Subject: Re: [PATCH 1/1, v7] cgroup/freezer: add per freezer duty ratio control
Posted by Matt Helsley on Sat, 12 Feb 2011 23:29:07 GMT
View Forum Message <> Reply to Message

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>
>
> Freezer subsystem is used to manage batch jobs which can start
> stop at the same time. However, sometime it is desirable to let
> the kernel manage the freezer state automatically with a given
> duty ratio.
> For example, if we want to reduce the time that backgroup apps
> are allowed to run we can put them into a freezer subsystem and
> set the kernel to turn them THAWED/FROZEN at given duty ratio.
>
> This patch introduces two file nodes under cgroup
> freezer.duty_ratio_pct and freezer.period_sec

Again: I don't think this is the right approach in the long term.
It would be better not to add this interface and instead enable the
cpu cgroup subsystem for non-rt tasks using a similar duty ratio
concept..

Nevertheless, I've added some feedback on the code for you here :).

>
> Usage example: set period to be 5 seconds and frozen duty ratio 90%
> [root@localhost aoa]# echo 90 > freezer.duty_ratio_pct
> [root@localhost aoa]# echo 5000 > freezer.period_ms
>
> Signed-off-by: Jacob Pan <jacob.jun.pan@linux.intel.com>
> ---
> Documentation/cgroups/freezer-subsystem.txt | 23 ++++
> kernel/cgroup_freezer.c | 153 ++++++++++++++++++++++++++-
> 2 files changed, 174 insertions(+), 2 deletions(-)
>
> diff --git a/Documentation/cgroups/freezer-subsystem.txt
b/Documentation/cgroups/freezer-subsystem.txt
> index 41f37fe..2022b32 100644
> --- a/Documentation/cgroups/freezer-subsystem.txt
> +++ b/Documentation/cgroups/freezer-subsystem.txt
> @@ -100,3 +100,26 @@ things happens:
>   and returns EINVAL)
> 3) The tasks that blocked the cgroup from entering the "FROZEN"
>   state disappear from the cgroup's set of tasks.
> +
> +In embedded systems, it is desirable to manage group of applications
> +for power saving. E.g. tasks that are not in the foreground may be

> +frozen and unfrozen periodically to save power without affecting user
> +experience. In this case, user/management software can attach tasks
> +into freezer cgroup then specify duty ratio and period that the
> +managed tasks are allowed to run.
> +
> +Usage example:
> +Assuming freezer cgroup is already mounted, application being managed
> +are included the "tasks" file node of the given freezer 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...

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.

> diff --git a/kernel/cgroup_freezer.c b/kernel/cgroup_freezer.c
> index e7bebb7..aaa91ca 100644
> --- a/kernel/cgroup_freezer.c
> +++ b/kernel/cgroup_freezer.c
> @@ -21,6 +21,7 @@
> #include <linux/uaccess.h>
> #include <linux/freezer.h>
> #include <linux/seq_file.h>
> +#include <linux/kthread.h>
>
> enum freezer_state {
>   CGROUP_THAWED = 0,
> @@ -28,12 +29,35 @@ enum freezer_state {
>   CGROUP_FROZEN,
> };
>
> +enum duty_ratio_params {
> + FREEZER_DUTY_RATIO = 0,
> + FREEZER_PERIOD,

> +};
> +
> +struct freezer_toggle {
> + unsigned int enabled:1;
> + unsigned int freeze_thaw:1; /* 1: freeze 0: thaw */
> +} __packed;
> +
> +struct freezer_duty {
> + u32 ratio; /* percentage of time frozen */
> + u32 period_pct_ms; /* one percent of the period in miliseconds */
> +};

I'd rather see you merge these two structures -- I don't see the value
of keeping them separate nor of packing the struct. You could also merge
the work item in there:

struct freezer_duty {
 struct delayed_work freezer_work; /* work to duty-cycle a cgroup */

 u32 ratio; /* percentage of time frozen */
 u32 period_pct_ms; /* one percent of the period in miliseconds */
 unsigned int enabled:1;
 unsigned int freeze_thaw:1; /* 1: freeze 0: thaw */
};

(I'm going to make the rest of my comments without assuming you've done
this because it'll make them easier to follow given the context)

> +
> struct freezer {
>   struct cgroup_subsys_state css;
>   enum freezer_state state;
> + struct freezer_duty duty;
> + struct delayed_work freezer_work; /* work to duty-cycle a cgroup */
> + struct freezer_toggle toggle;
>   spinlock_t lock; /* protects _writes_ to state */
> };
>
> +static int try_to_freeze_cgroup(struct cgroup *cgroup, struct freezer *freezer);
> +static void unfreeze_cgroup(struct cgroup *cgroup, struct freezer *freezer);
> +static int freezer_change_state(struct cgroup *cgroup,
> +   enum freezer_state goal_state);
> +
> static inline struct freezer *cgroup_freezer(
>   struct cgroup *cgroup)
> {
> @@ -63,6 +87,41 @@ int cgroup_freezing_or_frozen(struct task_struct *task)
>   return result;

> }
>
> +static DECLARE_WAIT_QUEUE_HEAD(freezer_wait);
> +
> +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:

> +
> + 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.

> + 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.

> +
> +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.

Of course if you use locking and eliminate the toggle.freeze_thaw field
as I've suggested then you can ignore this.

> +}
> +
> /*
>   * cgroups_write_string() limits the size of freezer state strings to
>   * CGROUP_LOCAL_BUFFER_SIZE
> @@ -150,7 +209,12 @@ static struct cgroup_subsys_state *freezer_create(struct
cgroup_subsys *ss,
> static void freezer_destroy(struct cgroup_subsys *ss,
>   struct cgroup *cgroup)
> {
> - kfree(cgroup_freezer(cgroup));
> + struct freezer *freezer;
> +
> + freezer = cgroup_freezer(cgroup);
> + if (freezer->toggle.enabled)
> + cancel_delayed_work_sync(&freezer->freezer_work);
> + kfree(freezer);
> }
>
> /*
> @@ -282,6 +346,16 @@ static int freezer_read(struct cgroup *cgroup, struct cftype *cft,

> return 0;
> }
>
> +static u64 freezer_read_duty_ratio(struct cgroup *cgroup, struct cftype *cft)
> +{
> + return cgroup_freezer(cgroup)->duty.ratio;
> +}
> +
> +static u64 freezer_read_period(struct cgroup *cgroup, struct cftype *cft)
> +{
> + return cgroup_freezer(cgroup)->duty.period_pct_ms * 100;
> +}
> +
> static int try_to_freeze_cgroup(struct cgroup *cgroup, struct freezer *freezer)
> {
> struct cgroup_iter it;
> @@ -353,6 +427,7 @@ static int freezer_write(struct cgroup *cgroup,
> {
> int retval;
> enum freezer_state goal_state;
> + struct freezer *freezer;
>
> if (strcmp(buffer, freezer_state_strs[CGROUP_THAWED]) == 0)
> goal_state = CGROUP_THAWED;
> @@ -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)

     {

> + 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?

> + 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()).

> + 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.

> + }
>   if (!cgroup_lock_live_group(cgroup))
>   return -ENODEV;
>   retval = freezer_change_state(cgroup, goal_state);
> @@ -368,12 +454,75 @@ static int freezer_write(struct cgroup *cgroup,
>   return retval;
> }
>
> +#define FREEZER_KH_PREFIX "freezer_"

This #define is unused.

> +static int freezer_write_param(struct cgroup *cgroup, struct cftype *cft,
> + u64 val)
> +{
> + struct freezer *freezer;
> + int ret = 0;
> +
> + freezer = cgroup_freezer(cgroup);
> +
> + if (!cgroup_lock_live_group(cgroup))
> + return -ENODEV;

Because the delayed work function does not lock the cgroup this whole
function can race with the delayed work. So we need to be careful about how
we set/test all of these new fields -- we probably want to reuse the
freezer state lock for this so I've sprinkled some lock/unlock bits with
my comments below.

> +
> + switch (cft->private) {
> + case FREEZER_DUTY_RATIO:
> + if (val >= 100) {

ratio == 100 ought to be allowed too.

> +   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.

Finally, I wonder if a 64-bit value is really necessary -- is a period
longer than roughly 50 days really necessary?

In summary, couldn't you just do something like:

  if ((val < 100) || (val > UINT_MAX)) {
   /* Too much time or may have rounded down to 0 */
   ret = -EINVAL;
   goto exit;
  }
  val = (unsigned int)val / 100;
  spin_lock_irq(&freezer->lock);
  freezer->duty.period_pct_ms = val;

> + break;
> + default:
> + BUG();
> + }
> +

```
> + /* 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);
```

Where the helper is:

```
static void freezer_disable_duty_cycling(struct freezer *freezer)
{
 if (freezer->toggle.enabled) {
  cancel_delayed_work_sync(&freezer->freezer_work);
  freezer->toggle.enabled = 0;
 }
}
```

which could also be called from freezer_write() I think.

Cheers,
-Matt Helsley

_____

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