Subject: [RFC PATCH 0/2] per cgroup timer slack and freezer duty cycle
Posted by jacob.jun.pan on Wed, 01 Dec 2010 19:00:10 GMT
View Forum Message <> Reply to Message

From: Jacob Pan <jacob.jun.pan@linux.intel.com>

Hi,

These are two cgroup related patches for supporting power friendly
behavior of batch jobs in a cgroup.
The main usage is for managing background applications where more
relaxed timer rounding effect and reduced CPU time are desired.

Thanks,

Jacob

Jacob Pan (2):
  cgroup: add per cgroup timer_slack_ns
  cgroup/freezer: add per freezer duty ratio control

 include/linux/cgroup.h | 1 +
 include/linux/init_task.h | 3 +-
 kernel/cgroup.c | 43 ++++++++++++
 kernel/cgroup_freezer.c | 157 ++-
 4 files changed, 201 insertions(+), 3 deletions(-)

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

Subject: [RFC PATCH 1/2] cgroup: add per cgroup timer_slack_ns
Posted by jacob.jun.pan on Wed, 01 Dec 2010 19:00:11 GMT
View Forum Message <> Reply to Message

From: Jacob Pan <jacob.jun.pan@linux.intel.com>

Per task timer_slack_ns was introduced a while ago to allow tuning
timer rounding behavior such that tasks can be made mode power friendly.
This patch introduces per cgroup timer slack value which will override
the default timer slack value once a task is attached to a cgroup. By
default, the root cgroup timer slack value is set to the same 50us as
in all the tasks.
At runtime, user can choose to change cgroup timer slack value by
echo xxx > cgroup.timer_slack_ns

The usage of such feature can be found in mobile devices where certain
background apps are attached to a cgroup and minimum wakeups are desired.

Signed-off-by: Jacob Pan <jacob.jun.pan@linux.intel.com>
---
 include/linux/cgroup.h | 1 +
 include/linux/init_task.h | 3 ++-
 kernel/cgroup.c | 43 +++
 3 files changed, 46 insertions(+), 1 deletions(-)

diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
index ed4ba11..afac9bb 100644
--- a/include/linux/cgroup.h
+++ b/include/linux/cgroup.h
@@ -194,6 +194,7 @@ struct cgroup_pidlist {

 struct cgroup {
  unsigned long flags; /* "unsigned long" so bitops work */
+ unsigned long timer_slack_ns;

  /*
   * count users of this cgroup. >0 means busy, but doesn't
diff --git a/include/linux/init_task.h b/include/linux/init_task.h
index 1f8c06c..5860af6 100644
--- a/include/linux/init_task.h
+++ b/include/linux/init_task.h
@@ -110,6 +110,7 @@ extern struct cred init_cred;
 # define INIT_PERF_EVENTS(tsk)
 #endif

+#define TIMER_SLACK_NS_DEFAULT (50000) /* 50 usec default slack */
 /*
  * INIT_TASK is used to set up the first task table, touch at
  * your own risk!. Base=0, limit=0x1fffff (=2MB)
@@ -163,7 +164,7 @@ extern struct cred init_cred;
  .cpu_timers = INIT_CPU_TIMERS(tsk.cpu_timers), \
  .fs_excl = ATOMIC_INIT(0),   \
  .pi_lock = __RAW_SPIN_LOCK_UNLOCKED(tsk.pi_lock), \
- .timer_slack_ns = 50000, /* 50 usec default slack */ \
+ .timer_slack_ns = TIMER_SLACK_NS_DEFAULT,   \
  .pids = {   \
   [PIDTYPE_PID] = INIT_PID_LINK(PIDTYPE_PID), \
   [PIDTYPE_PGID] = INIT_PID_LINK(PIDTYPE_PGID), \
diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index 66a416b..0e0a254 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -57,6 +57,7 @@

```
 #include <linux/vmalloc.h> /* TODO: replace with more sophisticated array */
 #include <linux/eventfd.h>
 #include <linux/poll.h>
+#include <linux/init_task.h>

 #include <asm/atomic.h>

@@ -1324,6 +1325,7 @@ static void init_cgroup_root(struct cgroupfs_root *root)
  root->number_of_cgroups = 1;
  cgrp->root = root;
  cgrp->top_cgroup = cgrp;
+ cgrp->timer_slack_ns = TIMER_SLACK_NS_DEFAULT;
  init_cgroup_housekeeping(cgrp);
 }

@@ -1787,6 +1789,7 @@ int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
   goto out;
  }
  rcu_assign_pointer(tsk->cgroups, newcg);
+ tsk->timer_slack_ns = cgrp->timer_slack_ns;
  task_unlock(tsk);

  /* Update the css_set linked lists if we're using them */
@@ -3049,6 +3052,38 @@ static int cgroup_write_notify_on_release(struct cgroup *cgrp,
  return 0;
 }

+static u64 cgroup_read_timer_slack_ns(struct cgroup *cgrp,
+    struct cftype *cft)
+{
+ return cgrp->timer_slack_ns;
+}
+
+static int cgroup_write_timer_slack_ns(struct cgroup *cgrp,
+    struct cftype *cft,
+    u64 val)
+{
+ struct cgroup_iter it;
+ struct task_struct *task;
+
+ /* TODO: upper range checking for max slack */
+ if (val)
+  cgrp->timer_slack_ns = val;
+ else {
+  printk(KERN_ERR "cgroup %s: invalid timer slack value %llu\n",
+   cgrp->dentry->d_name.name, val);
+  return -EINVAL;
+ }
```

```
+
+ /* change timer slack value for all tasks in the cgroup */
+ cgroup_iter_start(cgrp, &it);
+ while ((task = cgroup_iter_next(cgrp, &it)))
+  task->timer_slack_ns = val;
+
+ cgroup_iter_end(cgrp, &it);
+
+ return 0;
+}
+
 /*
  * Unregister event and free resources.
  *
@@ -3268,6 +3303,13 @@ static struct cftype files[] = {
   .read_u64 = cgroup_clone_children_read,
   .write_u64 = cgroup_clone_children_write,
  },
+ {
+  .name = CGROUP_FILE_GENERIC_PREFIX "timer_slack_ns",
+  .read_u64 = cgroup_read_timer_slack_ns,
+  .write_u64 = cgroup_write_timer_slack_ns,
+  .mode = S_IRUGO | S_IWUSR,
+ },
+
 };

 static struct cftype cft_release_agent = {
@@ -3393,6 +3435,7 @@ static long cgroup_create(struct cgroup *parent, struct dentry *dentry,
  cgrp->parent = parent;
  cgrp->root = parent->root;
  cgrp->top_cgroup = parent->top_cgroup;
+ cgrp->timer_slack_ns = TIMER_SLACK_NS_DEFAULT;

  if (notify_on_release(parent))
   set_bit(CGRP_NOTIFY_ON_RELEASE, &cgrp->flags);
--
1.7.0.4
```


Subject: [RFC PATCH 2/2] cgroup/freezer: add per freezer duty ratio control
Posted by jacob.jun.pan on Wed, 01 Dec 2010 19:00:12 GMT

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
freezer.duty_ratio and freezer.period_ms
Usage example, set period to be 50 ms and frozen duty ratio 90%
[root@localhost aoa]# echo 90 > freezer.duty_ratio
[root@localhost aoa]# echo 50 > freezer.period_ms

Each freezer will be controlled by its own kernel thread which is
in sleep unless there is a state/parameter change.

Signed-off-by: Jacob Pan <jacob.jun.pan@linux.intel.com>
---
 kernel/cgroup_freezer.c | 157 ++-
 1 files changed, 155 insertions(+), 2 deletions(-)

diff --git a/kernel/cgroup_freezer.c b/kernel/cgroup_freezer.c
index e7bebb7..07941fa 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 +29,23 @@ enum freezer_state {
  CGROUP_FROZEN,
 };

+struct freezer_duty {
+ u32 ratio; /* percentage of time allowed to run */
+ u32 period; /* period in seconds */
+};
+
 struct freezer {
  struct cgroup_subsys_state css;

```
  enum freezer_state state;
+ struct freezer_duty duty;
+ struct task_struct *fkh;
  spinlock_t lock; /* protects _writes_ to state */
 };

+static struct task_struct *freezer_task;
+static int try_to_freeze_cgroup(struct cgroup *cgroup, struct freezer *freezer);
+static void unfreeze_cgroup(struct cgroup *cgroup, struct freezer *freezer);
+
 static inline struct freezer *cgroup_freezer(
   struct cgroup *cgroup)
 {
@@ -63,6 +75,34 @@ int cgroup_freezing_or_frozen(struct task_struct *task)
  return result;
 }

+static DECLARE_WAIT_QUEUE_HEAD(freezer_wait);
+
+static int freezer_kh(void *data)
+{
+ 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) {
+   if (try_to_freeze_cgroup(cgroup, freezer))
+    pr_info("cannot freeze\n");
+   msleep(freezer->duty.period * freezer->duty.ratio);
+   unfreeze_cgroup(cgroup, freezer);
+   msleep(freezer->duty.period *
+    (100 - freezer->duty.ratio));
+  } else if (freezer->duty.ratio == 100) {
+   if (try_to_freeze_cgroup(cgroup, freezer))
+    pr_info("cannot freeze\n");
+   sleep_on(&freezer_wait);
+  } else {
+   sleep_on(&freezer_wait);
+   pr_debug("freezer thread wake up\n");
+  }
+ } while (!kthread_should_stop());
+ return 0;
+}
+
 /*
  * cgroups_write_string() limits the size of freezer state strings to
  * CGROUP_LOCAL_BUFFER_SIZE
```

```
@@ -150,7 +190,11 @@ 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);
+ kthread_stop(freezer->fkh);
+ kfree(freezer);
 }

 /*
@@ -282,6 +326,51 @@ static int freezer_read(struct cgroup *cgroup, struct cftype *cft,
  return 0;
 }

+static int freezer_read_duty_ratio(struct cgroup *cgroup, struct cftype *cft,
+     struct seq_file *m)
+{
+ struct freezer *freezer;
+ u32 duty_ratio;
+ char result[8];
+
+ if (!cgroup_lock_live_group(cgroup))
+  return -ENODEV;
+
+ freezer = cgroup_freezer(cgroup);
+ spin_lock_irq(&freezer->lock);
+ duty_ratio = freezer->duty.ratio;
+ spin_unlock_irq(&freezer->lock);
+ cgroup_unlock();
+
+ sprintf(result, "%d", duty_ratio);
+ seq_puts(m, result);
+ seq_putc(m, '\n');
+ return 0;
+}
+
+static int freezer_read_period(struct cgroup *cgroup, struct cftype *cft,
+     struct seq_file *m)
+{
+ struct freezer *freezer;
+ u32 period;
+ char result[8];
+
+ if (!cgroup_lock_live_group(cgroup))
```

```
+  return -ENODEV;
+
+ freezer = cgroup_freezer(cgroup);
+ spin_lock_irq(&freezer->lock);
+ period = freezer->duty.period;
+ spin_unlock_irq(&freezer->lock);
+ cgroup_unlock();
+
+ sprintf(result, "%d", period);
+ seq_puts(m, result);
+ seq_putc(m, '\n');
+
+ return 0;
+}
+
 static int try_to_freeze_cgroup(struct cgroup *cgroup, struct freezer *freezer)
 {
  struct cgroup_iter it;
@@ -368,19 +457,83 @@ static int freezer_write(struct cgroup *cgroup,
  return retval;
 }

+static int freezer_write_duty_ratio(struct cgroup *cgroup,
+    struct cftype *cft,
+    const char *buffer)
+{
+ struct freezer *freezer;
+ unsigned long ratio;
+
+ if (strict_strtoul(buffer, 10, &ratio) < 0)
+  return -EINVAL;
+
+ if (ratio > 100) {
+  pr_err("Out of range, ratio should be < 100\n");
+  return -EINVAL;
+ }
+
+ freezer = cgroup_freezer(cgroup);
+ spin_lock_irq(&freezer->lock);
+ freezer->duty.ratio = ratio;
+ spin_unlock_irq(&freezer->lock);
+ wake_up(&freezer_wait);
+
+ return 0;
+}
+
+static int freezer_write_period(struct cgroup *cgroup,
+    struct cftype *cft,
```

```
+    const char *buffer)
+{
+ struct freezer *freezer;
+
+ freezer = cgroup_freezer(cgroup);
+ spin_lock_irq(&freezer->lock);
+ if (strict_strtoul(buffer, 10, &freezer->duty.period) < 0)
+  return -EINVAL;
+ spin_unlock_irq(&freezer->lock);
+ wake_up(&freezer_wait);
+
+ return 0;
+}
+
 static struct cftype files[] = {
  {
   .name = "state",
   .read_seq_string = freezer_read,
   .write_string = freezer_write,
  },
+ {
+  .name = "duty_ratio",
+  .read_seq_string = freezer_read_duty_ratio,
+  .write_string = freezer_write_duty_ratio,
+ },
+ {
+  .name = "period_ms",
+  .read_seq_string = freezer_read_period,
+  .write_string = freezer_write_period,
+ },
 };

+#define FREEZER_KH_PREFIX  "freezer_"
 static int freezer_populate(struct cgroup_subsys *ss, struct cgroup *cgroup)
 {
+ int ret = 0;
+ char thread_name[32];
+ struct freezer *freezer;
+
  if (!cgroup->parent)
   return 0;
- return cgroup_add_files(cgroup, ss, files, ARRAY_SIZE(files));
+
+ freezer = cgroup_freezer(cgroup);
+ ret = cgroup_add_files(cgroup, ss, files, ARRAY_SIZE(files));
+
+ snprintf(thread_name, 32, "%s%s", FREEZER_KH_PREFIX,
+  cgroup->dentry->d_name.name);
```

```
+ freezer->fkh = kthread_run(freezer_kh, (void *)cgroup, thread_name);
+ if (!IS_ERR(freezer_task))
+  return 0;
+ return ret;
 }

 struct cgroup_subsys freezer_subsys = {
--
1.7.0.4
```

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

---

Subject: Re: [RFC PATCH 1/2] cgroup: add per cgroup timer_slack_ns
Posted by jacob.jun.pan on Thu, 02 Dec 2010 20:45:59 GMT
View Forum Message <> Reply to Message

On Wed, 1 Dec 2010 12:17:20 -0800
Paul Menage <menage@google.com> wrote:

> When multiple hierarchies are mounted a task will be in one cgroup on
> each hierarchy - which cgroup's timer_slack_ns value should be used
> for it?
>
> This doesn't belong in the generic cgroups framework. It sounds like
> it's something that should be its own subsystem, or else part of
> something like the cpu subsystem.
good point. the current behavior is that the timer_slack_ns value will
be assigned by the last hierarchy it attaches to, which might not be
desirable.

I guess timer_slack_ns alone cannot qualify for a new subsystem, I will
try to fit it into cpuset.

Thanks for the review,

Jacob

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

---

Subject: Re: [RFC PATCH 1/2] cgroup: add per cgroup timer_slack_ns

Posted by jacob.jun.pan on Thu, 02 Dec 2010 22:06:50 GMT
View Forum Message <> Reply to Message

On Thu, 2 Dec 2010 12:52:36 -0800
Paul Menage <menage@google.com> wrote:

> On Thu, Dec 2, 2010 at 12:45 PM, jacob pan
> <jacob.jun.pan@linux.intel.com> wrote:
> >
> > I guess timer_slack_ns alone cannot qualify for a new subsystem,
>
> I think that it's providing a sufficiently different control from
> existing subsystems that making it a separate subsystem is reasonable.
> The cpuset subsystem definitely doesn't sound like the right place,
> since it's more dealing with numa isolation/policy. The cpu subsystem
> could be, since it also deals with scheduling stuff.
>
> Paul
thanks for the pointer. let me try to add a new subsystem. call
it timer? perhaps will have future expansion.

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

---

Subject: Re: [RFC PATCH 2/2] cgroup/freezer: add per freezer duty ratio control
Posted by jacob.jun.pan on Wed, 02 Feb 2011 19:32:53 GMT
View Forum Message <> Reply to Message

On Tue, 1 Feb 2011 16:23:01 +0200
"Kirill A. Shutemov" <kirill@shutemov.name> wrote:

Thanks for your review, I will fix the error handling. Please also see
my comments on create vs populate.
> On Wed, Dec 01, 2010 at 11:00:12AM -0800,
> jacob.jun.pan@linux.intel.com wrote:
> > From: Jacob Pan <jacob.jun.pan@linux.intel.com>
>
> > +#define FREEZER_KH_PREFIX "freezer_"
> > static int freezer_populate(struct cgroup_subsys *ss, struct
> > cgroup *cgroup) {
> > + int ret = 0;
> > + char thread_name[32];
> > + struct freezer *freezer;
> > +
> >   if (!cgroup->parent)
> >   return 0;

> > - return cgroup_add_files(cgroup, ss, files,
> > ARRAY_SIZE(files)); +
> > + freezer = cgroup_freezer(cgroup);
> > + ret = cgroup_add_files(cgroup, ss, files,
> > ARRAY_SIZE(files)); +
> > + snprintf(thread_name, 32, "%s%s", FREEZER_KH_PREFIX,
> > + cgroup->dentry->d_name.name);
> > + freezer->fkh = kthread_run(freezer_kh, (void *)cgroup,
> > thread_name);
> > + if (!IS_ERR(freezer_task))
> > + return 0;
> > + return ret;
>
> Why do you create on freezer_populate, not on freezer_create?
I want to use the cgroup directory name for the kernel thread. If I do
that in create() instead of populate, I would have to add more changes:
1. pass dentry to the create function since that is called before
cgroup_create_dir()
2. skip the dummy root as part of the initialization.
So I chose to use populate, are there any issues with the functionality?

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