Patchwork x86/kvm: fix condition to update kvm master clocks

Submitter	Roman Kagan
Date	May 26, 2016, 2:49 p.m.
Message ID	<1464274195-31296-1-git-send-email-rkagan@virtuozzo.com>
Download	mbox | patch
Permalink	/patch/9137051/
State	New
Headers	show

Comments

Roman Kagan - May 26, 2016, 2:49 p.m.

```
The condition to schedule per-VM master clock updates is inversed; as a
result the master clocks are never updated and the kvm_clock drift WRT
host's NTP-disciplined clock (which was the motivation to introduce this
machinery in the first place) still remains.

Fix it, and reword the comment to make it more apparent what the desired
behavior is.

Cc: Owen Hofmann <osh@google.com>
Cc: Paolo Bonzini <pbonzini@redhat.com>
Cc: Marcelo Tosatti <mtosatti@redhat.com>
Signed-off-by: Roman Kagan <rkagan@virtuozzo.com>
---
 arch/x86/kvm/x86.c | 6 +++---
 1 file changed, 3 insertions(+), 3 deletions(-)
```

Radim Kr?má? - May 26, 2016, 8:19 p.m.

```
2016-05-26 17:49+0300, Roman Kagan:
> The condition to schedule per-VM master clock updates is inversed; as a
> result the master clocks are never updated and the kvm_clock drift WRT
> host's NTP-disciplined clock (which was the motivation to introduce this
> machinery in the first place) still remains.
>
> Fix it, and reword the comment to make it more apparent what the desired
> behavior is.
>
> Cc: Owen Hofmann <osh@google.com>
> Cc: Paolo Bonzini <pbonzini@redhat.com>
> Cc: Marcelo Tosatti <mtosatti@redhat.com>
> Signed-off-by: Roman Kagan <rkagan@virtuozzo.com>
> ---
>  arch/x86/kvm/x86.c | 6 +++---
>  1 file changed, 3 insertions(+), 3 deletions(-)
>
> diff --git a/arch/x86/kvm/x86.c b/arch/x86/kvm/x86.c
> index c805cf4..d8f591c 100644
> --- a/arch/x86/kvm/x86.c
> +++ b/arch/x86/kvm/x86.c
> @@ -5810,10 +5810,10 @@ static int pvclock_gtod_notify(struct notifier_block *nb, unsigned long unused,
>
>       update_pvclock_gtod(tk);
>
> -     /* disable master clock if host does not trust, or does not
> -      * use, TSC clocksource
> +     /* only schedule per-VM master clock updates if the host uses TSC and
> +      * there's at least one VM in need of an update
>        */
```

```
> -     if (gtod->clock.vclock_mode != VCLOCK_TSC &&
> +     if (gtod->clock.vclock_mode == VCLOCK_TSC &&

I think we still want to disable master clock when vclock_mode is not
VCLOCK_TSC.

>           atomic_read(&kvm_guest_has_master_clock) != 0)

And I don't see why we don't want to enable master clock if the host
switches back to TSC.

>               queue_work(system_long_wq, &pvclock_gtod_work);

Queueing unconditionally seems to be the correct thing to do.

Interaction between kvm_gen_update_masterclock(), pvclock_gtod_work(),
and NTP could be a problem:  kvm_gen_update_masterclock() only has to
run once per VM, but pvclock_gtod_work() calls it on every VCPU, so
frequent NTP updates on bigger guests could kill performance.  (Maybe
kvm_gen_update_masterclock() is too wasteful even when called once.)
--
To unsubscribe from this list: send the line "unsubscribe kvm" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at  http://vger.kernel.org/majordomo-info.html
```

Roman Kagan - May 27, 2016, 5:28 p.m.

```
On Thu, May 26, 2016 at 10:19:36PM +0200, Radim Krčmář wrote:
> 2016-05-26 17:49+0300, Roman Kagan:
> > The condition to schedule per-VM master clock updates is inversed; as a
> > result the master clocks are never updated and the kvm_clock drift WRT
> > host's NTP-disciplined clock (which was the motivation to introduce this
> > machinery in the first place) still remains.
> >
> > Fix it, and reword the comment to make it more apparent what the desired
> > behavior is.
> >
> > Cc: Owen Hofmann <osh@google.com>
> > Cc: Paolo Bonzini <pbonzini@redhat.com>
> > Cc: Marcelo Tosatti <mtosatti@redhat.com>
> > Signed-off-by: Roman Kagan <rkagan@virtuozzo.com>
> > ---
> >  arch/x86/kvm/x86.c | 6 +++---
> >  1 file changed, 3 insertions(+), 3 deletions(-)
> >
> > diff --git a/arch/x86/kvm/x86.c b/arch/x86/kvm/x86.c
> > index c805cf4..d8f591c 100644
> > --- a/arch/x86/kvm/x86.c
> > +++ b/arch/x86/kvm/x86.c
> > @@ -5810,10 +5810,10 @@ static int pvclock_gtod_notify(struct notifier_block *nb, unsigned long unused,
> >
> >     update_pvclock_gtod(tk);
> >
> > -   /* disable master clock if host does not trust, or does not
> > -    * use, TSC clocksource
> > +   /* only schedule per-VM master clock updates if the host uses TSC and
> > +    * there's at least one VM in need of an update
> >      */
> > -   if (gtod->clock.vclock_mode != VCLOCK_TSC &&
> > +   if (gtod->clock.vclock_mode == VCLOCK_TSC &&
>
> I think we still want to disable master clock when vclock_mode is not
> VCLOCK_TSC.

Hmm, indeed.  I missed that this thing is used both to update the
contents and to enable/disable it.  I'd say it's confusing...

>
> >           atomic_read(&kvm_guest_has_master_clock) != 0)
>
> And I don't see why we don't want to enable master clock if the host
> switches back to TSC.
```

Agreed (even though I guess it's not very likely: AFAICS once switched
to a different clocksource, the host can switch back to TSC only upon
human manipulating /sys/devices/system/clocksource).

> >   queue_work(system_long_wq, &pvclock_gtod_work);
> >
> Queueing unconditionally seems to be the correct thing to do.

The notifier is registered at kvm module init, so the work will be
scheduled even when there are no VMs at all.

> Interaction between kvm_gen_update_masterclock(), pvclock_gtod_work(),
> and NTP could be a problem: kvm_gen_update_masterclock() only has to
> run once per VM, but pvclock_gtod_work() calls it on every VCPU, so
> frequent NTP updates on bigger guests could kill performance.

Unfortunately, things are worse than that: this stuff is updated on
every *tick* on the timekeeping CPU, so, as long as you keep at least
one of your CPUs busy, the update rate can reach HZ. The frequency of
NTP updates is unimportant; it happens without NTP updates at all.

So I tend to agree that we're perhaps better off not fixing this bug and
leaving the kvm_clocks to drift until we figure out how to do it with
acceptable overhead.

Roman.
--
To unsubscribe from this list: send the line "unsubscribe kvm" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at http://vger.kernel.org/majordomo-info.html

Marcelo Tosatti - May 29, 2016, 11:38 p.m.

On Thu, May 26, 2016 at 05:49:55PM +0300, Roman Kagan wrote:
> The condition to schedule per-VM master clock updates is inversed; as a
> result the master clocks are never updated and the kvm_clock drift WRT
> host's NTP-disciplined clock (which was the motivation to introduce this
> machinery in the first place) still remains.
>
> Fix it, and reword the comment to make it more apparent what the desired
> behavior is.
>
> Cc: Owen Hofmann <osh@google.com>
> Cc: Paolo Bonzini <pbonzini@redhat.com>
> Cc: Marcelo Tosatti <mtosatti@redhat.com>
> Signed-off-by: Roman Kagan <rkagan@virtuozzo.com>
> ---
> arch/x86/kvm/x86.c | 6 +++---
> 1 file changed, 3 insertions(+), 3 deletions(-)
>
> diff --git a/arch/x86/kvm/x86.c b/arch/x86/kvm/x86.c
> index c805cf4..d8f591c 100644
> --- a/arch/x86/kvm/x86.c
> +++ b/arch/x86/kvm/x86.c
> @@ -5810,10 +5810,10 @@ static int pvclock_gtod_notify(struct notifier_block *nb, unsigned long unused,
>
>   update_pvclock_gtod(tk);
>
> -   /* disable master clock if host does not trust, or does not
> -   * use, TSC clocksource
> +   /* only schedule per-VM master clock updates if the host uses TSC and
> +   * there's at least one VM in need of an update
>   */
> -   if (gtod->clock.vclock_mode != VCLOCK_TSC &&
> +   if (gtod->clock.vclock_mode == VCLOCK_TSC &&
>   atomic_read(&kvm_guest_has_master_clock) != 0)
>   queue_work(system_long_wq, &pvclock_gtod_work);
>
> --
> 2.5.5

```
NAK, as stated this leaves clocks out of sync between execution of
pvclock_gtod_notify and pvclock_gtod_work execution.
--
To unsubscribe from this list: send the line "unsubscribe kvm" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at  http://vger.kernel.org/majordomo-info.html
```

Marcelo Tosatti - June 9, 2016, 3:27 a.m.

```
On Sun, May 29, 2016 at 08:38:44PM -0300, Marcelo Tosatti wrote:
> On Thu, May 26, 2016 at 05:49:55PM +0300, Roman Kagan wrote:
> > The condition to schedule per-VM master clock updates is inversed; as a
> > result the master clocks are never updated and the kvm_clock drift WRT
> > host's NTP-disciplined clock (which was the motivation to introduce this
> > machinery in the first place) still remains.
> >
> > Fix it, and reword the comment to make it more apparent what the desired
> > behavior is.
> >
> > Cc: Owen Hofmann <osh@google.com>
> > Cc: Paolo Bonzini <pbonzini@redhat.com>
> > Cc: Marcelo Tosatti <mtosatti@redhat.com>
> > Signed-off-by: Roman Kagan <rkagan@virtuozzo.com>
> > ---
> >  arch/x86/kvm/x86.c | 6 +++---
> >  1 file changed, 3 insertions(+), 3 deletions(-)
> >
> > diff --git a/arch/x86/kvm/x86.c b/arch/x86/kvm/x86.c
> > index c805cf4..d8f591c 100644
> > --- a/arch/x86/kvm/x86.c
> > +++ b/arch/x86/kvm/x86.c
> > @@ -5810,10 +5810,10 @@ static int pvclock_gtod_notify(struct notifier_block *nb, unsigned long unused,
> >
> >     update_pvclock_gtod(tk);
> >
> > -   /* disable master clock if host does not trust, or does not
> > -    * use, TSC clocksource
> > +   /* only schedule per-VM master clock updates if the host uses TSC and
> > +    * there's at least one VM in need of an update
> >      */
> > -   if (gtod->clock.vclock_mode != VCLOCK_TSC &&
> > +   if (gtod->clock.vclock_mode == VCLOCK_TSC &&
> >         atomic_read(&kvm_guest_has_master_clock) != 0)
> >             queue_work(system_long_wq, &pvclock_gtod_work);
> >
> > --
> > 2.5.5
>
> NAK, as stated this leaves clocks out of sync between execution of
> pvclock_gtod_notify and pvclock_gtod_work execution.

Ok, its not feasible to keep both REF_CLOCK (MSR) and shared memory
(REF_PAGE) clocks in sync. Even if the frequency correction is the same
for both, the kernel updates monotonic clock differently than the
stated frequency that is:

    monotonic clock (advertised via vsyscall notifier to use mult/freq pair) != tsc*freq

So the best solution IMO is to:

    reads of guest clock = max(shared memory clock, get_kernel_ns() +
                               kvmclock_offset)

Where reads of guest clock include: 1) kvm_get_time_and_clockread
(updates to kvmclock areas), 2) rdmsr(REF_CLOCK).

Unless someone has a better idea, Roman, can you update your patch to
include such solution? for kvm_get_time_and_clockread, can fast forward
kvmclock_offset so that

kvmclock_offset + get_kernel_ns() = shared memory clock
```

```
--
To unsubscribe from this list: send the line "unsubscribe kvm" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at  http://vger.kernel.org/majordomo-info.html
```

Marcelo Tosatti - June 9, 2016, 3:45 a.m.

```
On Thu, Jun 09, 2016 at 12:27:10AM -0300, Marcelo Tosatti wrote:
> On Sun, May 29, 2016 at 08:38:44PM -0300, Marcelo Tosatti wrote:
> > On Thu, May 26, 2016 at 05:49:55PM +0300, Roman Kagan wrote:
> > > The condition to schedule per-VM master clock updates is inversed; as a
> > > result the master clocks are never updated and the kvm_clock drift WRT
> > > host's NTP-disciplined clock (which was the motivation to introduce this
> > > machinery in the first place) still remains.
> > >
> > > Fix it, and reword the comment to make it more apparent what the desired
> > > behavior is.
> > >
> > > Cc: Owen Hofmann <osh@google.com>
> > > Cc: Paolo Bonzini <pbonzini@redhat.com>
> > > Cc: Marcelo Tosatti <mtosatti@redhat.com>
> > > Signed-off-by: Roman Kagan <rkagan@virtuozzo.com>
> > > ---
> > >  arch/x86/kvm/x86.c | 6 +++---
> > >  1 file changed, 3 insertions(+), 3 deletions(-)
> > >
> > > diff --git a/arch/x86/kvm/x86.c b/arch/x86/kvm/x86.c
> > > index c805cf4..d8f591c 100644
> > > --- a/arch/x86/kvm/x86.c
> > > +++ b/arch/x86/kvm/x86.c
> > > @@ -5810,10 +5810,10 @@ static int pvclock_gtod_notify(struct notifier_block *nb, unsigned long unused,
> > >
> > >   update_pvclock_gtod(tk);
> > >
> > > - /* disable master clock if host does not trust, or does not
> > > -  * use, TSC clocksource
> > > + /* only schedule per-VM master clock updates if the host uses TSC and
> > > +  * there's at least one VM in need of an update
> > >    */
> > > - if (gtod->clock.vclock_mode != VCLOCK_TSC &&
> > > + if (gtod->clock.vclock_mode == VCLOCK_TSC &&
> > >       atomic_read(&kvm_guest_has_master_clock) != 0)
> > >           queue_work(system_long_wq, &pvclock_gtod_work);
> > >
> > > --
> > > 2.5.5
> >
> > NAK, as stated this leaves clocks out of sync between execution of
> > pvclock_gtod_notify and pvclock_gtod_work execution.
>
> Ok, its not feasible to keep both REF_CLOCK (MSR) and shared memory
> (REF_PAGE) clocks in sync. Even if the frequency correction is the same
> for both, the kernel updates monotonic clock differently than the
> stated frequency that is:
>
>     monotonic clock (advertised via vsyscall notifier to use mult/freq pair) != tsc*freq
>
> So the best solution IMO is to:
>
>     reads of guest clock = max(shared memory clock, get_kernel_ns() +
>                                kvmclock_offset)
>
> Where reads of guest clock include: 1) kvm_get_time_and_clockread
> (updates to kvmclock areas), 2) rdmsr(REF_CLOCK).
>
> Unless someone has a better idea, Roman, can you update your patch to
> include such solution? for kvm_get_time_and_clockread, can fast forward
> kvmclock_offset so that
>
> kvmclock_offset + get_kernel_ns() = shared memory clock
```

```
Also please update the kvm-unit-test to check both ways (i did here and
what happens is that):

        Case1: with frequency correction from vsyscall:

            a = read_shared_clock;
            b = rdmsr(REF_CLOCK);

            fail: a > b

        Case2: without frequency correction from vsyscall:

            a = rdmsr(REF_CLOCK);
            b = read_shared_clock;

            fail: b < a

And the frequency correction advertised via syscall is the same as what
get_kernel_ns() is using. Conclusion: update_wall_time() updates to
monotonic clock do not match tsc*freqcorrection.

So there is nothing you can do to keep these clocks in sync
(other then checking what is the largest of them when reading
REF_CLOCK).



--
To unsubscribe from this list: send the line "unsubscribe kvm" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at  http://vger.kernel.org/majordomo-info.html
```

Roman Kagan - June 9, 2016, 12:09 p.m.

```
On Thu, Jun 09, 2016 at 12:27:10AM -0300, Marcelo Tosatti wrote:
> Ok, its not feasible to keep both REF_CLOCK (MSR) and shared memory
> (REF_PAGE) clocks in sync. Even if the frequency correction is the same
> for both, the kernel updates monotonic clock differently than the
> stated frequency that is:
>
>     monotonic clock (advertised via vsyscall notifier to use mult/freq pair) != tsc*freq
>
> So the best solution IMO is to:
>
>     reads of guest clock = max(shared memory clock, get_kernel_ns() +
>                                kvmclock_offset)
>
> Where reads of guest clock include: 1) kvm_get_time_and_clockread
> (updates to kvmclock areas), 2) rdmsr(REF_CLOCK).
>
> Unless someone has a better idea, Roman, can you update your patch to
> include such solution? for kvm_get_time_and_clockread, can fast forward
> kvmclock_offset so that
>
> kvmclock_offset + get_kernel_ns() = shared memory clock

I'm not sure I understand what you mean.

->system_time in pvclock *is* assigned kernel_ns + kvmclock_offset, i.e.
at the time kvm_get_time_and_clockread() runs they are in sync by
definition.  They can diverge later due to different whole number math
applied.

There's also a problem that there can be arbitrary amount of time
between assigning the return value for guest's rdmsr and actually
entering the guest to deliver that value.

In general I'm starting to feel the shared memory clock is trying to
provide stronger guarantees than really useful.  E.g. there's no such
thing as synchronous TSC between vCPUs in a virtual machine, so every
guest assuming it is broken; in reality that means that every sane guest
must tolerate certain violations of monotonicity when multiple CPUs are
```

used for timekeeping. I wonder if this consideration can allow for some
simplification of the paravirtual clock code...

Roman.
--
To unsubscribe from this list: send the line "unsubscribe kvm" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at http://vger.kernel.org/majordomo-info.html

Marcelo Tosatti - June 9, 2016, 6:25 p.m.

On Thu, Jun 09, 2016 at 03:09:46PM +0300, Roman Kagan wrote:
> On Thu, Jun 09, 2016 at 12:27:10AM -0300, Marcelo Tosatti wrote:
> > Ok, its not feasible to keep both REF_CLOCK (MSR) and shared memory
> > (REF_PAGE) clocks in sync. Even if the frequency correction is the same
> > for both, the kernel updates monotonic clock differently than the
> > stated frequency that is:
> >
> >   monotonic clock (advertised via vsyscall notifier to use mult/freq pair) != tsc*freq
> >
> > So the best solution IMO is to:
> >
> >   reads of guest clock = max(shared memory clock, get_kernel_ns() +
> >   kvmclock_offset)
> >
> > Where reads of guest clock include: 1) kvm_get_time_and_clockread
> > (updates to kvmclock areas), 2) rdmsr(REF_CLOCK).
> >
> > Unless someone has a better idea, Roman, can you update your patch to
> > include such solution? for kvm_get_time_and_clockread, can fast forward
> > kvmclock_offset so that
> >
> > kvmclock_offset + get_kernel_ns() = shared memory clock
>
> I'm not sure I understand what you mean.
>
> ->system_time in pvclock *is* assigned kernel_ns + kvmclock_offset, i.e.
> at the time kvm_get_time_and_clockread() runs they are in sync by
> definition. They can diverge later due to different whole number math
> applied.

Sync kvmclock_offset + get_kernel_ns() = shared memory clock (what you
read from the guest).

Add wrapper around get_kernel_ns + kvmclock_offset reads:

Record somewhere the last returned (last_returned_guestclock).

```
u64 read_guest_clock(struct kvm *kvm)
{
      mutex_lock(&guest_clock_mutex);
      ret = get_kernel_ns() + kvmclock_offset;
      kvm->arch.last_returned_guestclock = ret;
      mutex_unlock(&guest_clock_mutex);
}
```

Sync code (to be executed at masterclock updates and rdmsr(REF_CLOCK)):

1. read guest shared memory = smval.
2. read guest clock = get_kernel_ns() + kvmclock_offset = kclock
3. if (kclock < smval)
        kvmclock_offset += smval - kclock

Two possibilites for clocks state:

P1. shared memory clock > get_kernel_ns() + kvmclock_offset
P2. shared memory clock < get_kernel_ns() + kvmclock_offset

Two possibilites for guest behaviour:

```
G1. a = shared memory clock read;
    b = get_kernel_ns() + kvmclock_offset

G1/P1:

    a = shared memory clock read (previous read, stored in memory)
    b = get_kernel_ns() + kvmclock_offset

After sync code above: Note smval > a, so b = smval > a

G1/P2:

    a = shared memory clock read (previous read, stored in memory)
    b = get_kernel_ns() + kvmclock_offset

a < b, fine.

G2 (second possibility for guest behaviour)
    a = get_kernel_ns() + kvmclock_offset
    b = shared memory clock read

G2/P1: fine, b > a.

G2/P2:
    a = get_kernel_ns() + kvmclock_offset >
    b = shared memory clock read

So we have to either reduce get_kernel_ns() + kvmclock_offset so that
b is larger or 'stop get_kernel_ns() + kvmclock_offset'.

Can make get_kernel_ns() + kvmclock_offset be as small as
last_returned_guestclock (otherwise users of get_kernel_ns() +
kvmclock_offset can see time backwards).

    0. mutex_lock(&guest_clock_mutex);
    01. getkernelns = get_kernel_ns();
    1. read guest shared memory = smval.
    2. kclock = getkernelns + kvmclock_offset
    3. if (kclock > smval)
        kvmclock_offset -= min(kvmclock_offset - last_returned_guestclock,
                               kclock - smval)
    4. kclock = getkernelns + kvmclock_offset
    5. if (kclock > smval) {
        schedule_timeout(kclock - smval);
        kvmclock_offset -= min(kvmclock_offset - last_returned_guestclock,
                               kclock - smval)
       }
    6. mutex_unlock(&guest_clock_mutex);

That works, right?

I'll code that patch next week if you don't beat me to it.


> There's also a problem that there can be arbitrary amount of time
> between assigning the return value for guest's rdmsr and actually
> entering the guest to deliver that value.

Don't think that matters, see the 4 cases above.

> In general I'm starting to feel the shared memory clock is trying to
> provide stronger guarantees than really useful.  E.g. there's no such
> thing as synchronous TSC between vCPUs in a virtual machine, so every
> guest assuming it is broken;

There is, the TSC is monotonic between pCPUs:

    pCPU1          |       pCPU2

1.  a = read tsc
2.                      b = read tsc.
```

```
> in reality that means that every sane guest
> must tolerate certain violations of monotonicity when multiple CPUs are
> used for timekeeping.  I wonder if this consideration can allow for some
> simplification of the paravirtual clock code...

I think applications can fail.

--
To unsubscribe from this list: send the line "unsubscribe kvm" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at  http://vger.kernel.org/majordomo-info.html
```

Patch

```
diff --git a/arch/x86/kvm/x86.c b/arch/x86/kvm/x86.c
index c805cf4..d8f591c 100644
--- a/arch/x86/kvm/x86.c
+++ b/arch/x86/kvm/x86.c
@@ -5810,10 +5810,10 @@  static int pvclock_gtod_notify(struct notifier_block *nb, unsigned long unused,

        update_pvclock_gtod(tk);

-       /* disable master clock if host does not trust, or does not
-        * use, TSC clocksource
+       /* only schedule per-VM master clock updates if the host uses TSC and
+        * there's at least one VM in need of an update
         */
-       if (gtod->clock.vclock_mode != VCLOCK_TSC &&
+       if (gtod->clock.vclock_mode == VCLOCK_TSC &&
            atomic_read(&kvm_guest_has_master_clock) != 0)
                queue_work(system_long_wq, &pvclock_gtod_work);
```