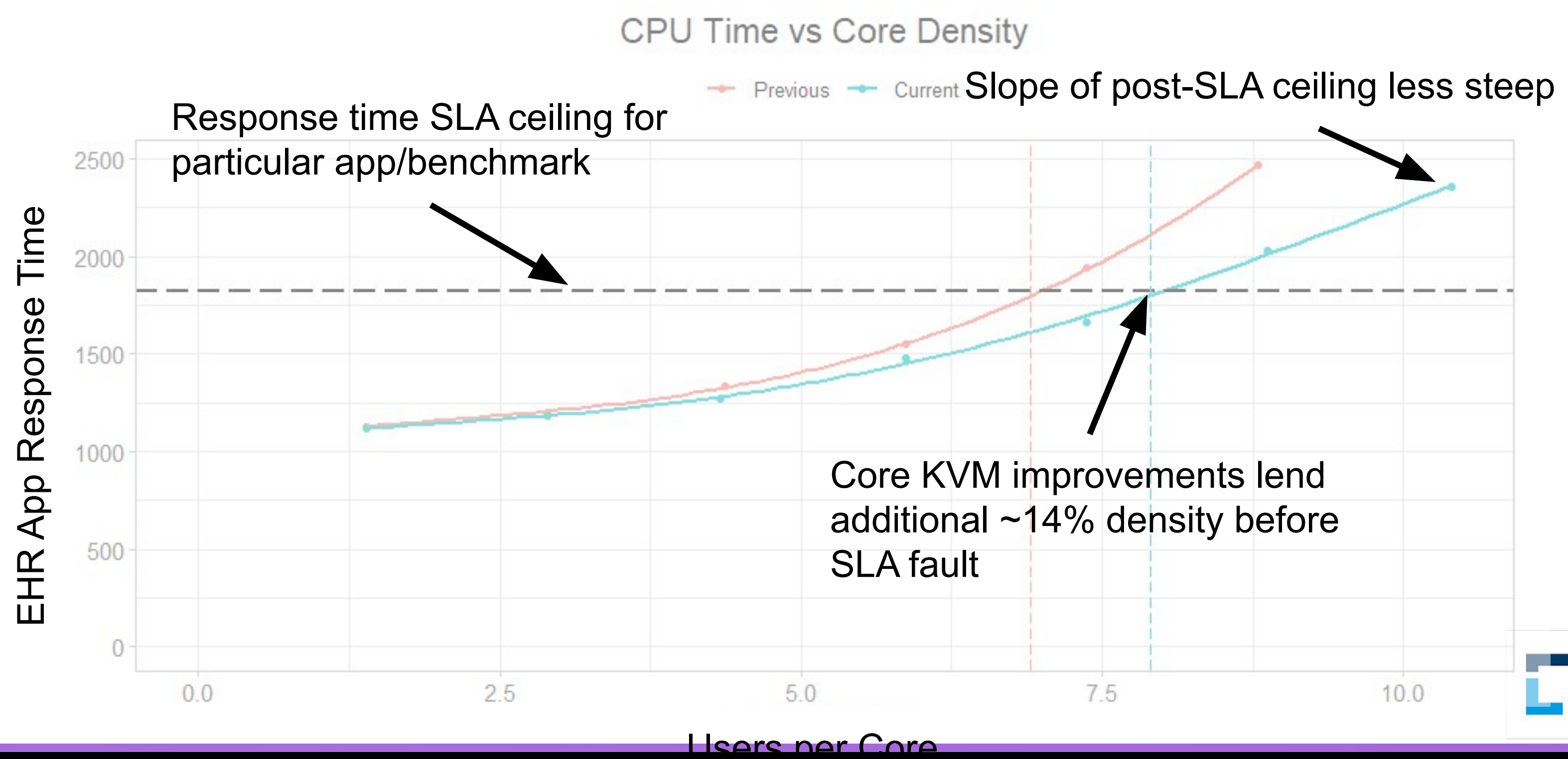
#1b: added exit fastpath for APIC_WRITE

#4: SPEC_CTRL overhead gone, backported bugs.c seccomp fix from upstream

Optimization Results: Improved EHR App Response Time



Optimization Results: Improved EHR App Response Time



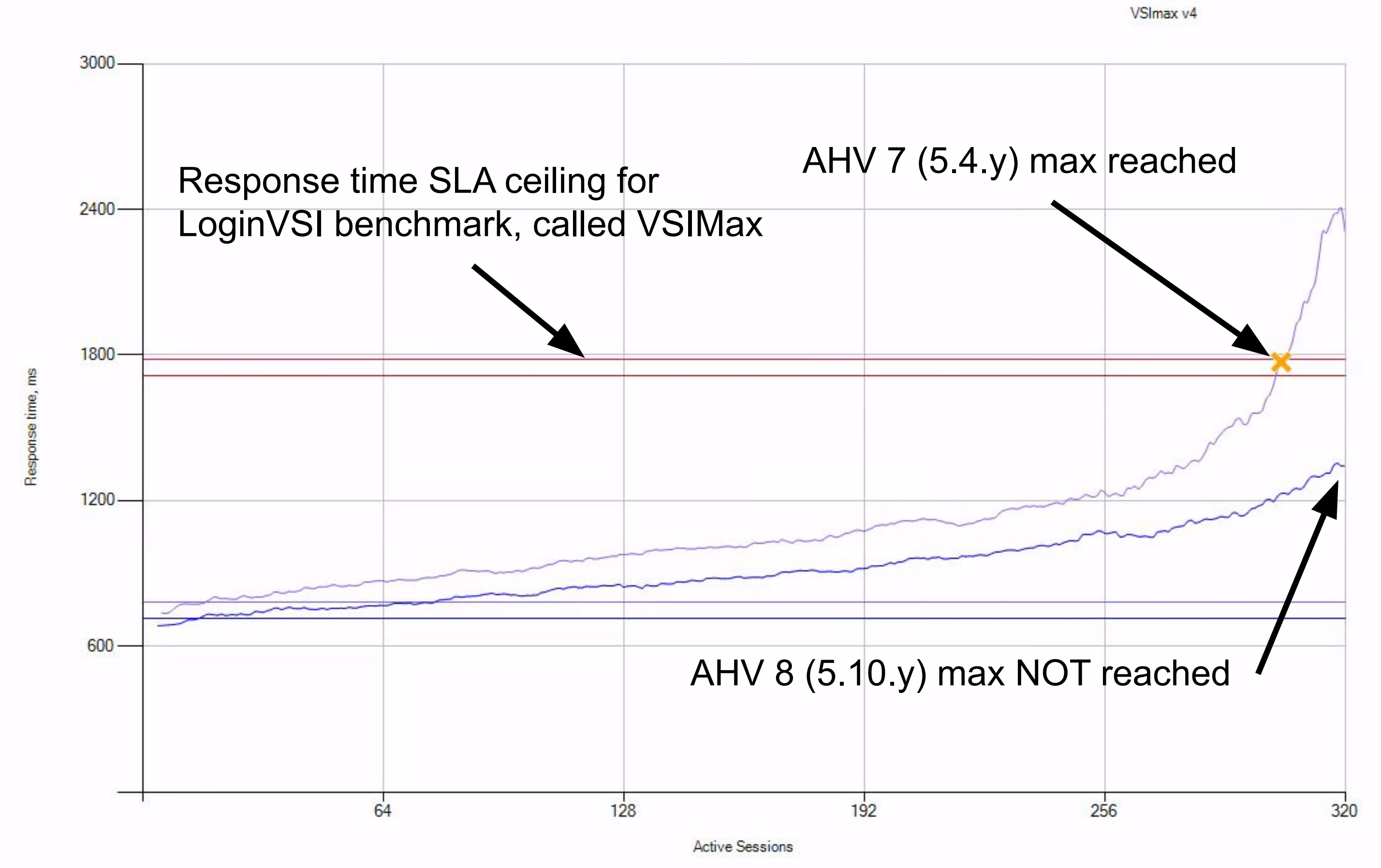
Optimization Results: Improved LoginVSI Density

Compared new kernel optimized to previous release

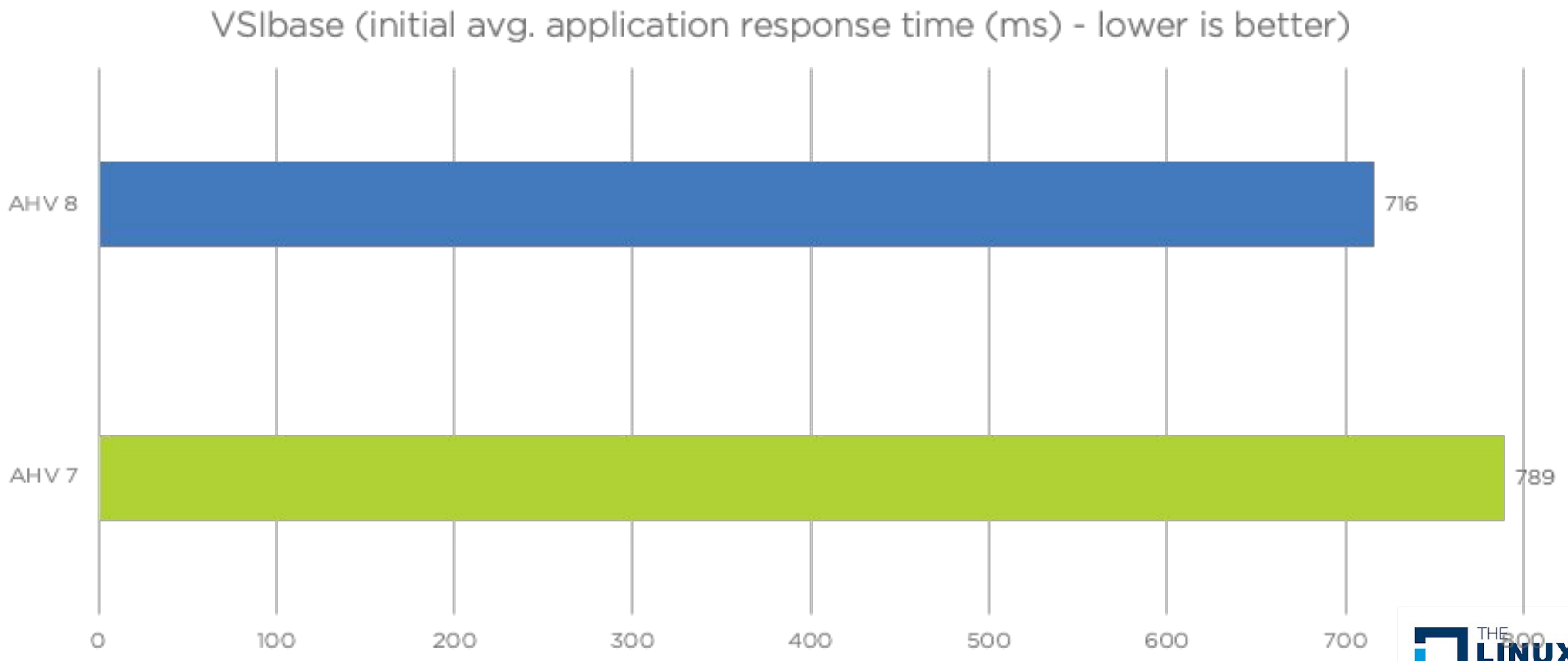
VMs: ~320 VMs/host
Windows 10 - 21H2 - updated July 2022, Office 2019 - x64
2 vCPU, 4 GB memory, LoginVSI 4.1

Hardware: Single node
CPU: 2x Intel(R) Xeon(R) Gold 6342 CPU @ 2.8GHz (Ice Lake)
MEM: 2TB
Disk: 6x 1.9TB SSD
AOS 6.5.1 with AHV 7 (5.4.y) vs AHV 8 (5.10.y with optimizations, minus hv-apicv)

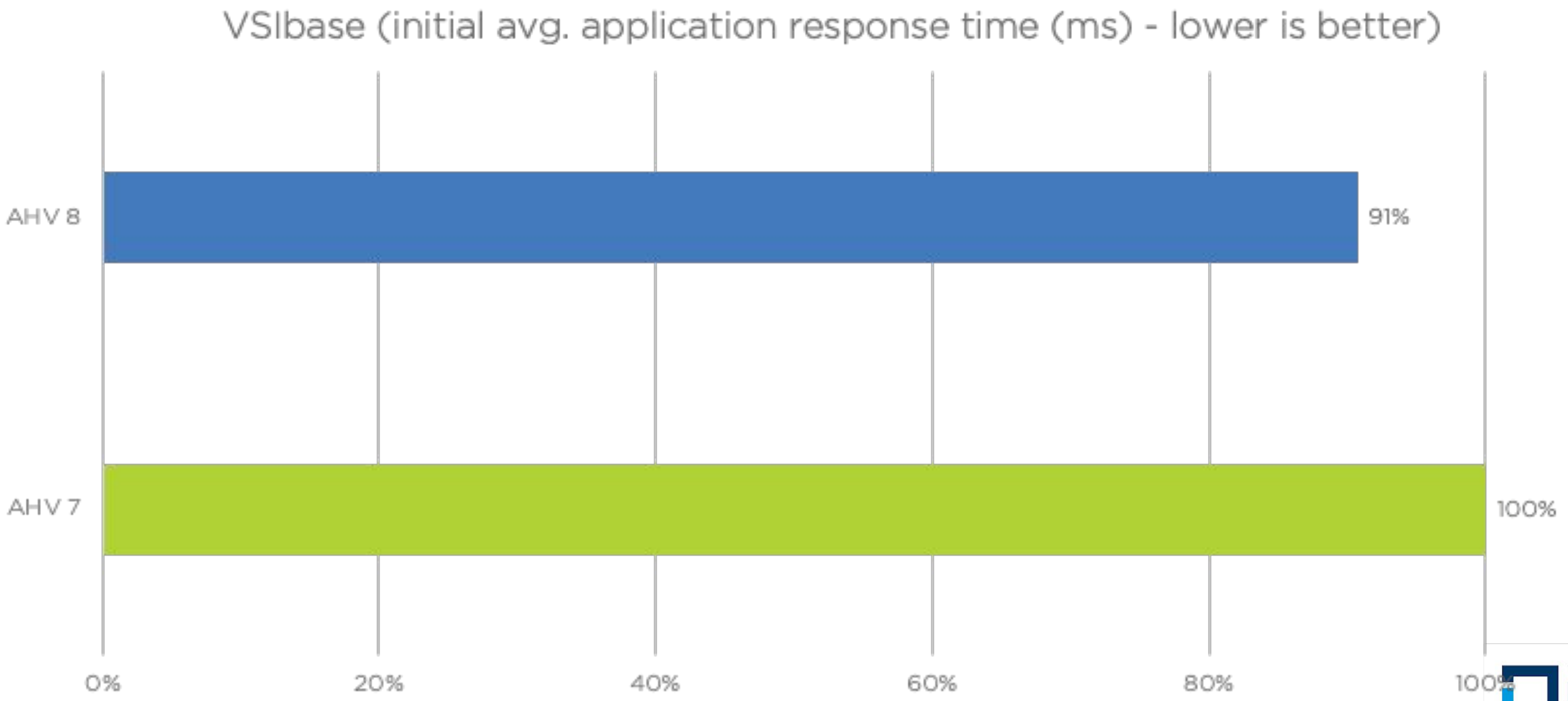
Optimization Results: Improved LoginVSI Density



Optimization Results: Improved LoginVSI Density

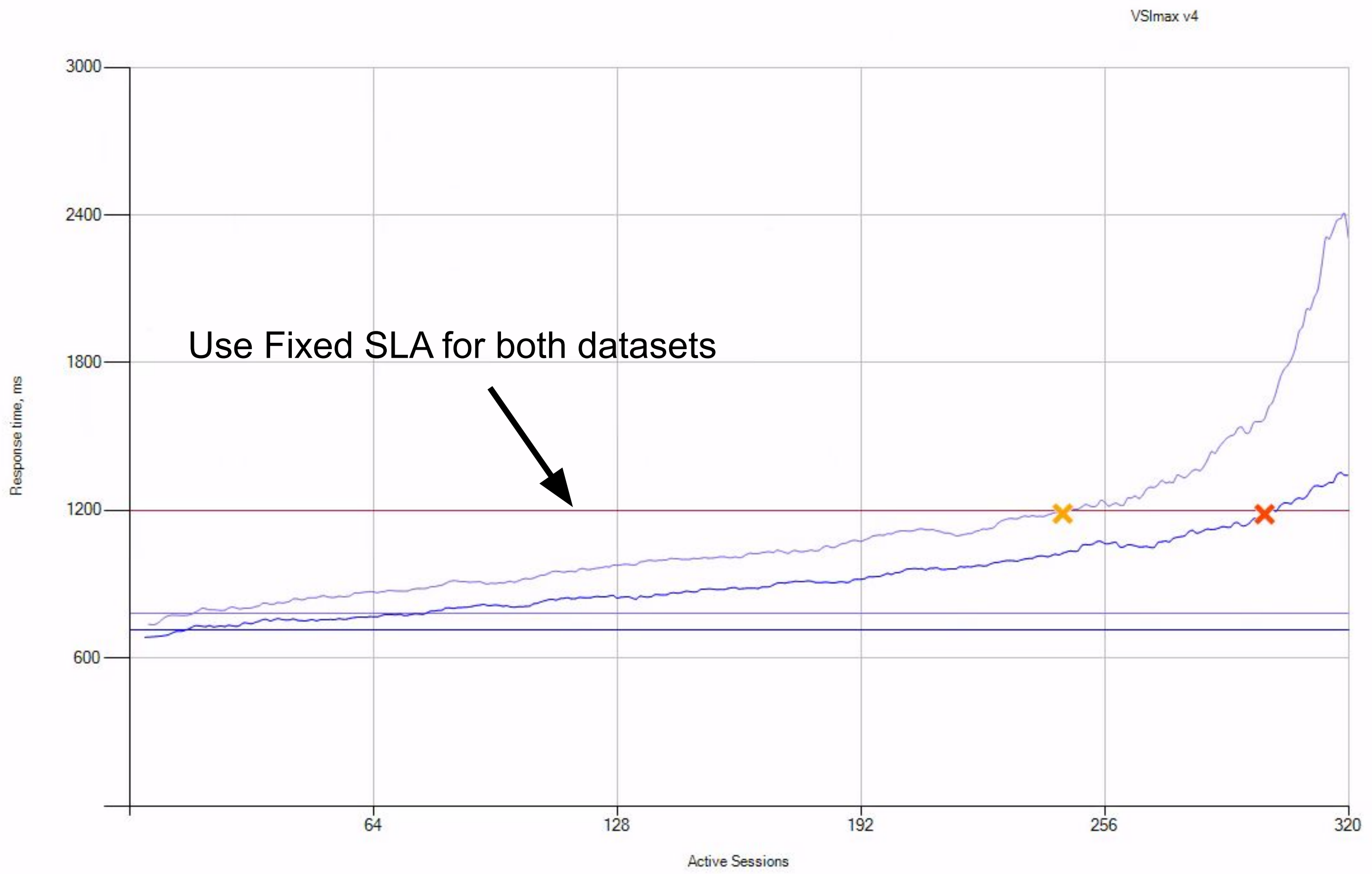


Optimization Results: Improved LoginVSI Density

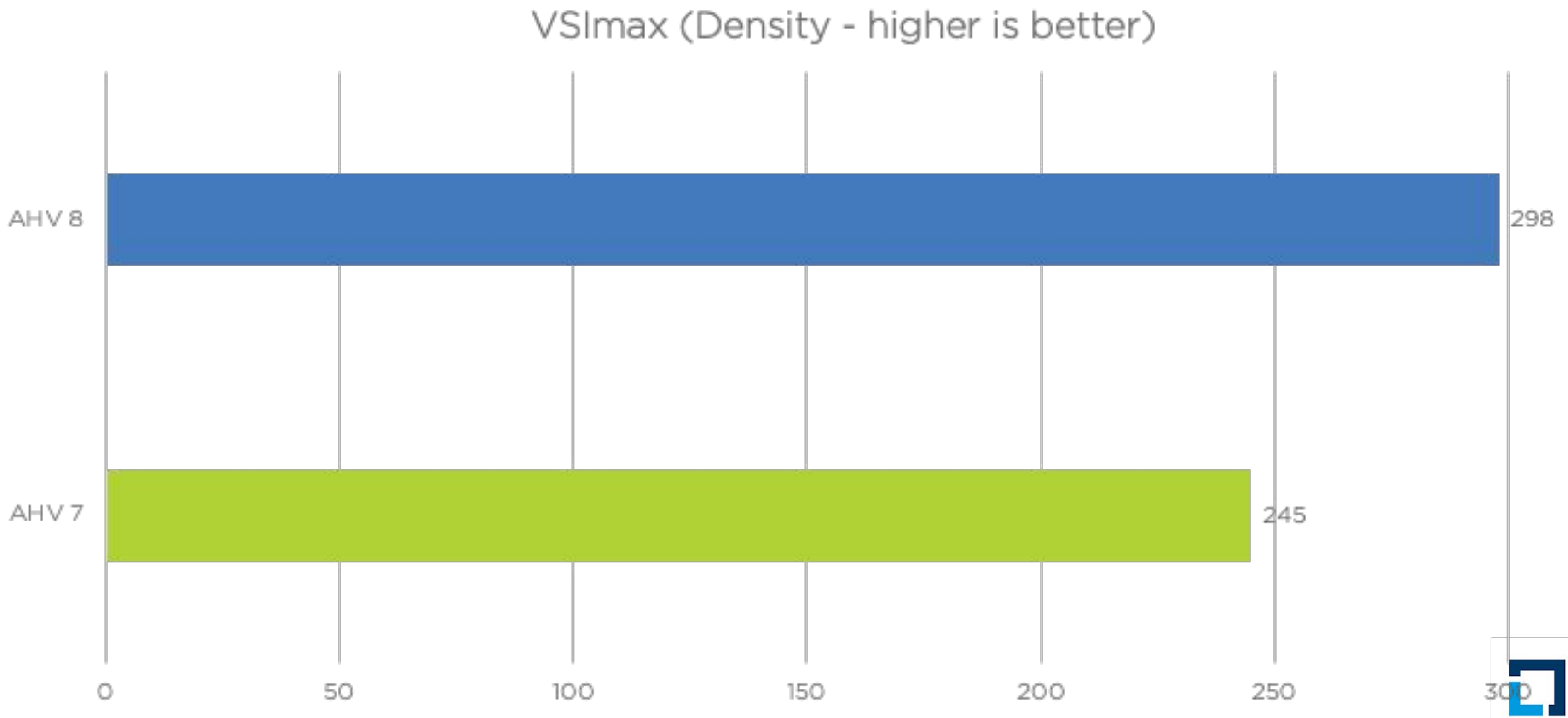


Optimization Results: Improved LoginVSI Density

Filtering LoginVSI Max upper threshold, like EHR did??

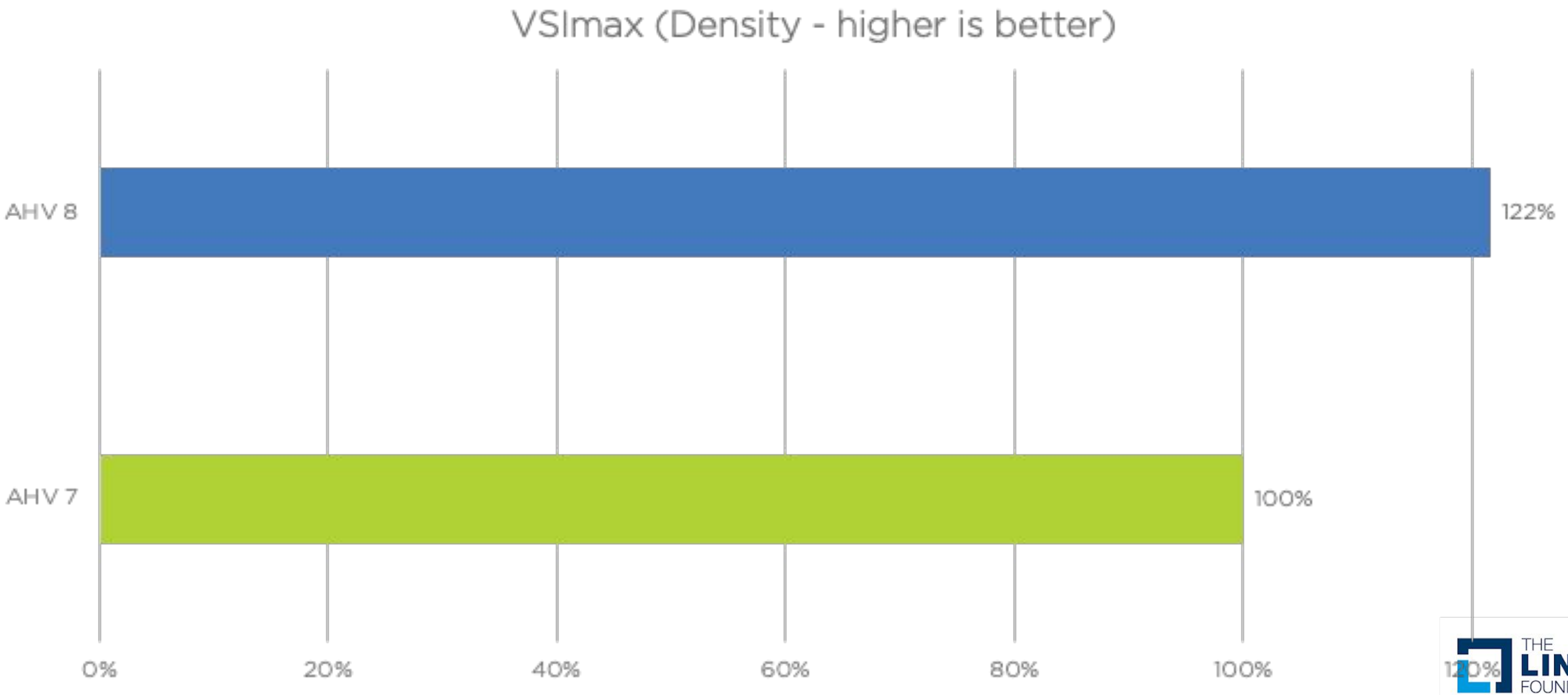


Optimization Results: Improved LoginVSI Density
What if we filter LoginVSI Max upper threshold?



THE LINUX FOUNDATION

Optimization Results: Improved LoginVSI Density
What if we filter LoginVSI Max upper threshold?



Zooming Out
Related Ecosystem Enhancements

Optimizations Found Along the Way
Related Ecosystem Enhancements

- Open vSwitch
  - Thundering Herd with handler* wakeups
  - Netlink stats gathering overhead suppression
- Linux VirtIO Driver
  - Enabling proper virtio_scsi MQ handling in RHEL 7.x
- Windows VirtIO Driver
  - Enabling large (256K+) IO sizes
  - VirtIO spec change: Indirect descriptor table size

Open vSwitch: thundering herd from handler* wakeups

Problem: Threading code in OVS [1] causes handler* threads to wake up concurrently in a thundering herd.

Result: Subtle nuance for CPU-sensitive EHR workloads is the herd may kick vCPUs off-cpu that recently went into halt-polling, due to **`single_task_running()`** exit condition on **`kvm_vcpu_can_poll()`**.

Issue: Reduction in file descriptors [1], allowed kernel to wake up more/all threads at once. Stumbled upon this using Google SchedViz (see next slide), was existing issue in RHBZ 1834444.

Fix: Backported Kernel [2] and OVS [3] series to add per-CPU upcall dispatch. ***Note: Both sides are required or per-cpu upcall dispatch will not work.***

Improvement: 28x reduction in wakeups, reduction in application tail latency due to more effective halt polling.

`[1] 69c51582ff78 ("dpif-netlink: don't allocate per thread netlink sockets") - OVS 2.11+`

`[2] b83d23a2a38b ("openvswitch: Introduce per-cpu upcall dispatch") - Kernel 5.15+`

`[3] b1e517bd2f81 ("dpif-netlink: Introduce per-cpu upcall dispatch.") - OVS 2.16+`

Open vSwitch: thundering herd from handler* wakeups

Scheduling collection by `local_user` on target on `Wednesday, May 25, 2022 at 11:57:07 AM GMT-04:00`

Threads | Events (74)

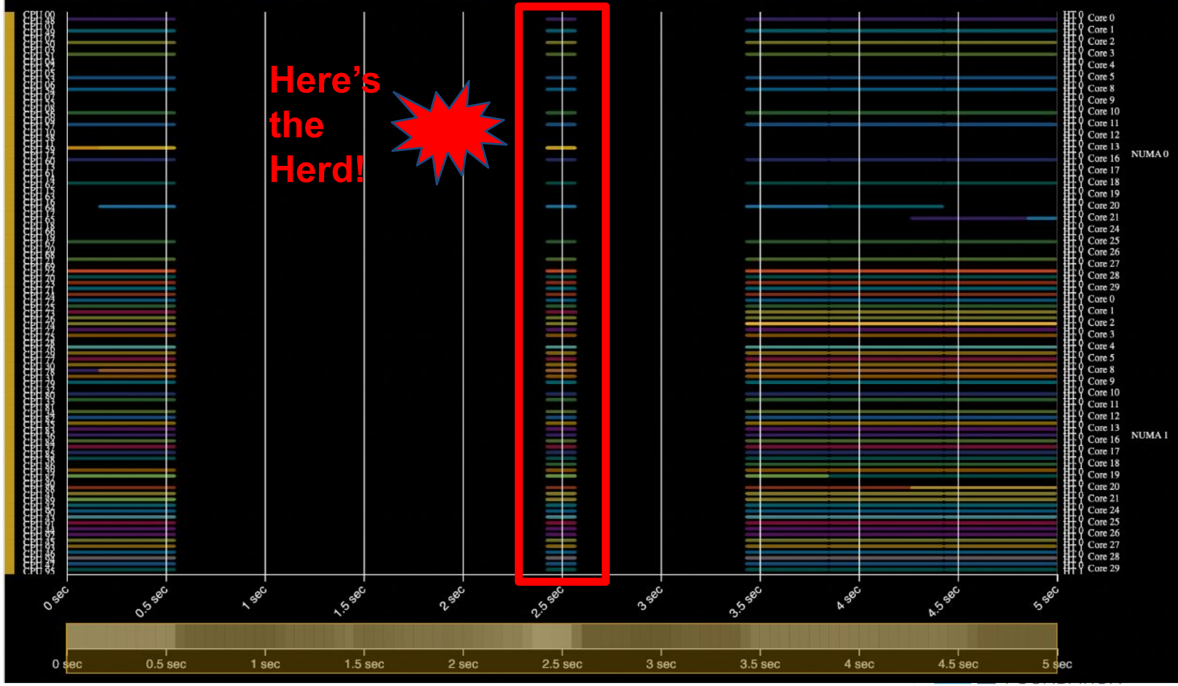
Filter PID | Command: handler

	PID	Command	1177 Wakeups ↓	12 Migrations	3.372 ms Waiting	10.381 ms Running	6.002 m Sleeping	0 ns Unknown
	3009	handler3	23	0	47.594 µs	482.793 µs	5.001 s	0 ns
	3011	handler4	22	0	45.265 µs	453.983 µs	5.001 s	0 ns
	3014	handler8	21	0	34.662 µs	279.645 µs	5.001 s	0 ns
	3012	handler6	20	0	28.674 µs	539.949 µs	5.001 s	0 ns
	3015	handler9	20	0	23.421 µs	292.186 µs	5.001 s	0 ns
	3013	handler7	19	0	40.894 µs	321.474 µs	5.001 s	0 ns
	3044	handler38	18	0	36.271 µs	92.651 µs	5.001 s	0 ns
	3050	handler44	18	0	28.412 µs	86.327 µs	5.002 s	0 ns
	3007	handler1	17	0	33.51 µs	1.555 ms	5.000 s	0 ns
	3045	handler39	17	0	31.172 µs	144.428 µs	5.001 s	0 ns
	3046	handler40	17	0	46.861 µs	85.559 µs	5.001 s	0 ns
	3047	handler41	17	0	31.577 µs	86.31 µs	5.002 s	0 ns
	3048	handler42	17	0	30.191 µs	79.171 µs	5.002 s	0 ns

Items per page: 13 1 - 13 of 72

Idle While Overloaded

Wall Time: 1.376 msec 0.028% | **Per CPU Time:** 1.414 msec 0.028% | **Per Thread Time:** 1.414 msec 0.028% | **CPU Utilization Fraction:** 11.04%



Open vSwitch: thundering herd from handler* wakeups

Before fixups: 1177 Wakeups

After fixups: 47 Wakeups

Result: 28x reduction in Wakeups!

Trivial reproduction from 5 second measurement during steady state, 1x VM reading from memory cache workload, very low network activity. Problem was significantly worse during activity.

Note: Both numbers include 5 wakeups from dbus_handler

Filter PID | Command

handler

PID	Command	1177 Wakeups	12 Migrations	3.372 ms Waiting	10.381 ms Running	6.002 m Sleeping	0 ns Unknown
3007	handler1	17	0	33.51 μs	1.555 ms	5.000 s	0 ns
3008	handler2	16	0	36.39 μs	837.78 μs	5.001 s	0 ns
3009	handler3	23	0	47.594 μs	482.793 μs	5.001 s	0 ns
3011	handler4	22	0	45.265 μs	453.983 μs	5.001 s	0 ns
3012	handler6	20	0	28.674 μs	539.949 μs	5.001 s	0 ns
3013	handler7	19	0	40.894 μs	321.474 μs	5.001 s	0 ns
3014	handler8	21	0	34.662 μs	279.645 μs	5.001 s	0 ns
3015	handler9	20	0	23.421 μs	292.186 μs	5.001 s	0 ns
3016	handler10	16	0	36.65 μs	75.688 μs	5.002 s	0 ns
3017	handler11	16	0	35.841 μs	72.613 μs	5.002 s	0 ns
3018	handler12	16	0	46.513 μs	105.052 μs	5.001 s	0 ns
3019	handler13	16	0	31.873 μs	74.722 μs	5.002 s	0 ns
3020	handler14	16	0	35.754 μs	69.838 μs	5.002 s	0 ns
3021	handler15	16	0	206.529 μs	91.888 μs	5.001 s	0 ns
3022	handler16	16	1	33.814 μs	160.321 μs	5.001 s	0 ns
3023	handler17	16	0	28.314 μs	73.164 μs	5.002 s	0 ns
3024	handler18	16	0	48.286 μs	109.268 μs	5.001 s	0 ns
3025	handler19	16	1	499.28 μs	107.458 μs	5.001 s	0 ns
3026	handler20	16	0	35.757 μs	118.58 μs	5.001 s	0 ns
3027	handler21	16	1	26.182 μs	88.445 μs	5.002 s	0 ns
3028	handler22	16	0	33.782 μs	68.877 μs	5.002 s	0 ns
3029	handler23	16	0	46.553 μs	94.739 μs	5.001 s	0 ns
3030	handler24	16	0	35.858 μs	68.281 μs	5.002 s	0 ns
3031	handler25	16	0	38.954 μs	69.057 μs	5.002 s	0 ns
3032	handler26	16	0	35.087 μs	97.503 μs	5.001 s	0 ns

Items per page: 25 1 - 25 of 72

Threads Events 3

Filter PID | Command

handler

PID	Command	47 Wakeups	2 Migrations	169.414 μs Waiting	6.939 ms Running	1.584 m Sleeping	0 ns Unknown
3022	handler7	1	0	1.671 μs	91.603 μs	5.002 s	0 ns
3030	handler14	4	0	17.194 μs	539.725 μs	5.001 s	0 ns
3031	handler15	2	0	9.658 μs	398.321 μs	5.001 s	0 ns
3032	handler16	1	0	2.135 μs	97.363 μs	5.002 s	0 ns
3033	handler17	1	1	1.837 μs	116.957 μs	5.002 s	0 ns
3039	handler23	1	0	1.687 μs	154.528 μs	5.002 s	0 ns
3048	handler32	1	0	3.856 μs	98.076 μs	5.002 s	0 ns
3052	handler36	1	0	4.831 μs	104.775 μs	5.002 s	0 ns
3058	handler42	1	0	2.566 μs	329.227 μs	5.001 s	0 ns
3061	handler45	4	0	7.667 μs	501.493 μs	5.001 s	0 ns
3066	handler50	1	0	3.52 μs	156.804 μs	5.002 s	0 ns
3067	handler51	12	0	62.389 μs	2.982 ms	4.999 s	0 ns
3071	handler55	1	0	4.534 μs	120.766 μs	5.002 s	0 ns
3073	handler57	3	0	5.862 μs	200.59 μs	5.001 s	0 ns
3087	handler71	1	0	3.512 μs	180.753 μs	5.002 s	0 ns
3088	handler72	3	1	4.738 μs	358.541 μs	5.001 s	0 ns
3089	handler73	3	0	6.544 μs	219.551 μs	5.001 s	0 ns
3090	handler74	1	0	11.414 μs	202.397 μs	5.001 s	0 ns
10602	dbus_handler	5	0	13.799 μs	85.15 μs	5.002 s	0 ns

Items per page: 25 1 - 19 of 19

OVS: netlink stats gathering overhead

Problem: OVS uses netlink to communicate with kernel. By default, for any netlink request, kernel gathers a bunch-o-stats to fill in response struct; however, the netlink request may not ***actually use*** said stats.

Result: ovs-vswitchd daemon constantly on-CPU, stealing cycles from CPU-sensitive EHR VMs.

Issue: Netlink call time dominated by the kernel-side internal stats gathering mechanism, specifically:

```
inet6_fill_link_af       <<< 42.2% of OVS' bridge_run() samples
  inet6_fill_ifla6_attrs
    __snmp6_fill_stats64
```

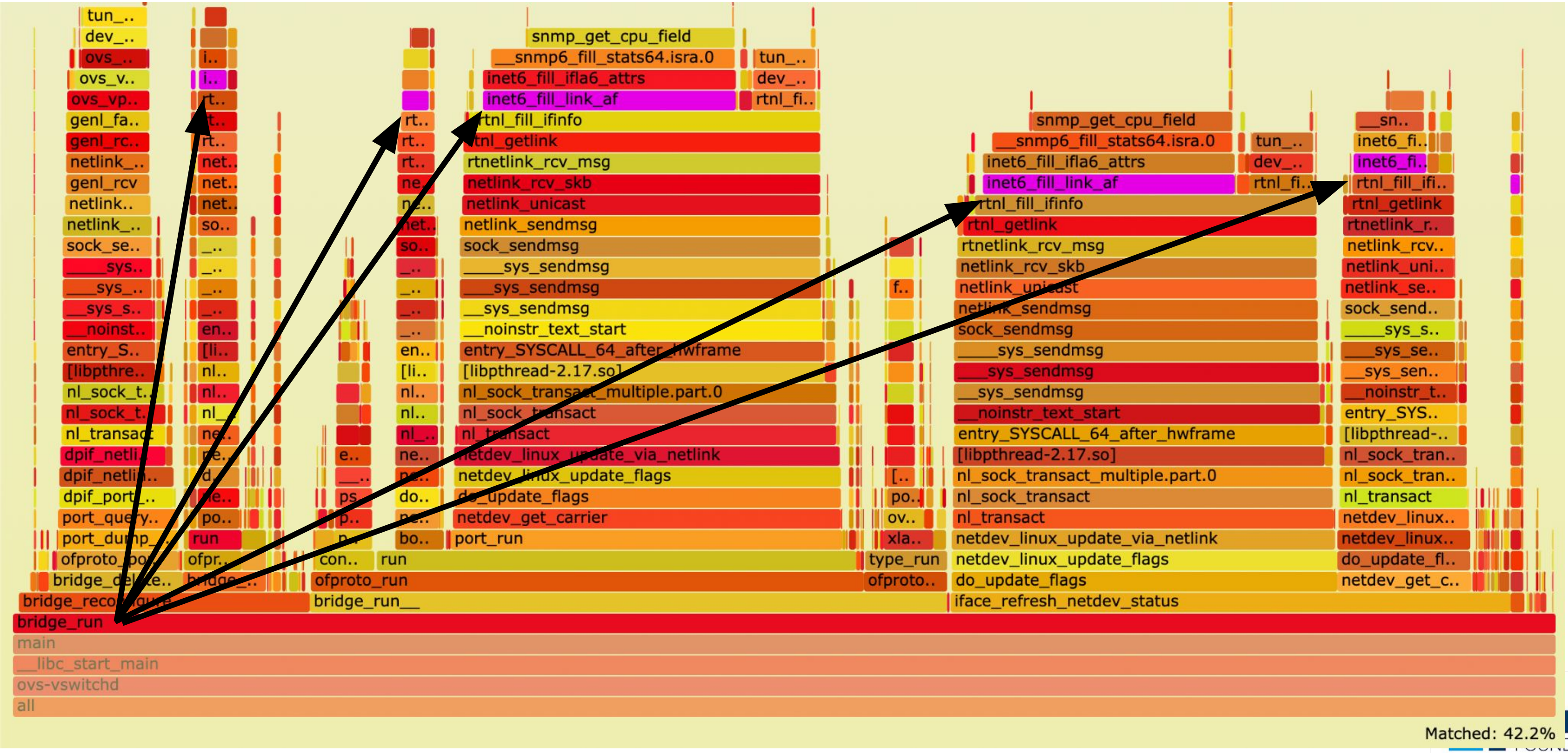
Fix: Patch [1] OVS to hint to kernel that certain netlink calls do not require stats gathering and backport [2] kernel fix to make the remaining stats gathering more efficient.

Improvement: Reduces amount of CPU samples during bootstorm in bridge_run() from 11.3 to 3.4%

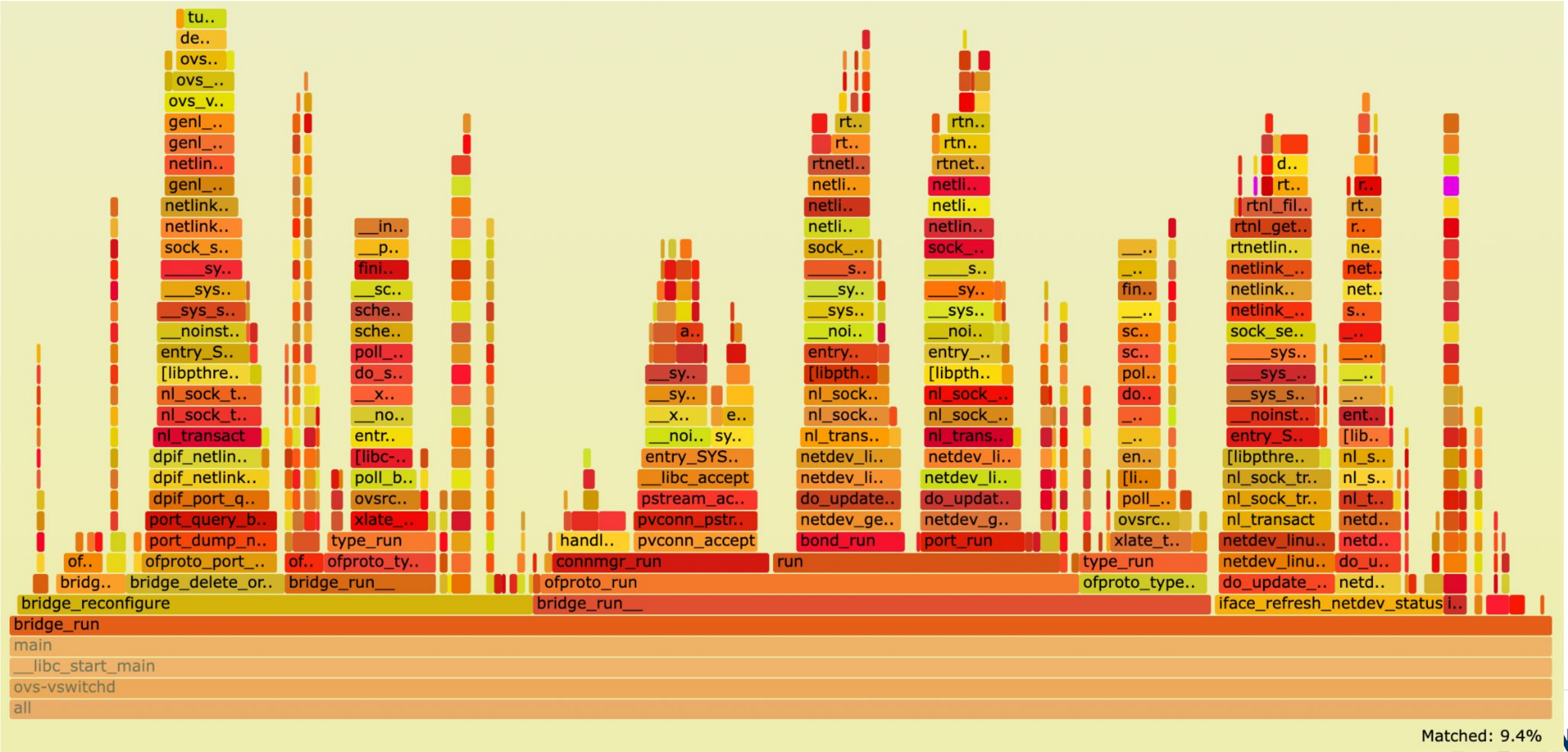
[1] c0e053f6d11d ("netdev-linux: Skip some internal kernel stats gathering")

[2] 59f09ae8fac4 ("net: snmp: inline snmp_get_cpu_field()") - Kernel 5.16+

OVS: netlink stats gathering overhead - Before



OVS: netlink stats gathering overhead - After



Linux VirtIO: Fixing virtio_scsi MQ on RHEL 7.x

Problem: Setting scsi_mod.use_blk_mq=y not sufficient to enable MQ on virtio-scsi MQ devices ***only*** on RHEL 7.x

Result: Broken load balancing to vhost-user-scsi backend, poor EHR max scalability

```
[root@host ~]# top
  PID USER        PR  NI  VIRT   RES   SHR S %CPU %MEM    TIME+   COMMAND
159638 qemu       20   0 64.2g   55m   10m S 1376.0  0.0  18216:38 qemu-kvm <<< Storage Controller
23920  qemu       20   0 48.1g   38m   10m S 1036.1  0.0 905:30.17 qemu-kvm <<< EHR Benchmark
23932  qemu       20   0 8192g   21m 1540 S 95.9    0.0 118:21.33 frodo    <<< vhost-user-scsi
[root@host ~]# top -H -b -n 1 |grep frodo
23932 qemu       20   0 8192g   21m 1540 S  0.0  0.0   0:00.45 frodo <<< control thread
23933 qemu       20   0 8192g   21m 1540 S  0.0  0.0  21:35.46 frodo <<< wk0
23934 qemu       20   0 8192g   21m 1540 R 96.2  0.0  28:38.67 frodo <<< wk1: all IO to single queue
```

Fix: Reported RH BZ 1752305, Fixed on RH Errata RHSA-2020:1016 (RHEL 7.8+, kernel-3.10.0.1127.el7+), RHEL kernel source was missing `ccbedf117f01` `("virtio_scsi: support multi hw queue of blk-mq")`

Windows VirtIO: large IO sizes & DB performance

Problem: One EHR vendor reporting DB benchmark stresses large IO for emulating high volume SQL DB backup and restore. Poor performance seen during these phases in particular, not consistent with Linux based reproductions.

Result: Customer go-live issues as benchmark pass is required for vendor sign off.

Issue: Defaults in virtio-win/vioscsi did not allow large contiguous IOs. Multiple attempts [1][2][3] at resolution didn't quite meet the mark and in fact caused both even worse performance and BSODs on our platform.

Fix: Upstreamed our fix [4] for off-by-one and max_sector handling to properly align IO sizes.

Improvement: Hit *100GbE line speed* during backup benchmark. Restore benchmark is smooth now.

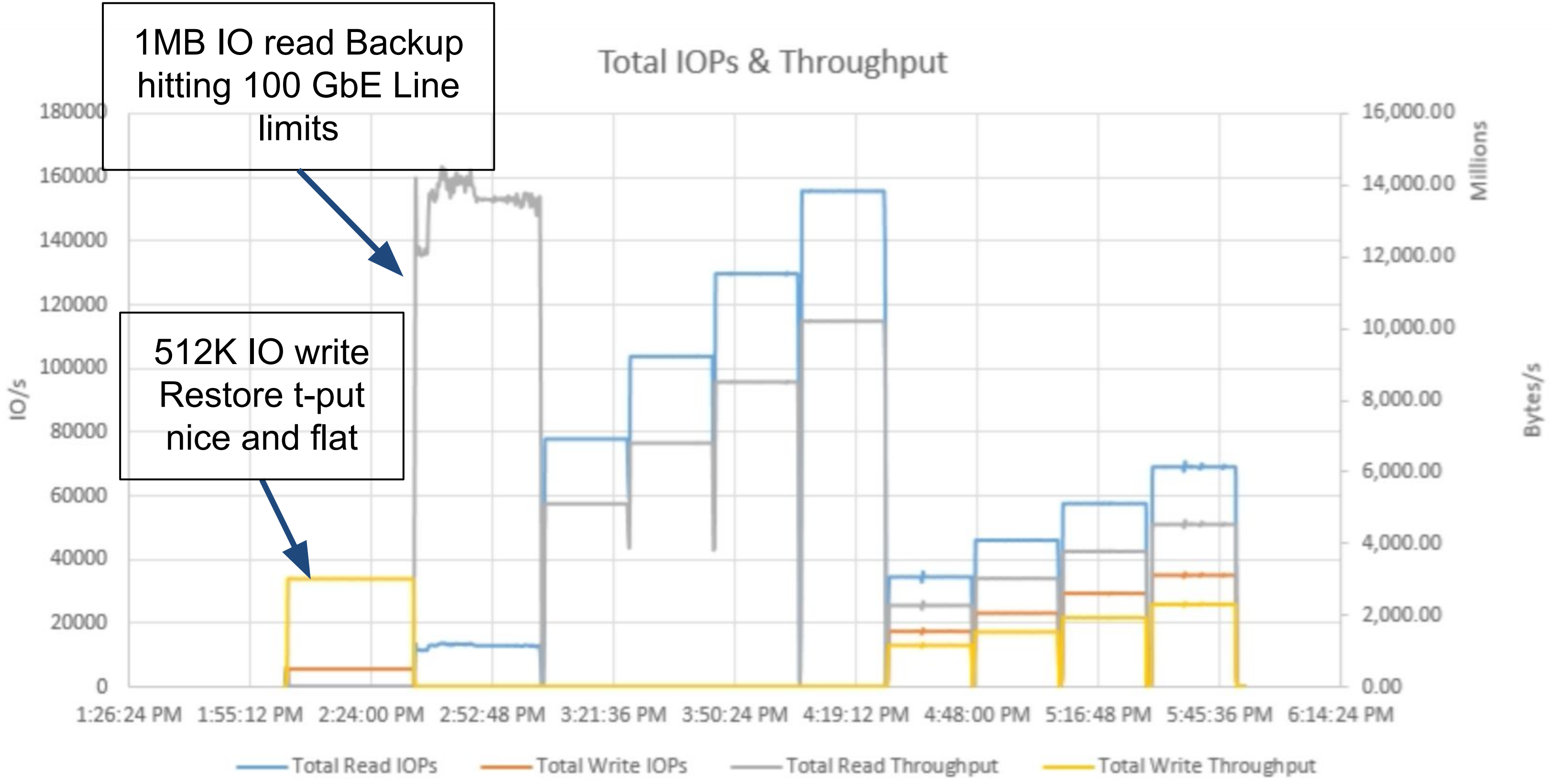
[1] 62e452b94b52 ("[vioscsi] Bug 1787022 - Windows virtio-scsi driver performs poorly ")

[2] c62a8a2c7bf7 ("vioscsi: Increasing max phys breaks to 512")

[3] 8a6ae70e2c7b ("[vioscsi] limit NumberOfPhysicalBreaks and MaximumTransferLength")

[4] 2c64f2af41bb ("vioscsi: fix MaximumTransferLength off-by-one and max_sectors handling") - source tag mm241 and higher, fedora virtio-win build 221 and higher

Windows VirtIO: large IO sizes & DB performance



Windows VirtIO: large IO sizes … the catch?!

Doing this actually ***should have*** required a VirtIO Spec Change!

VirtIO-win vioscsi driver violates VirtIO spec for maximum indirect descriptor size.

Windows 1MB IO takes 4K PAGE_SIZE * 256 SGL

256 SGL Indirect Desc Table Size ***should not*** be possible with VirtIO Spec, and default 128-2 ring size.

This surprise spec change broke our virtio-scsi-user implementation (we fixed it, at the cost of more memory allocation, matching QEMU)

Upstream Status: Pending VirtIO spec change

PR #122 Allow indirect descriptor tables to exceed the queue size

KVM FORUM

Appendix

Appendix
The Under the Covers Details for Debugging Example

Issue 1: Windows hv-apicv and APIC_WRITE fastpath

Problem: By default, Kernel/QEMU does not enable Intel vAPIC when base Windows enlightenments are enabled (e.g. `hv-stimer, hv-synic, hv-vapic`), deferring instead to Hyper-V's Synthetic Interrupt Controller. Even when `hv-apicv` enabled (not to be confused with `hv-vapic`), kernel still ***will not*** handle them in the fast path if the guest has less than 240 vCPUs.

Result: Without `hv-apicv`, Interrupts are emulated and handled by the kvm_emulate_wrmsr path, which does not live in the **`vmx_exit_handlers_fastpath()`**. With `hv-apicv`, IPI's are accelerated and trap-like; however, they are still not handled in exit fastpath, as EXIT_REASON is **`APIC_WRITE`**. Note: See Vitaly's talk Emulating Hyper-V in 2022 for more details on hv-apicv (aka hv-avic).

Fix: Enabled **`hv-apicv`** + add fastpath for **`EXIT_REASON_APIC_WRITE`** in **`vmx_exit_handlers_fastpath().`**

Note: There are Libvirt enablement issues (no `hv-apicv` support yet).

Note: Need to upstream APIC_WRITE fastpath kernel patch.

Issue 1: Windows hv-apicv and APIC_WRITE fastpath

arch/x86/kvm/vmx/vmx.c

```
static fastpath_t vmx_exit_handlers_fastpath(struct kvm_vcpu *vcpu)
        switch (to_vmx(vcpu)->exit_reason.basic) {
        case EXIT_REASON_MSR_WRITE:              <<< Handles Linux + Large Windows VMs
                return handle_fastpath_set_msr_irqoff(vcpu);
+       case EXIT_REASON_APIC_WRITE:             <<< Handles “Small” Windows VMs
+               if (kvm_vcpu_apicv_active(vcpu)) {
+                       handle_apic_write(vcpu);
+                       return EXIT_FASTPATH_EXIT_HANDLED;
+               } else {
+                       return EXIT_FASTPATH_NONE;
+               }
        case EXIT_REASON_PREEMPTION_TIMER:
                return handle_fastpath_preemption_timer(vcpu);
        default:
```

Issue 2: vmx_vcpu_run SPEC_CTRL rdmsr overhead

Problem: Guests that use eIBRS write to SPEC_CTRL MSR 1-2 times on boot, then never again; however, kernel disables SPEC_CTRL interception in MSR bitmap unilaterally, giving guest direct access to MSR. This is done to avoid an exit on every IBRS write, which was key for IBRS performance, but is moot for eIBRS “one-and-done” enablement pattern.

Result: When interception is disabled, kernel must rdmsr SPEC_CTRL MSR on every single exit, which is roughly 40-50% of the “flat top” in **`vmx_vcpu_run()`**. Note, this happens within **`vmx_vcpu_enter_exit()`** as part of the guest return, but prior to exit fastpath, so cycles spent here delay handling IPI delivery.

Fix: For guests that enable eIBRS, do not disable interception in MSR bitmap [1], negating need to do rdmsr.

[1] [PATCH] [v3] KVM: VMX: do not disable interception for MSR_IA32_SPEC_CTRL on eIBRS

Issue 2: vmx_vcpu_run and debugctl regression

Problem: debugctl value is cached on vCPU load, as it architecturally cleared on vmexit and would need to be restored if set by host prior to load. On many/most systems, debugctl isn't set during steady state; however, a commit [1] in 5.17, backported through stable to 5.10, turns this on constantly.

Result: 30-40% of the "flat top" in **`vmx_vcpu_run()`** is due to constantly resetting debugctl msr from cached value. Note, this happens immediately after **`vmx_vcpu_enter_exit()`** returns, but prior to exit fastpath, so cycles spent here delay handling IPI delivery.

Fix: For us, reverting was the cleanest route, as we aren't exposing the system to the use cases outlined in the commit.

`[1] a01994f5e5c7 ("x86/perf: Default set FREEZE_ON_SMI for all")`

Issue 3: ‘Mystery’ XSAVE overhead, Demystified

Problem: Nutanix control plane auto masks CPU features to handle migration compatibility, including a deny list. Deny list includes `MPX` and `PKU`, which influence xstate features. We need to mask it to maintain migration compatibility across Ice Lake and non-Ice Lake. PKU masked due to handling bug (long ago).

Result: Masking XSAVE-able features changes the XSAVE mask, so every single pass through `kvm_load_{guest|host}_xsave_state()` spams `xsetbv`, delaying time-to-enter and time-to-IPI handling.

Fix: Fixup host side mask calculation in very early code to fully compile out MPX and PKU from host kernel to make xstate feature masks match. This also gets rid of `rdpkru/wrpkru` from *xsave_state() calls too.

Note: Need to upstream patch series for review.

Note: See Soham and Shivam’s talk on CPU Feature Management: Lessons from the Trenches for more discussion on more learnings in this space.

Issue 3: Mismatched xstate features: Host XSAVE

Host XSAVE state loads very early, and even compiling out features doesn't change early code masking.

```
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x008: 'MPX bounds registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x010: 'MPX CSR'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x020: 'AVX-512 opmask'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x040: 'AVX-512 Hi256'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x080: 'AVX-512 ZMM_Hi256'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x200: 'Protection Keys User registers'
[    0.000000] x86/fpu: xstate_offset[2]:  576, xstate_sizes[2]:  256
[    0.000000] x86/fpu: xstate_offset[3]:  832, xstate_sizes[3]:   64
[    0.000000] x86/fpu: xstate_offset[4]:  896, xstate_sizes[4]:   64
[    0.000000] x86/fpu: xstate_offset[5]:  960, xstate_sizes[5]:   64
[    0.000000] x86/fpu: xstate_offset[6]: 1024, xstate_sizes[6]:  512
[    0.000000] x86/fpu: xstate_offset[7]: 1536, xstate_sizes[7]: 1024
[    0.000000] x86/fpu: xstate_offset[9]: 2560, xstate_sizes[9]:    8
[    0.000000] x86/fpu: Enabled xstate features 0x2ff, context size is 2568 bytes, using 'compacted' format.
```

Issue 3: Mismatched xstate features: Guest XSAVE

Control Plane automatically masks MPX and PKU feature sets, which are on deny list.

```
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
                        <<< Missing MPX here
[    0.000000] x86/fpu: Supporting XSAVE feature 0x020: 'AVX-512 opmask'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x040: 'AVX-512 Hi256'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x080: 'AVX-512 ZMM_Hi256'
                        <<< Missing PKU here
[    0.000000] x86/fpu: xstate_offset[2]:  576, xstate_sizes[2]:  256
[    0.000000] x86/fpu: xstate_offset[5]:  832, xstate_sizes[5]:   64
[    0.000000] x86/fpu: xstate_offset[6]:  896, xstate_sizes[6]:  512
[    0.000000] x86/fpu: xstate_offset[7]: 1408, xstate_sizes[7]: 1024
[    0.000000] x86/fpu: Enabled xstate features 0xe7, context size is 2432 bytes, using 'compacted' format
```

Issue 3: Mismatched forever, pain in vcpu_run() loop

arch/x86/kvm/x86.c

```
void kvm_load_guest_xsave_state(struct kvm_vcpu *vcpu) {         <<< Called before vmx_vcpu_enter_exit()
   if (kvm_read_cr4_bits(vcpu, X86_CR4_OSXSAVE)) {
       if (vcpu->arch.xcr0 != host_xcr0)                         <<< Branch hit: Guest 0xe7 != Host 0x2ff
           xsetbv(XCR_XFEATURE_ENABLED_MASK, vcpu->arch.xcr0);   <<< Pain on vm enter, delays enter
   }
}

void kvm_load_host_xsave_state(struct kvm_vcpu *vcpu) {   <<< Called after vmx_vcpu_enter_exit() returns
   if (kvm_read_cr4_bits(vcpu, X86_CR4_OSXSAVE)) {
       if (vcpu->arch.xcr0 != host_xcr0)                  <<< Branch hit: Guest 0xe7 != Host 0x2ff
           xsetbv(XCR_XFEATURE_ENABLED_MASK, host_xcr0);  <<< Pain on vm exit, delays exit handling
   }
}
```

Issue 4: wicked x86_virt_spec_ctrl overhead

Problem: In QEMU 2.11+, -sandbox on is enabled by default [1], which turns on seccomp. If KVM host has CONFIG_SECCOMP=y, all SEECOMP jails enable TIF_SPEC_IB and TIF_SSBD, due to spectre_v2=auto applying to both prctl and seccomp.

Result: Almost always, the guest and host SPEC_CTRL values will not match, resulting in each entry/exit spamming wrmsr SPEC_CTRL to constantly reset the value. Wrmsr to SPEC_CTRL stalls pipeline completely.

Fix: Backport [2] change in default from 5.16, **HUGE** tax cut, as host/guest much more likely to match.

[1] RHBZ 1492597 - Enable seccomp by out of the box with QEMU >= 2.11

[2] 2f46993d83ff (“x86: change default to spec_store_bypass_disable=prctl spectre_v2_user=prctl”) 5.16+

(Thank you Andrea Arcangeli !!)

Issue 4: wicked x86_virt_spec_ctrl overhead

Host dmesg extract

```
[    4.294333] Spectre V2 : mitigation: Enabling conditional Indirect Branch Prediction Barrier
[    4.294333] Spectre V2 : User space: Mitigation: STIBP via seccomp and prctl
[    4.294335] Speculative Store Bypass: Mitigation: Speculative Store Bypass disabled via prctl and seccomp
```

Process status of qemu-kvm VM

```
[root@mauricio06 ~]# cat /proc/5733/status
Name:  qemu-kvm
Seccomp:    2
Seccomp_filters: 1
Speculation_Store_Bypass:   thread force mitigated
```

Overhead via perf top

```
  6.11%  [kernel]      [k] x86_virt_spec_ctrl <<< BRUTAL!
```

Appendix
New to Flamegraphs?

Profiling C/C++: Linux Perf & Flame Graphs

USAGE

- **How**: Use Linux Perf to sample service (or entire system) and run through steps to convert to Flame graph svg.
- **What**: Study flat tops, understand everything with >1% sample size, diff flame graphs while iterating, use different on-cpu, off-cpu, different perf trace points like cache misses, etc (see Flame Graph docs, presos, vids).
- ***Watch out for [Unknown] frames.***
- **Note**: perf.data output can be used for other things ***besides*** flame graphs, so it is valuable.

THE LINUX FOUNDATION

Profiling C/C++: Linux Perf & Flame Graphs

PROS

- Very easy to grab in minute(s), provides fantastic insights to what's on-CPU.
- Easy to understand, Easy to use (searchable).
- Easy to manipulate to show data how you might like, ala different merging.
- Methodology reusable across many languages.

CONS

- Linux perf has a zillion options, easy to get lost on non-important things.
- Requires ***as much symbolization as possible*** as it is out of process, so if 3rd party component is on-CPU with no frames or symbols, you're blind.

Profiling C/C++: Linux Perf & Flame Graphs

Dependencies

```
wget https://github.com/brendangregg/FlameGraph/archive/master.zip
unzip master.zip
```

Grabbing profile

```
sudo perf record -F 997 -a -g -- sleep 10
sudo perf script -f > example.perf
```

Massage Data (and merge related stacks as needed)

```
./FlameGraph-master/stackcollapse-perf.pl example.perf > example.folded
./FlameGraph-master/flamegraph.pl example.folded > example-separate.svg
sed -i 's/CPU_\([0-9]*\)/CPU_merged/g' example.folded
sed -i 's/handler\([0-9]*\)/handler-merged/g' example.folded
sed -i 's/revalidator\([0-9]*\)/revalidator-merged/g' example.folded
sed -i 's/vhost-\([0-9]*\)/vhost-merged/g' example.folded
./FlameGraph-master/flamegraph.pl example.folded > example-merged.svg
```

Viewing profile

```
scp user@host:example-separate.svg . ## Open in Chrome - original flamegraph
scp user@host:example-merged.svg .   ## Open in Chrome - merged flamegraph
```