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
View Forum Message <> Reply to Message

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