
Subject: Re: [PATCH 1/1, v6] cgroup/freezer: add per freezer duty ratio control
Posted by [Matt Helsley](#) on Thu, 10 Feb 2011 18:58:52 GMT

[View Forum Message](#) <> [Reply to Message](#)

On Thu, Feb 10, 2011 at 11:15:22AM +0200, Kirill A. Shutemov wrote:

> On Wed, Feb 09, 2011 at 07:04:42PM -0800, Matt Helsley wrote:

```
>>> +{
>>> + struct cgroup *cgroup = (struct cgroup *)data;
>>> + struct freezer *freezer = cgroup_freezer(cgroup);
>>> +
>>> + do {
>>> + if (freezer->duty.ratio < 100 && freezer->duty.ratio > 0 &&
>>> + freezer->duty.period_pct_ms) {
>>> + if (try_to_freeze_cgroup(cgroup, freezer))
>>> + pr_info("cannot freeze\n");
>>> + msleep(freezer->duty.period_pct_ms *
>>> + freezer->duty.ratio);
>>> + unfreeze_cgroup(cgroup, freezer);
>>> + msleep(freezer->duty.period_pct_ms *
>>> + (100 - freezer->duty.ratio));
>>> + } else {
>>> + sleep_on(&freezer_wait);
>>> + pr_debug("freezer thread wake up\n");
>>> + }
>>> + } while (!kthread_should_stop());
>>> + return 0;
>>> +}
```

>> Seems to me you could avoid the thread-per-cgroup overhead and the
>> sleep-loop code by using one timer-per-cgroup. When the timer expires
>> you freeze/thaw the cgroup associated with the timer, setup the next
>> wakeup timer, and use only one kernel thread to do it all. If you
>> use workqueues you might even avoid the single kernel thread.

>> Seems to me like that'd be a good fit for embedded devices.

>
> I proposed to use delayed workqueues (schedule_delayed_work()).

Even better.

```
>
>>> +#define FREEZER_KH_PREFIX "freezer_"
>>> +static int freezer_write_param(struct cgroup *cgroup, struct cftype *cft,
>>> + u64 val)
>>> +{
>>> + struct freezer *freezer;
>>> + char thread_name[32];
>>> + int ret = 0;
```

```

>>> +
>>> + freezer = cgroup_freezer(cgroup);
>>> +
>>> + if (!cgroup_lock_live_group(cgroup))
>>> + return -ENODEV;
>>> +
>>> + switch (cft->private) {
>>> + case FREEZER_DUTY_RATIO:
>>> + if (val >= 100 || val < 0) {
>>> + ret = -EINVAL;
>>> + goto exit;
>>> + }
>>> + freezer->duty_ratio = val;
>>
>> Why can't val == 100? At that point it's always THAWED and no kernel
>> thread is necessary (just like at 0 it's always FROZEN and no kernel
>> thread is necessary).
>
> val == 100 is interface abuse, I think. I just turn off the feature, if
> you want.

```

And how is userspace supposed to do that at runtime if we can't disable it by writing to the state file (see below)? Then I don't see anyway to get rid of the duty cycling unless you clear out the cgroup and recreate it.

Frankly, I think 0 and 100 percent aren't interface abuse. Anybody who knows it's a percent value will naturally try to put 0 or 100 there.

```

>>> static struct cftype files[] = {
>>> {
>>> .name = "state",
>>> .read_seq_string = freezer_read,
>>> .write_string = freezer_write,
>>>
>>
>> It's not clear what should happen when userspace writes the state
>> file after writing a duty_ratio_pct.
>
> It should return -EBUSY, I think.

```

Ahh, that is another solution I hadn't considered. That further proves my point though :). It's not obvious what should happen and that's a red-flag that we're defining policy and should be careful which solution we select.

```

>
>>> },
>>> + {

```

```
>>> + .name = "duty_ratio_pct",
>>> + .private = FREEZER_DUTY_RATIO,
>>> + .read_u64 = freezer_read_duty_ratio,
>>> + .write_u64 = freezer_write_param,
>>> + },
>>
>> nit: Why use a u64 for a value that can only be 0-100? (or perhaps
>> 0-1000 if you wanted sub-1% granularity...)
>
> .read_u64/.write_64 is a standard cgroup's interface.
```

Oops -- I was thinking there was a smaller variant of these.

Cheers,
-Matt Helsley

Containers mailing list
Containers@lists.linux-foundation.org
<https://lists.linux-foundation.org/mailman/listinfo/containers>
