Subject: Re: [PATCH 1/1, v7] cgroup/freezer: add per freezer duty ratio control
Posted by jacob.jun.pan on Tue, 15 Feb 2011 22:18:46 GMT
View Forum Message <> Reply to Message

On Mon, 14 Feb 2011 15:09:33 -0800
Matt Helsley <matthltc@us.ibm.com> wrote:

> On Mon, Feb 14, 2011 at 11:41:42AM -0800, jacob pan wrote:
> > On Sat, 12 Feb 2011 15:29:07 -0800
> > Matt Helsley <matthltc@us.ibm.com> wrote:
> >
> > > On Fri, Feb 11, 2011 at 11:10:44AM -0800,
> > > jacob.jun.pan@linux.intel.com wrote:
> > > > From: Jacob Pan <jacob.jun.pan@linux.intel.com>
>
> <snip>
>
> > > > cgroup. +To make the tasks frozen at 90% of the time every 5
> > > > seconds, do: +
> > > > +[root@localhost]# echo 90 > freezer.duty_ratio_pct
> > > > +[root@localhost]# echo 5000 > freezer.period_ms
> > > > +
> > > > +After that, the application in this freezer cgroup will only be
> > > > +allowed to run at the following pattern.
> > > > +   __   __   __
> > > > + | |<-- 90% frozen -->| | | |
> > > > +____| |__________________| |______________| |_____
> > > > +
> > > > + |<---- 5 seconds ---->|
> > >
> > > So most of the time I've been reviewing this I managed to invert
> > > it! I imagined "duty" meant the tasks were "on duty" ie runnable
> > > ie thawed. But according this this documentation it's the
> > > opposite...
> > >
> > My logic is that since this is a freezer, so positive logic should
> > be frozen instead of thaw.
>
> Yup, I figured as much. That's the reason I didn't ask you to swap the
> meaning of the ratio values.
>
> > > I've reviewed my review and now my comments are consistent with
> > > the above. :) However it makes me wonder if there are better
> > > names which would avoid this confusion.
> > >
> > How about frozen_time_pct?
>
using frozen_time_percentage in v9

> Much better! nit: I don't know if _pct is obvious to everyone but it
> only takes 4 more characters to make it so..
>
> > > > diff --git a/kernel/cgroup_freezer.c b/kernel/cgroup_freezer.c
>
> <snip>
>
> > > > +static void freezer_work_fn(struct work_struct *work)
> > > > +{
> > > > + struct freezer *freezer;
> > > > + unsigned long delay_jiffies = 0;
> > > > + enum freezer_state goal_state;
> > > > +
> > >
> > > Looking better. There are alot of field accesses here which can
> > > race with writes to the cgroup's duty ratio and period files.
> > > They should be protected. Perhaps we can reuse the freezer spin
> > > lock. That also has the benefit that we can eliminate the
> > > toggle.freeze_thaw bit I think:
> > >
> > I did think about the race, it does exist. but in practice. My
> > thought was that since freezer_change_state() holds the spin_lock
> > of the freezer, the race with writes to params are harmless, it
> > just means the new period or ratio will take effect in the next
> > period.
>
> I considered this but didn't like the idea of relying on it. More
> below.
>
fair enough, I added spin lock in v9
> > In terms of using freezer spin lock to eliminate toggle flag, I am
> > not sure if i know how to do that. Are you suggesting based on
> > whether the spin lock is taken or not, we can decide the toggle?
> > but the freeze spin lock is used by other functions as well not
> > just the delay work here. I guess I have missed something.
>
> I was thinking that with the lock held you can check the state
> variable and just do the "opposite" of what it indicates:
>
> state TODO
> FROZEN THAWED
> FREEZING THAWED
> THAWED FROZEN
>
> Then you don't need the separate bit to indicate which state it should
> try to change to next.
>]

good idea, using it in v9
> > > > +
> > > > + freezer = container_of(work, struct freezer,
> > > > freezer_work.work);
> > > > + /* toggle between THAWED and FROZEN state.
> > > > + * thaw if freezer->toggle.freeze_thaw = 0; freeze
> > > > otherwise
> > > > + * skip the first round if already in the target
> > > > states.
> > > > + */
> > >
> > > spin_lock(&freezer->lock);
> > >
> > > > + if ((freezer->toggle.freeze_thaw && freezer->state ==
> > > > CGROUP_FROZEN) ||
> > > > + (!freezer->toggle.freeze_thaw &&
> > > > +   freezer->state == CGROUP_THAWED)) {
> > > > + delay_jiffies = 0;
> > >
> > > This looks wrong. We should never schedule freezer work delayed
> > > by 0 jiffies -- even if the delayed work API allows it. With
> > > 0-length delays I'd worry that we could peg the CPU in an obscure
> > > infinite loop.
> > >
> > > I think you can safely eliminate this block and the "exit_toggle"
> > > label.
> > >
> > Good point. My initial thought was that since the period for
> > targeted usage is quite long, e.g. 30 sec., we want to start the
> > duty ratio right away. But that shouldn't matter since we already
> > schedule work based on the new ratio/period.
> > > > + goto exit_toggle;
> > > > + } else if (freezer->toggle.freeze_thaw) {
> > >
> > > if (freezer->state == CGROUP_THAWED) {
> > >
> > > > + goal_state = CGROUP_FROZEN;
> > > > + delay_jiffies =
> > > > msecs_to_jiffies(freezer->duty.ratio *
> > > > +
> > > > freezer->duty.period_pct_ms);
> > > > + } else {
> > > > + goal_state = CGROUP_THAWED;
> > > > + delay_jiffies = msecs_to_jiffies((100 -
> > > > freezer->duty.ratio) *
> > > > +
> > > > freezer->duty.period_pct_ms);
> > > > + }

> > > > + freezer_change_state(freezer->css.cgroup, goal_state);
> > >
> > > __freezer_change_state(freezer->css.cgroup, goal_state);
> > > spin_unlock(&freezer->lock);
> > >
> > > (where the __freezer_change_state() function expects to already
> > > have the freezer lock -- you can make that your first patch and
> > > this your second)
> > >
> > > But you ought to double check the lock ordering, may-sleep, and
> > > whether the _irq variants are correct.
> > >
> > I agree with the change to deal with race but again, I don't see the
> > harm of the race other than delaying one period. If the user has to
> > change period and duty ratio separately, there will always be a
> > window of unwanted params unless user disable it first.
>
> But those windows could be pretty large if you delay it that long and
> that could be confusing. With the lock will it be delayed?
>
There is no way to prevent such window. e.g. if user wants to change
from 50% of 3 second period to 90% of 2 second period, it will get a
mismatch for one period. lock does not help here. Users just have to
disable the toggling mode if they want to prevent that.

> > Can you please explain the problem might be caused by the race.
> > > > > +
> > > > > +exit_toggle:
> > > > > + schedule_delayed_work(&freezer->freezer_work,
> > > > > delay_jiffies);
> > > > > + freezer->toggle.freeze_thaw ^= 1;
> > > >
> > > > This looks wrong. It looks like there could be a race between the
> > > > next scheduled work and the toggling of the freeze_thaw value.
> > > > This race would cause the cgroup to miss one or more duty cycles.
> > > > You'd have to re-order these two lines and probably need an smp
> > > > barrier of one sort or another between them.
> > > >
> > > I will fix that. good point.
> > >
> > >
> > > > Of course if you use locking and eliminate the toggle.freeze_thaw
> > > > field as I've suggested then you can ignore this.
> > > >
> > > same as before, not sure how to reuse the freezer spin lock for
> > > this. can you please explain.
>
> Well you just need to acquire the spin lock when you enter the timer

> function, calculate delay_jiffies and goal state without the need for
> the freeze_thaw field, then drop the lock.
>
> At that point you can initiate the state change and then do the
> schedule_delayed_work().
>
> <snip>

done in v9.
>
> > > > @@ -360,7 +435,18 @@ static int freezer_write(struct cgroup
> > > > *cgroup, goal_state = CGROUP_FROZEN;
> > > > else
> > > > return -EINVAL;
> > > > -
> > > > + /* we should stop duty ratio toggling if user wants to
> > > > + * force change to a valid state.
> > > > + */
> > > > + freezer = cgroup_freezer(cgroup);
> > > > + if (freezer->duty.period_pct_ms && freezer->duty.ratio
> > > > < 100) {
> > >
> > > If duty.ratio is 100 then the delayed work should be cancelled
> > > too. In fact it doesn't matter what the duty.ratio or
> > > period_pct_ms are -- writes to this file should always disable
> > > the duty cycle. Thus you can omit the above if () and do this:
> > >
> > > > + if (freezer->toggle.enabled)
> > >
> > agreed, i will fix it.
> > > {
> > >
> > > > +
> > > > cancel_delayed_work_sync(&freezer->freezer_work);
> > > > + freezer->duty.ratio = 0;
> > >
> > > Actually, shouldn't this be 0 if the cgroup is going to be thawed
> > > and 100 if it's going to be frozen?
> > >
> > I am using 0 as an invalid value when toggle is not enabled.
> > perhaps i should introduce -1 such that when user override occurs
> > we just do freezer->toggle.enabled = 0;
> > freezer->duty.ratio = -1;
> > freezer->duty.period_pct_ms = -1;
> > then we can allow and or 100% where both will turn off toggle as
> > well.
>
> Nope. Then you will have "negative" sleeps in the timer function which

> just begs for misinterpretation. For example, look at msleep -- it
> takes an unsigned int. This coupled with the race is a recipe for an
> unintended long sleeps.
>
> We don't need special values here -- just the enabled flag. When
> enabled you can report the ratio from the ratio field. When not
> enabled you can report the ratio by looking at the freezer state
> (might want to do an update_if_frozen() first). Or you could just
> have writes to the freezer.state always update the ratio. You don't
> need to vary period_pct_ms at all when enabling/disabling the duty
> ratio.
>
> That way at all times the values reported to userspace are consistent,
> there are no "special" values, and writes to either file trigger the
> correct changes between enabled/disabled and freezer state. For
> example you might do:
>
> $ echo 0 > freezer.frozen_time_pct
> $ cat freezer.state
> THAWED
> $ cat freezer.frozen_time_pct
> 0
> $ echo 100 > freezer.frozen_time_pct
> $ cat freezer.state
> FREEZING
> $ cat freezer.state
> FROZEN
> $ cat freezer.frozen_time_pct
> 100
> $ echo THAWED > freezer.state
> $ cat freezer.frozen_time_pct
> 0
>
fixed in v9
> >
> > > > + freezer->duty.period_pct_ms = 0;
> > >
> > > I think this should always be a non-zero value -- even when duty
> > > cycling is disabled. Perhaps:
> > >
> > >   freezer->duty.period_pct_ms = 1000/100;
> > >
> > > So it's clear the default period is 1000ms and one percent of it
> > > is 10ms.
> > >
> > > (NOTE: To make it always non-zero you also need to add one line
> > > to the cgroup initialization code in freezer_create()).
> > how about -1 as suggested above.

> >
> > >
> > > > + freezer->toggle.enabled = 0;
> > > > + pr_info("freezer state changed by user, stop
> > > > duty ratio\n");
> > >
> > > nit: I don't think this pr_info() is terribly useful.
> > >
> > I will make it pr_debug instead.
>
> Hmm, OK I suppose.
>
> <snip>
>
> > > > +
> > > > + switch (cft->private) {
> > > > + case FREEZER_DUTY_RATIO:
> > > > + if (val >= 100) {
> > >
> > > ratio == 100 ought to be allowed too.
> > Ok, 100% frozen would be equivalent to echo FROZEN > freezer.state.
> > I will document these corner cases. I think as long as these
> > behaviors
>
> Actually the tricky part to document has nothing to do with the
> value of frozen_time_pct being 0 or 100. It has everything to do with
> which write happened "last".
>
> For all freezer.state values the value that should be read from
> freezer.frozen_time_pct depends on whether it's due to a write to
> freezer.frozen_time_pct or freezer.state. Writes directly to
> freezer.frozen_time_pct should show what was written (if it's in 0-100
> inclusive). Writes to freezer.state should appear to modify
> freezer.frozen_time_pct to be consistent.
>
> That's easily managed within the respective write functions.
> Alternately, the read function for freezer.frozen_time_pct could check
> the enabled bit and use that to switch which method it uses to
> "read" the value.
>
> Note how the period has nothing to do with any of this. It's just
> a timescale factor which ensures there's a maximum frequency at which
> we can change between FROZEN and THAWED (soon to be 1HZ).
>
little confused, there is no need to "appear to modify" things. I just
made freezer.state write in sync with ratio value.

> > are documented well so that user can get the anticipated results,

> > the interface does matter that much.
> >
> > >
> > > > +   ret = -EINVAL;
> > > > +   goto exit;
> > > > + }
> > >
> > > Add:
> > >
> > >   spin_lock_irq(&freezer->lock);
> > >
> > > > + freezer->duty.ratio = val;
> > >
> > > Because this can race with the delayed work.
> > >
> > > > + break;
> > > > + case FREEZER_PERIOD:
> > > > + do_div(val, 100);
> > > > + freezer->duty.period_pct_ms = val;
> > >
> > > This can race with the delayed work. Also I think that a 0ms
> > > period_pct_ms should be disallowed. Otherwise all the work delays
> > > go to zero and we'll probably peg the CPU so that it's just
> > > spinning the freezer state between FROZEN and THAWED and doing
> > > nothing else.
> > >
> > 0 or low number of period is dangerous for reason as you mentioned,
> > I think I should move back to one second resolution. Especially, we
> > are using common workqueue now.
>
> Sounds good.
>
> <snip>
>
> > > > + /* only use delayed work when valid params are given.
> > > > */
> > > > + if (freezer->duty.ratio && freezer->duty.period_pct_ms
> > > > &&
> > > > + !freezer->toggle.enabled) {
> > > > + pr_debug("starting duty ratio mode\n");
> > > > + INIT_DELAYED_WORK(&freezer->freezer_work,
> > > > freezer_work_fn);
> > > > + freezer->toggle.enabled = 1;
> > > > + schedule_delayed_work(&freezer->freezer_work,
> > > > 0);
> > > > + } else if ((!freezer->duty.ratio
> > > > || !freezer->duty.period_pct_ms) &&
> > > > + freezer->toggle.enabled) {

> > > > + pr_debug("invalid param, stop duty ratio mode
> > > > %p\n",
> > > > +   freezer->freezer_work.work.func);
> > > > +
> > > > cancel_delayed_work_sync(&freezer->freezer_work);
> > > > + freezer->toggle.enabled = 0;
> > > > + /* thaw the cgroup if we are not toggling */
> > > > + freezer_change_state(freezer->css.cgroup,
> > > > CGROUP_THAWED); +
> > > > + }
> > >
> > > I don't think this is as readable as (assuming the change above to
> > > disallow setting period_pct_ms to 0):
> > >
> > > if (freezer->duty.ratio == 100) {
> > >   freezer_disable_duty_cycling(freezer); /* see
> > > helper below */ __freezer_change_state(freezer->css.cgroup,
> > > CGROUP_FROZEN); } else if (freezer->duty.ratio == 0) {
> > >   freezer_disable_duty_cycling(freezer);
> > >   __freezer_change_state(freezer->css.cgroup,
> > > CGROUP_THAWED); } else {
> > >   if (freezer->toggle.enabled)
> > >   goto exit; /* Already enabled */
> > >   INIT_DELAYED_WORK(&freezer->freezer_work,
> > > freezer_work_fn); freezer->toggle.enabled = 1;
> > >   schedule_delayed_work(&freezer->freezer_work, 0);
> > > }
> > > spin_unlock_irq(&freezer->lock);
>
> Something to look into: you might even be able to factor this chunk to
> share it between both cgroup file write functions.
>
I did some consolidation in v9, the checks for cancellation are all
merged in one function now.
I did not introduce __freezer_change_state() but rather just use the
internal freeze and unfreeze function w/o lock (when caller already has
the lock).

Thanks,

Jacob

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containe rs