Google

Vcpu priority boosting for latency sensitive workloads

Vineeth Pillai (Google) Joel Fernandes (Google)



Agenda

- The problem
- The root cause
- The solution
- Some performance numbers
- Future work

The Problem

- Guest virtual machine experiences considerable latencies when the host is under load (sometimes 100s of ms of latency).
 - Overcommitted physical cpus(multiple VMs, host processes, ...)
- Android apps runs in a virtual machine in chromeOS and on low-end devices under considerable load, we observe
 - Audio glitches
 - Stuttering and unsmooth Video playback

The root cause

- Double scheduling: Host schedules vcpu threads and the guest schedules the tasks running inside the guest
- But both schedulers are unaware of the other
 - Hosts schedules vcpu threads without knowing what's being run on the vcpu
 - Guest schedules tasks without knowing where the vcpu is running physically
- vCPUs are regular CFS tasks in the host and does not get to run in a timely fashion when the host is experiencing load
- Host scheduler tries to be fair and doesn't know about the priority requirements
- This can cause issues with latencies, power consumption, resource utilization etc.

An example

Traces collected (guest and host) when video lag was happening (android's VLC player)

nRdy73	~	Ŧ	5 K													
crosvm_vcpu0 5305		Ŧ	Runn	Running			Rut	nnable (Pr	eempted)		Runnable (Preem	pted)	Runni			
crosvm_vcpu1 5306		Ŧ	reempted)	R Running	Running				Runnabl	le (Preempted)			100	Running		
Cpu O			C C10_ C V.b_	c crosvm.	Crosvm_	2	chrome 27_ ThreadPool_	chro_ Chro_		chrome [27616] ThreadPoolForeg [276	22]		sh_	crosvm [5027] crosvm_vcpu1_	chrome 27616 ThreadPoolForeg 276	22]
Cpu 1			CCTO CCTO	Crosvm	crosvm [ch_ CL	c chrome c ThreadPo	27616]	chrome [27616] chrome [27616]	chrome 27616 ThreadPoolForeg	chrome [27616] chrome [27616]	chr	C705	ch. chrome. Ct. Thread.	chrome 1256 CrBrowserMain (1256)	chr Cit
Сри 100			syst_ bind_	S				0	kg videolan.vic [1010 NDK MediaCodec_[1011	0979] 281]						
Сри 101			com.spoti	6					system_server [1000	355] 856]				SY		



A (possible) Solution for minimizing latencies

- CFS is not strictly priority aware, so let's do RT!
- Boosting priority of vcpus to RT helps
- But vcpus running as RT through the life time of a VM has negative impact to the whole system
- The issue is manifested vcpus may hijack the host!
 - We have the same problem now: vcpus running a normal workload may preempt a latency sensitive workload in the host.

Another (possible) solution

- Let the host and guest talk ;-)
- Share scheduling information between host and guest
- A cooperative scheduling framework by sharing the information between host and guest so that both could make educated scheduling decision





Dynamic vcpu priority management

- One specific use case for cooperative guest/host scheduling
- Aimed at reducing latencies for latency sensitive tasks in the guest.
- Guest shares the priority requirements and host can boost/unboost as needed



Dynamic vcpu priority management

- Host proactively boosts the vcpu thread when it knows that the guest will be running a latency sensitive work load, example: interrupt injection
- Guest shares its need for a priority boost when running latency sensitive workloads
 - Host boosts the vcpu thread and shares the boost status with guest
- Guest can request an unboost when it no longer runs latency sensitive workloads
- Host implements throttling and forceful unboosting for buggy/rogue guest kernels



Latency sensitive guest contexts

- NMIs, interrupts, softirqs
- Tasks with priority higher than SCHED_OTHER
- Preemption disabled in guest



Priority Inversion - Need to boost spinlocks too!

- May happen if not all Vcpus are boosted
- CFS VCPU preempted after taking a spinlock
- RT VCPU blocks on the spin lock acquired by the above CFS VCPU
- Even worse if the the RT VCPU preempts the CFS VCPU holding the lock on the same physical CPU.

Implementation (Proof of Concept)

- x86 (Intel and AMD) only but easily portable to other architectures
- Guest uses MSR to communicate the GPA of shared memory location
 - This also acts as guest intent to use the functionality
- Boosts the vcpu to RT
 - SCHED_RR and priority 8 by default and is tunable
- Unboost to SCHED_OTHER with nice 0
 - TODO: Could be made tunable. POC it is fixed.

Implementation (Proof of Concept)

Enabling the feature:

- Host uses CPUID to advertise the feature to guest
 - VMM can enable/disable the advertisement (PoC always advertises)
- Knobs to enable/disable this feature globally and per-vm.

Implementation Details: Synchronous boost/unboost

- Uses hypercall mechanism.
- Guest shall request a boost via hypercall if needed:
 - Currently not used. Only lazy (async) boosting is used in the POC.
- Guest requests an unboost via hypercall once it completes the latency sensitive workload:
 - Switching to SCHED_OTHER (normal tasks) in scheduler
 - Return to user mode SCHED_OTHER after servicing interrupt, softirq, preempt enable etc.
 - IRQ disabling/enabling doesn't do boosting currently.



What's hypercall overhead like?

- Overhead of Hypercall seen to be 10-20 micro seconds, or so.
 - Caveat: Unless VM is nested virt.
- Seems very reasonable, as the latencies this fixes are 3 orders of magnitude higher (10s or 100s of ms) !



Implementation Details - Asynchronous boosting

- Guest shares its intent (boost, preemption state) via shared memory and continues execution as long as it can.
- On the very next VMEXIT, host takes action.
- Avoids an extra VMEXIT caused by hypercall.



Implementation Details: Proactive boosting

- Host boosts the vcpu threads on situations where it has clear information that guest is running a latency sensitive workload
 - Interrupt Injection
 - Before injecting an interrupt into the guest, host boosts the priority of the vcpu
 - Guest VCPU halt/blocking
 - When the guest vcpu halts, the wakeup would be due to an event which needs to be handled immediately(interrupts, IPIs etc).
 - Priority is boosted as the guest vcpu task is switched out.

Care to be taken during interrupt handling

Scenario: Emulated device interrupt and VFIO pass thru devices. Host needs to inject an interrupt in guest on VCPU0

Google

Guest

Host

18

In the Interrupt exit path, check NEED RESCHED and deboost if not set. or Service Interrupt (NMI, IRQ, **Resume VM** exiting to CFS. soft IRQ) Timeline schedule() In VMM context (for emulated) or VFIO interrupt handler (for pass thru) Deboost and update In vCPU thread 1. Inject interrupt into guest (VCPU0) shared page context: schedule VCPU0 Boost VCPU0 to RT а. (vcpu run) Update shared page b.

c. Wakeup process.

Care to be taken during POSTED interrupt handling

Scenario: Guest is ON CPU. Posted interrupt is received directly in the guest on VCPU0



9

Google

Implementation Details: Shared memory communication

Guest lets the host know about its need union guest_schedinfo { struct { for boosting/un boosting u8 boost_req; Guest lets the host know if preemption _u8 preempt_disabled; }; is disabled or not ____u64 pad; }; Host sees the above on the very next VMEXIT and takes action accordingly. /* Structure passed in via MSR_KVM_SCHED_REGION Host updates the shared memory with */ the boost status. Guest uses this to struct pv_sched_data { _u64 boost_status; avoid unnecessary memory updates and union guest_schedinfo schedinfo; }; hypercalls

Performance

- Synthetic Test (micro benchmarks)
 - Cyclictest (guest and host) when host is idle and busy
 - Busy host simulated by stress-ng
 - \circ Intervals 500 us and 1000 us.
- Simulating real world workloads
 - Oboetester glitch test.

Synthetic tests - Idle Host (cyclictest - SCHED_RR)

VM: Average latency in micro seconds

Interval(us)	vanilla	vcpu_boost	static_rt
500	9	9	10
1000	34	35	35

VM: Max latency in micro seconds

Interval(us)	vanilla	vcpu_boost	static_rt
500	1577	1433	140
1000	6649	765	204

Legend:

Vanilla: Host kernel: 6.1 guest kernel: 5.10 Vcpu_boost: Host and guest kernels with the patches Static_rt: vanilla 6.1, with vcpu threads boosted to RT

Host: Average latency in micro seconds

Interval(us)	vanilla	vcpu_boost	static_rt
500	5	3	3
1000	3	3	3

Host: Max latency in micro seconds

Interval(us)	vanilla	vcpu_boost	static_rt		
500	1577	1526	15969		
1000	697	174	2444		

Synthetic tests - Busy Host (cyclictest - SCHED_RR)

VM: Average latency in micro seconds

Interval(us)	vanilla	vcpu_boost	static_rt
500	887	21	25
1000	6335	45	38

VM: Max latency in micro seconds

Interval(us)	vanilla	vcpu_boost	static_rt
500	216835	13978	1728
1000	199575	70651	1537

Legend:

Vanilla: Host kernel: 6.1 guest kernel: 5.10 Vcpu_boost: Host and guest kernels with the patches Static_rt: vanilla 6.1, with vcpu threads boosted to RT

Host: Average latency in micro seconds

Interval(us)	vanilla	vcpu_boost	static_rt
500	6	6	7
1000	11	11	14

Host: Max latency in micro seconds

Interval(us)	vanilla	vcpu_boost	static_rt
500	2075	2114	2447
1000	1886	1285	27104

Simulating real world workloads: Audio glitches

Oboetester app in android vm on chromeos used to test aud Test Gli

Host kernel : 6.1

Guest Kernel: 5.10

Buffer size	Nol	oad	Speedometer		
	Vanilla 6.1	vcpu_boost	Vanilla 6.1	vcpu_boost	
96 (2ms)	20	4	1365	67	
256 (5ms)	3	1	524	23	
512 (10ms)	0	0	25	24	

÷	🊺 Phone 🔻	_ 6	×			
Test Glitches						
Show Settings - INPP burst = 512, capacity timestamp.latency = time between callba written 0x0001A000 Started, #cb=214, f/ buffer size = 7680 = Input Margin: 0 0	UT / = 8192, devID = 9, 27.0/31.4/35.4 ms cks = 10.3/10.7/11 - read 0x00019C00 cb=512, 1% cpu (15 * 512) + 0, xRu 0 1 0 2 0 3	MMAP, SH .0 ms 0 = 1024 frames n# = 0 4				
Show Settings - OUT	PUT					
Show Settings - 001 P01 burst = 512, capacity = 8192, devID = 2, MMAP, SH timestamp.latency = 56.4/62.1/66.8 ms time between callbacks = 10.3/10.7/11.0 ms written 0x0001Ac00 - read 0x0001A800 = 1024 frames Started, #cb=215, f/cb=512, 1% cpu buffer size = 1024 = (2*512) + 0, xRun# = 0 buffer Size = 1024 = (2*512) + 0, xRun# = 0 buffer Size = 2, 1024 / 1024 / 8192 1 1 2 2 3 Tolerance = 0.100						
START	STOP	SHARE				
STARTSTOPSHAREstate = WAITING_FOR_LOCK unlocked.frames = 46487 locked.frames = 532 glitch.frames = 22228 reset.count = 1 peak.amplitude = 0.065038 sinea.noise.ratio.db = 0.00 time.total = 2.34 seconds time.no.glitches = 0.00 glitch.count = 114						

Google



Video lag example with the fixes applied

Traces collected (guest and host) during video playback (android's VLC player)

		00:00:00	I 00:00-05	00:00:10 00:00:15	00:00:20	00:00:25	30 I 00:00:35	00:00:40
00:05:58 + 570 657 906			00:00:05 928 200 000	00-00-05 928 400 000	00:00:05 928 600 000	00:00:05 928 800 000	00:00:05 929 000 000	00-00-05 929 200 000
nRdy55	~ Ŧ	5 K						
Сри 0			chrome [6017] ThreadPoolForeg (6022]	crosvm [4979] chrome crosvm_vcpu1 (5 Chrome	chrome [1382] CrBrowserMain [1382]		chrome [1453] hrome_ChildIOT [1514]	crosvm [4979] crosvm_vcpu1 [5245]
Сри 1			chrome [6017] ThreadPoolForeg (6020)	chrome [1382] Chrome_IOThrea_		crosvm [4979] crosvm_vcpu0 [5244]		chrome [14 Dem Thread [1
Сри 100			android.hardware.media.c2@1.0-sen c2@1.0-senice-[10001	vice-v4l2 [1000198] surfacefil I8] TimerDisp	ins_sur atV_af	surfaceflinger surfaceflinger	[1000143] [1000143]	
Сри 101		and	Iroid hardware media.c2@1.0-service-v4l DecodeComponent [1002849]	2 [1000198] kw_ kw_	android.har	dware.media.c2@1.0-service-v4l2 [100019 DecodeComponent (1002849]	8]	android HwBinder
Android logs								



Other use cases for cooperative scheduling

- Host can share the physical cpu load and pressure to guest and also the vcpu physical placement.
 - Guest can effectively decide the most effective vcpus for the tasks it needs to run
 - Guest can also request an effective vcpu placement if it has the above information. Host can accept or deny the request based on its requirements.
- Guest can share the vcpu load with the host and may include task level information(priority, load etc)
 - Host can make an educated decision on where to place the guest vcpus appropriately
 - Host can also suggest task placement hints to the guest.

Future Work

- Generic implementation which would make it easy for porting to other architectures
- Use paravirt ops
- Optimizations in the critical path for speed and cache efficiency.
- Standardize the memory sharing framework so as to extend it for other use cases.
- Priority management for irq disable/enable code paths.
- Optimizing proactive boosting during interrupt injection.

