Joyent

Experiences Porting KVM to SmartOS

Bryan Cantrill
VP, Engineering

bryan@joyent.com
@bcantrill

WTF is SmartOS?

- illumos-derived OS that is the foundation of both Joyent's public cloud and SmartDataCenter product
- As an illumos derivative, has several key features:
  - ZFS: Enterprise-class copy-on-write filesystem featuring constant time snapshots, writable clones, built-in compression, checksumming, volume management, etc.
  - DTrace: Facility for dynamic, *ad hoc* instrumentation of production systems that supports *in situ* data aggregation, user-level instrumentation, etc. — and is absolutely safe
  - OS-based virtualization (Zones): Entirely secure virtual OS instances offering hardware performance, high multi-tenancy
  - Network virtualization (Crossbow): Virtual NIC Infrastructure for easy bandwidth management and resource control

KVM on SmartOS?

- Despite its rich feature-set, SmartOS was missing an essential component: hardware virtualization
- Thanks to Intel and AMD, hardware virtualization can now be remarkably high performing...
- We firmly believe that the best hypervisor is the operating system — anyone attempting to implement a “thin” hypervisor will end up retracing OS history
- KVM shares this vision — indeed, pioneered it!
- Moreover, KVM is best-of-breed: highly competitive performance and a community with critical mass
- Imperative was clear: needed to port KVM to SmartOS!

Constraining the port

Joyent

- For business and resourcing reasons, elected to focus exclusively on Intel VT-x with EPT...
- ...but to not make decisions that would make later AMD SVM work impossible
- Only ever interested in x86-64 host support
- Only ever interested in x86 and x86-64 guests
- Willing to diverge as needed to support illumos constructs or coding practices…
- ...but wanted to maintain compatibility with QEMU/KVM interface as much as possible

Starting the port

- KVM was (rightfully) not designed to be portable in any real sense — it is specific to Linux and Linux facilities
- Became clear that emulating Linux functionality would be insufficient — there is simply too much divergence
- Given the stability of KVM in Linux 2.6.34, we felt confident that we could diverge from the Linux implementation — while still being able to consume and contribute patches as needed
- Also clear that just getting something to *compile* would be a significant (and largely serial) undertaking
- Joyent engineer Max Bruning started on this in late fall...

Getting to successful compilation

- To expedite compilation, unported blocks of code would be "XXX'd out" by being enclosed in `#ifdef XXX`
- To help understand when/where we hit XXX'd code paths, we put a special DTrace probe with `__FILE__` and `__LINE__` as arguments in the `#else` case
- We could then use simple DTrace enablings to understand what of these cases we were hitting to prioritize work:

```
kvm-xxx
{
        @[stringof(arg0), probefunc, arg1] = count();
}

tick-10sec
{
        printf("%-12s %-40s %-8s %8s\n",
            "FILE", "FUNCTION", "LINE", "COUNT");
        printa("%20s %8d %@8d\n", @);

}
```

Accelerating the port

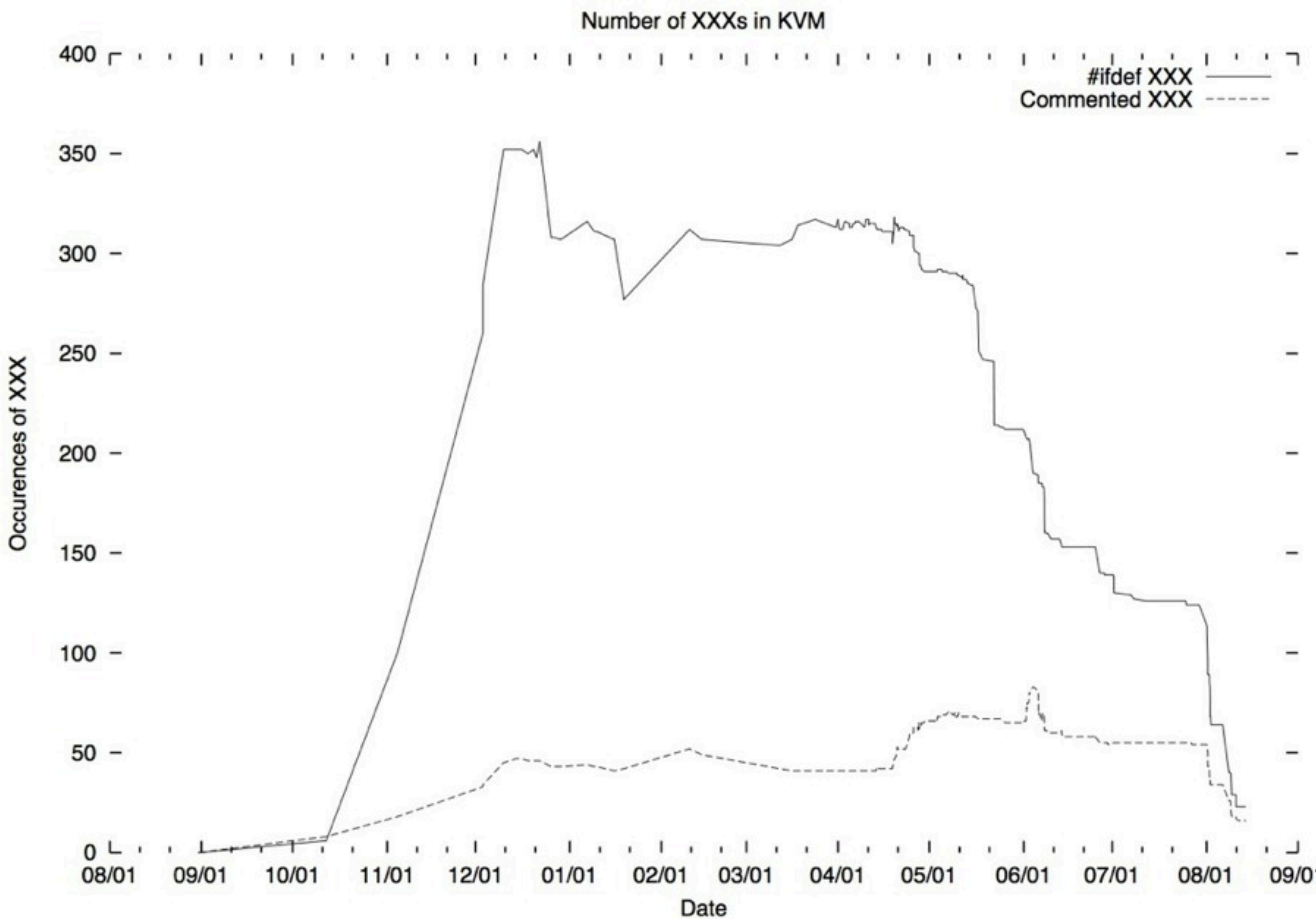
- By late March, Max could launch a virtual machine that could run in perpetuity without panicking…
- ...but also was not making any progress booting
- At this point, the work was more readily parallelized: Joyent’s Robert Mustacchi and I joined Max in April
- Added tooling to understand guest behavior, e.g.:
  - MDB support to map guest PFNs to QEMU VAs
  - MDB support for 16-bit disassembly (!)
  - DTrace probes on VM entry/exit and the ability to pull VM state in DTrace with a new `vmregs[]` variable

Making progress...

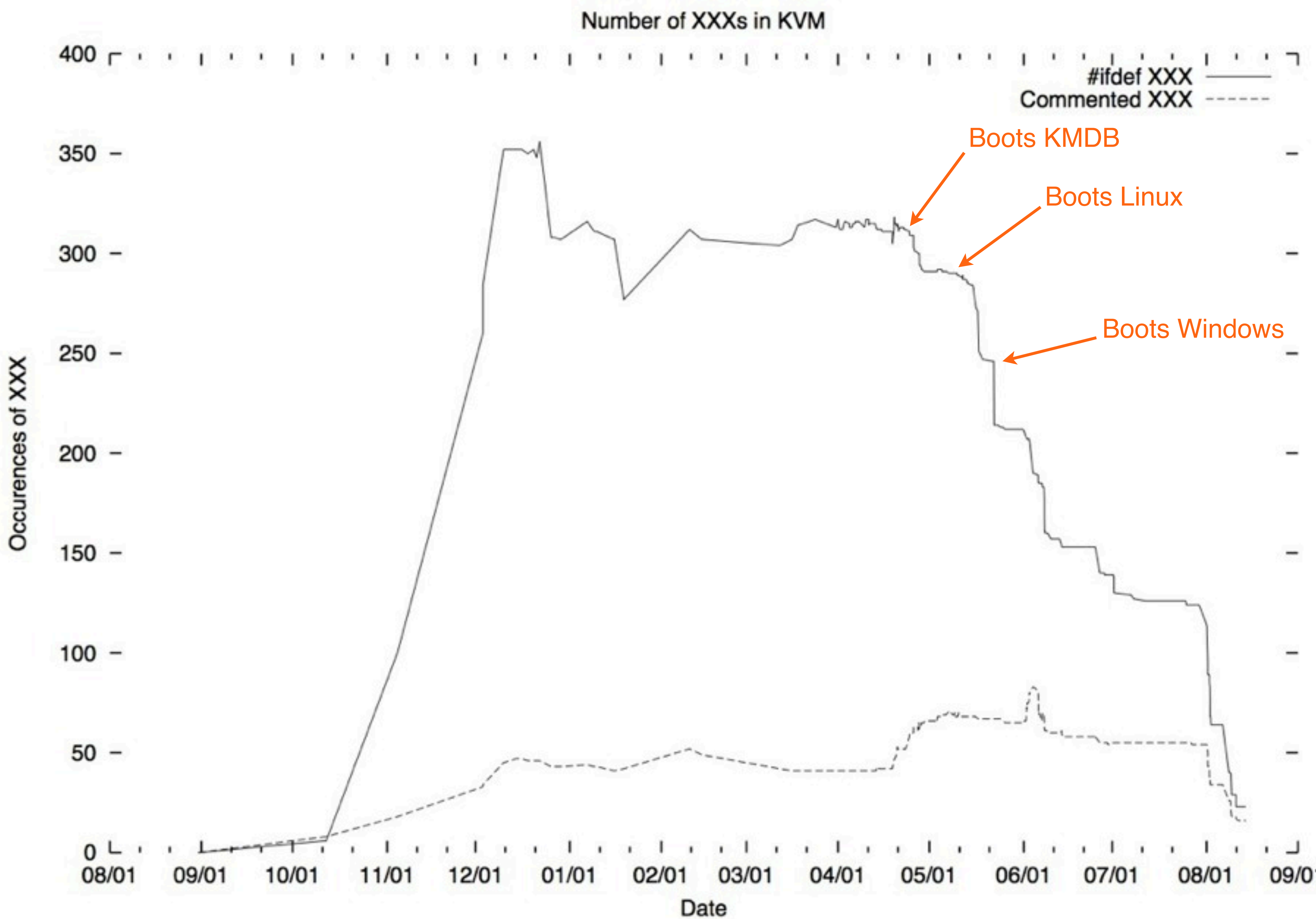
Joyent

- To make forward progress, we would debug the issue blocking us (inducing either guest or host panic)…
- ...which was usually due to a piece that hadn’t yet been ported or re-implemented
- We would implement that piece (usually eliminating an XXX’d block in the process), and debug the next issue
- The number of XXX’s over time tell the tale...

The tale of the port



Port milestones

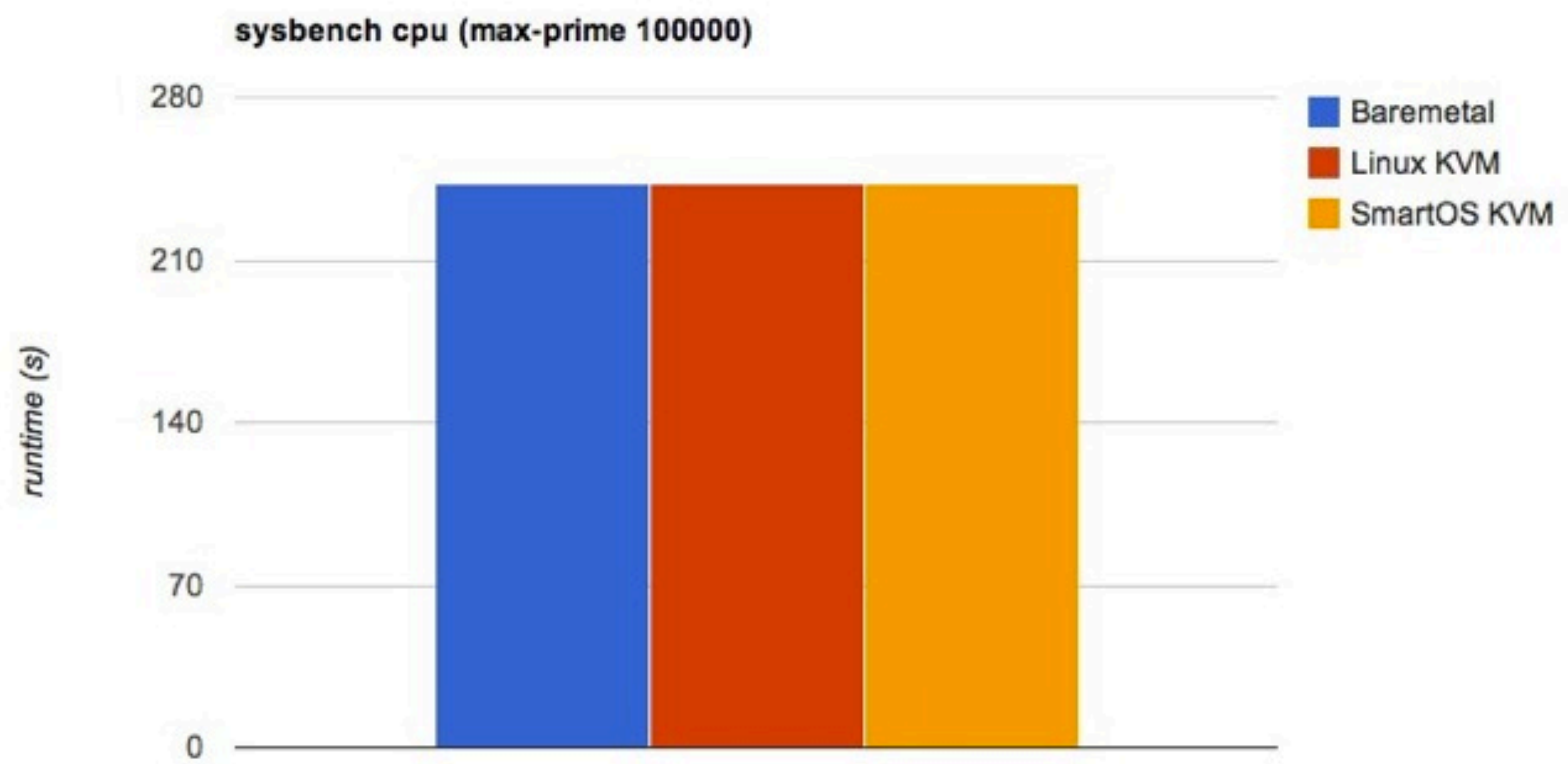


Notable bugs

- In the course of this port, we did not discover any bug that one would call a bug in KVM — it's *very* solid!
- Our bugs were (essentially) all self-inflicted, e.g.:
  - We erroneously configured QEMU such that both QEMU *and* KVM thought they were responsible for the 8254/8259!
  - We use a per-CPU GSBASE where Linux does not — Linux KVM doesn't have any reason to reload the host's GSBASE on CPU migration, but not doing so induces host GSBASE corruption: two physical CPUs have the same CPU pointer (one believes it's the other), resulting in <u>total mayhem</u>
  - We reimplemented the FPU save code in terms of our native equivalent — and introduced a nasty corruption bug in the process by plowing TS in CR0!

Port performance

- Not surprisingly, our port performs at baremetal speeds for entirely CPU-bound workloads:

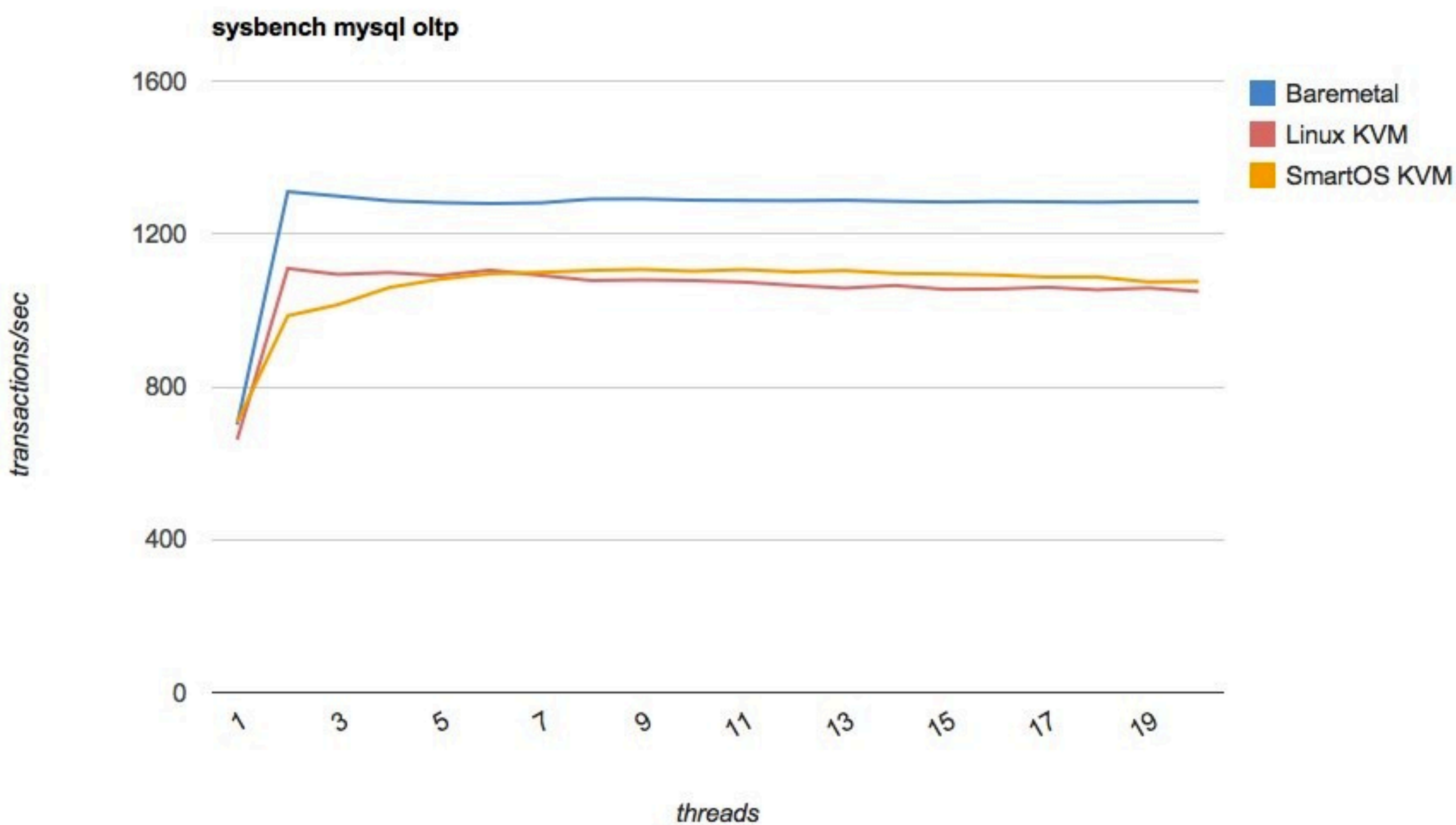


- But it took us a surprising amount of time to get to this result: due to dynamic overclocking, SmartOS KVM was initially operating 5% faster than baremetal!

Port performance

Joyent

- Our port of KVM seems to at least be in the hunt on other workloads, e.g.:



Port status

- Port is publicly available:
  - Github repo for KVM itself:
    `https://github.com/joyent/illumos-kvm`
  - Github repo for our branch of QEMU 0.14.1:
    `https://github.com/joyent/illumos-kvm-cmd`
  - `illumos-kvm-cmd` repo contains minor QEMU 0.14.1 patches to support our port, all of which we intend to upstream
- Within its scope, this port is at or near production quality
- Worthwhile to discuss the limitations of our port, the divergences of our port from Linux KVM, and the enhancements to KVM that our port allows...

Limitation: guest memory is locked down

- As a cloud provider, we have something of an opinion on this: overselling memory is only for idle workloads
- In our experience, the dissatisfaction from QoS variability induced by memory oversell is not paid for by the marginal revenue of that oversell
- We currently lock down guest memory; failure to lock down memory will result in failure to start
- For those high multi-tenancy environments, we believe that hardware is the wrong level at which to virtualize...

Limitation: no memory deduplication

Joyent

- We don’t currently have an analog to the kernel same-page mapping (KSM) found in Linux
- This is technically possible, but we don’t see an acute need (for the same reason we lock down guest memory)
- We are interested to hear experiences with this:
  - What kind of memory savings does one see?
  - Is one kind of guest (Windows?) more likely to see savings?
  - What kind of performance overhead from page scanning?

Limitation: no nested virtualization

Joyent

- We don’t currently support nested virtualization — and we’re not sure that we’re ever going to implement it
- While for our own development purposes, we would like to see VMware Fusion support nested virtualization, we don’t see an acute need to support it ourselves
- Would be curious to hear about experiences with nested virtualization; is it being used in production, or is it primarily for development?

Divergence: User/kernel interface

- To minimize patches floated on QEMU, wanted to minimize any changes to the user/kernel interface
- ...but we have no `anon_inode_getfd()` analog
- This is required to implement the model of a 1-to-1 mapping between a file descriptor and a VCPU
- Added a new `KVM_CLONE` ioctl that makes the driver state in the operated-upon instance point to another
- To create a VCPU, QEMU (re)opens `/dev/kvm`, and calls `KVM_CLONE` on the new instance, specifying the extant instance

Divergence: Context ops

- illumos has the ability to install *context ops* that are executed before and after a thread is scheduled on CPU
- Context ops were originally implemented to support CPU performance counter virtualization
- Context ops are installed with `installctx()`
- This facility proved essential — we use it to perform the equivalent of `kvm_sched_in()`/`kvm_sched_out()`

Divergence: Timers

- illumos has arbitrary resolution interval timer support via the *cyclic subsystem*
- Cyclics can be bound to a CPU or processor set and can be configured to fire at different interrupt levels
- While originally designed to be a high resolution interval timer facility (the system clock is implemented in terms of it), cyclics may also be used as a dynamically reprogrammable one-shots
- All KVM timers are implemented as cyclics
- We do not migrate cyclics when a VCPU migrates from one CPU to another, choosing instead to poke the target CPU from the cyclic handler

Enhancement: ZFS

- Strictly speaking, we have done nothing specifically for ZFS: running KVM on a ZFS volume (a *zvol*) Just Works
- But the presence of ZFS allows for KVM enhancements:
  - Constant time cloning allows for nearly instant provisioning of new KVM guests (assuming that the reference image is already present)
  - The ZFS's unified adaptive replacement cache (ARC) allows for guest I/O to be efficiently cached in the host — resulting in potentially massive improvements in random I/O (depending, of course, on locality)
  - We believe that ZFS remote replication can provide an efficient foundation for WAN-based cloning and migration

Enhancement: OS Virtualization

Joyent

- illumos has deep support for OS virtualization
- While our implementation does not require it, we run KVM guests in a *local zone*, with the QEMU process as the only process
- This was originally for reasons of accounting (we use the zone as the basis for QoS, resource management, I/O throttling, billing, instrumentation, etc.)…
- ...but given the recent KVM vulnerabilities, it has become a matter of security
- OS virtualization neatly containerizes QEMU and drastically reduces attack surface for QEMU exploits

Enhancement: Network virtualization

- illumos has deep support for network virtualization
- We create a *virtual NIC* (VNIC) per KVM guest
- We wrote simple glue to connect this to virtio — and have been able to push 1 Gb line to/from a KVM guest
- VNICs give us several important enhancements, all with minimal management overhead:
    - *Anti-spoofing* confines guests to a specified IP (or IPs)
    - *Flow management* allows guests to be capped at specified levels of bandwidth — essential in overcommitted networks
    - *Resource management* allows for observability into per-VNIC (and thus, per-guest) throughput from the host

Enhancement: Kernel statistics

- illumos has the *kstat facility* for kernel statistics
- We reimplemented `kvm_vcpu_stat` as a kstat
- We added a kvmstat tool to illumos that consumes these kstats, displaying them per-second and per-VCPU
- For example, one second of kvmstat output with two VMs running — one idle 2 VCPU Linux guest, with one booting 4 VCPU SmartOS guest:

```
 pid vcpu |  exits :  haltx   irqx  irqwx    iox  mmiox |   irqs   emul   eptv
4668    0 |     23 :      6      0      0      1      0 |      6     16      0
4668    1 |     25 :      6      1      0      1      0 |      6     16      0
5026    0 |  17833 :    223   2946    707    106      0 |   3379  13315      0
5026    1 |  18687 :    244   2761    512      0      0 |   3085  14803      0
5026    2 |  15696 :    194   3452    542      0      0 |   3568  11230      0
5026    3 |  16822 :    244   2817    487      0      0 |   3100  12963      0
```

Enhancement: DTrace

- As of QEMU 0.14, QEMU has DTrace probes — we lit those up on illumos
- Added a bevy of SDT probes to KVM itself, including all of the call-sites of the `trace_*()` routines
- Added `vmregs[]` variable that queries current VMCS, allowing for guest behavior to be examined
- Can all be enabled dynamically and safely, and aggregated on an arbitrary basis (e.g., per-VCPU, per-VM, per-CPU, etc.)
- Pairs well with kvmstat to understand workload characteristics in production deployments

Enhancement: DTrace, cont.

Joyent

- Example D script:

```
kvm-guest-exit
{
        @[pid, tid, strexitno[vmregs[VMX_VM_EXIT_REASON]] = count();
}

tick-1sec
{
        printf("%10s %10s %-50s %s\n",
            "PID", "TID", "REASON", "COUNT");
        printa("%10d %10d %-50s %@d\n", @);
        printf("\n");
        clear(@);
}
```

- e.g., output from `fork()/exit()`-heavy workload:

```
       PID        TID REASON                                             COUNT
      3949          3 EXIT_REASON_CR_ACCESS                              0
      3949          3 EXIT_REASON_HLT                                    0
      3949          3 EXIT_REASON_IO_INSTRUCTION                         2
      3949          3 EXIT_REASON_EXCEPTION_NMI                          11
      3949          3 EXIT_REASON_EXTERNAL_INTERRUPT                     14
      3949          3 EXIT_REASON_APIC_ACCESS                            202
      3949          3 EXIT_REASON_CPUID                                  8440
```

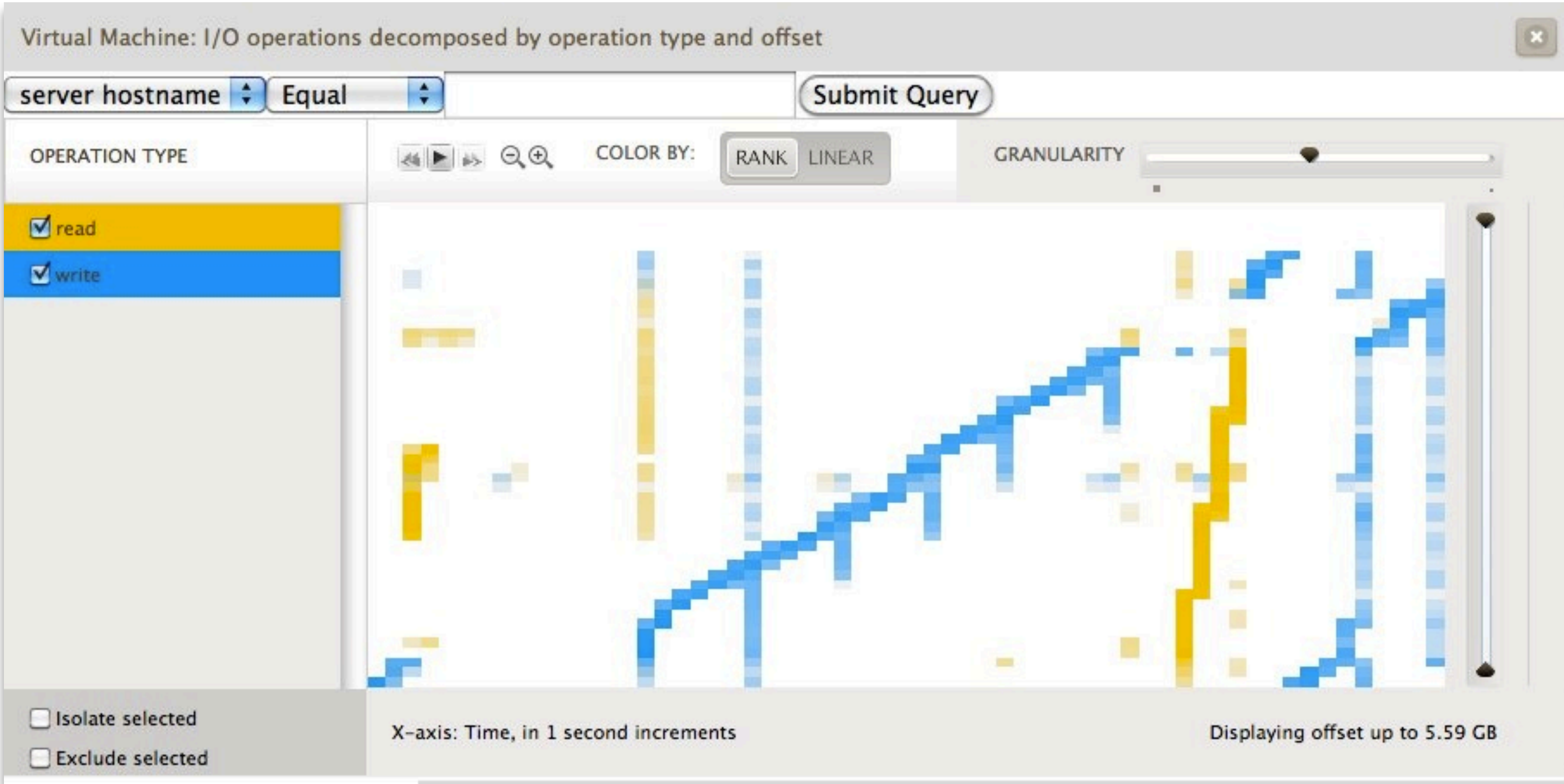
WTF?!

Enhancement: DTrace, cont.

- Orthogonal to this work, we have developed a real-time analytics framework that instruments the cloud using DTrace and visualizes the result
- We have extended this facility to the new DTrace probes in our KVM port
- We have only been experimenting with this very recently, but the results have been fascinating!
- For example...

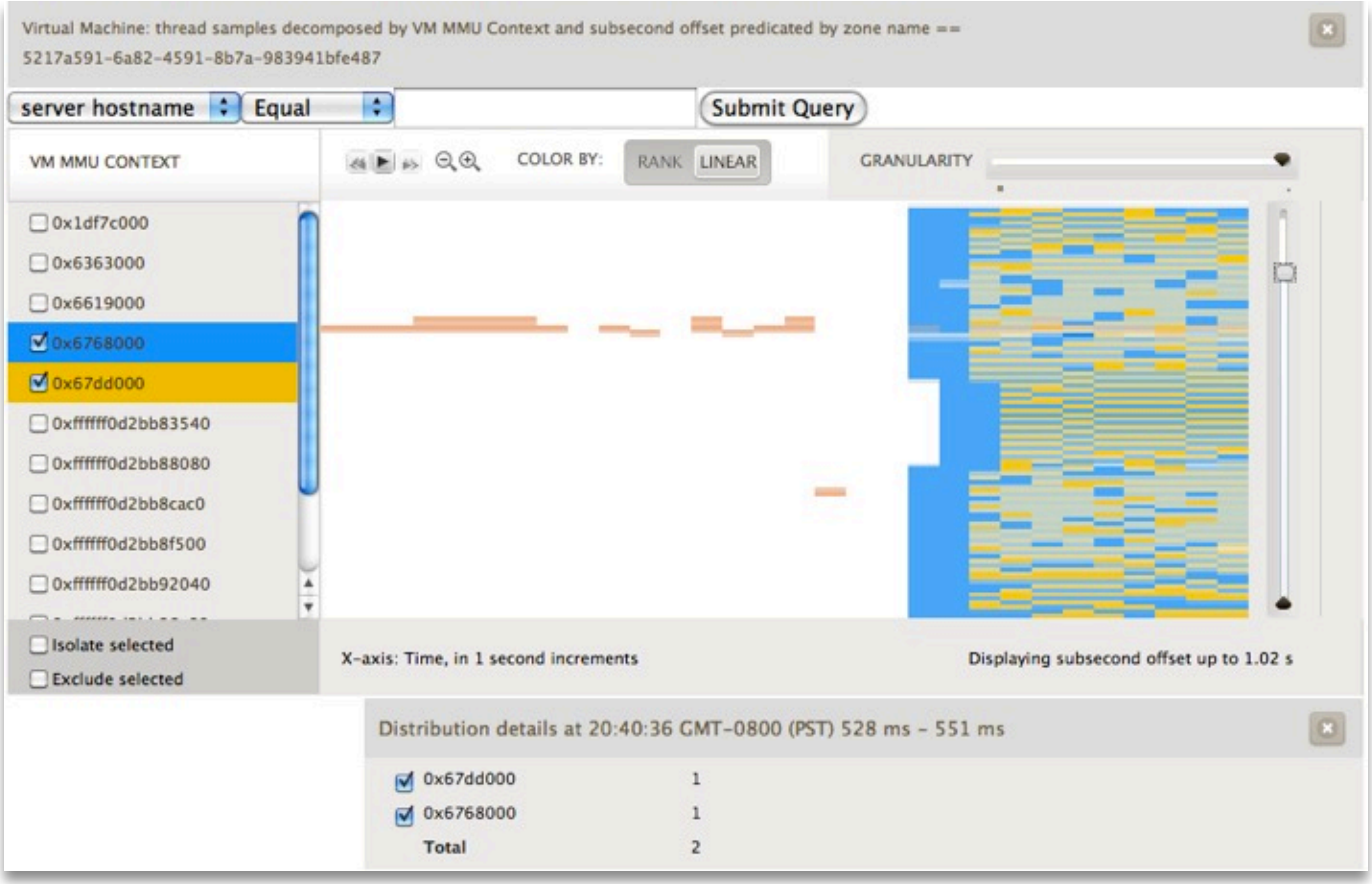
Enhancement: Visualizing DTrace on KVM

- Observing ext3 write offsets in a logical volume on a workload that creates and removes a 3 GB file:



Enhancement: Visualizing DTrace on KVM

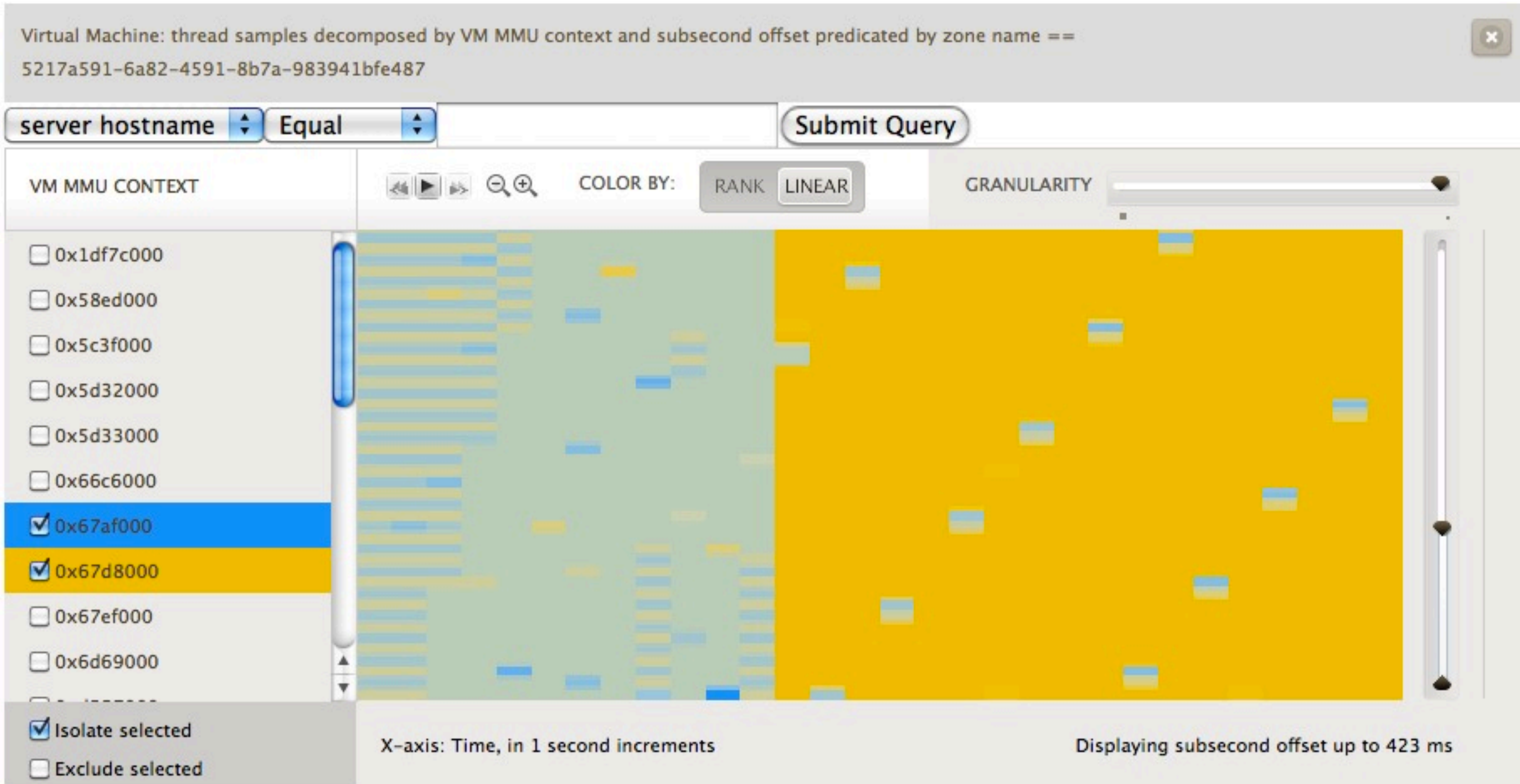
- Decomposing by guest CR3 and millisecond offset within-the-second, sampled at 99 hertz with two compute-bound processes:



Enhancement: Visualizing DTrace on KVM

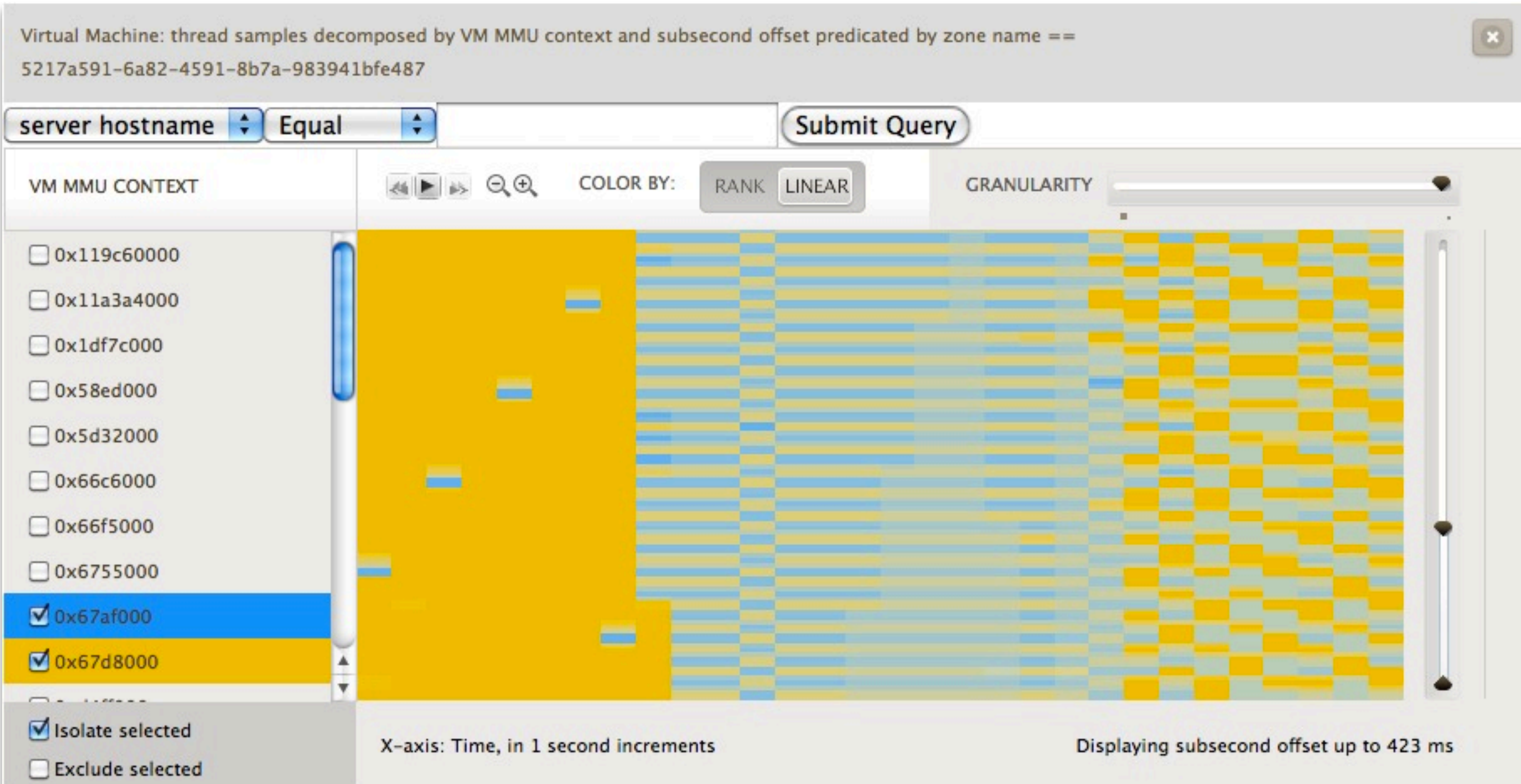
Joyent

- Same view, but now sampled at 999 hertz — and with one of the compute-bound processes reniced:



Enhancement: Visualizing DTrace on KVM

- Same view, same sample frequency — but horsing around with nice values:



Enhancement: Visualizing DTrace on KVM

- Interrupt requests decomposed by IRQ vector and offset within-the-second:

Engaging the community

- We are very excited to engage the KVM community; potential areas of collaboration:
  - Working on KVM performance. With DTrace, we have much better visibility into guest behavior; it seems possible (if not likely!) that resulting improvements to KVM will carry from one host system to the other
  - Collaborating on testing. We would love to participate in automated KVM testing infrastructure; we dream of a farm of oddball ISOs and the infrastructure to boot and execute them!
  - Collaborating on benchmarking. We have not examined SPECvirt_sc2010 in detail, but would like to work with the community to develop standard benchmarks

Thank you!

Joyent

- Josh Wilsdon and Rob Gulewich of Joyent for their instrumental assistance in this effort
- Brendan Gregg of Joyent for examining the performance of KVM — and for his tenacity in discovering the effects of dynamic overclocking!
- Fabrice Bellard for lighting the path with QEMU
- Intel for a rippin’ fast CPU (+ EPT!) in Nehalem
- Avi Kivity and team for putting it all together with KVM!
- The illumos community for their enthusiastic support