Subject: [PATCH 1/1, v7] cgroup/freezer: add per freezer duty ratio control
Posted by jacob.jun.pan on Fri, 11 Feb 2011 19:10:44 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 under cgroup
freezer.duty_ratio_pct and freezer.period_sec

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 ---->|
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 */
+};
+
 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;
+
+
+ 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.
+  */
+ if ((freezer->toggle.freeze_thaw && freezer->state == CGROUP_FROZEN) ||
+  (!freezer->toggle.freeze_thaw &&
+   freezer->state == CGROUP_THAWED)) {
+  delay_jiffies = 0;
+  goto exit_toggle;
+ } else if (freezer->toggle.freeze_thaw) {
+  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);
+
+exit_toggle:
+ schedule_delayed_work(&freezer->freezer_work, delay_jiffies);
+ freezer->toggle.freeze_thaw ^= 1;
```

```
+}
+
 /*
  * 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 (freezer->toggle.enabled)
+   cancel_delayed_work_sync(&freezer->freezer_work);
+  freezer->duty.ratio = 0;
+  freezer->duty.period_pct_ms = 0;
+  freezer->toggle.enabled = 0;
+  pr_info("freezer state changed by user, stop duty ratio\n");
+ }
  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_"
+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;
+
+ switch (cft->private) {
+ case FREEZER_DUTY_RATIO:
+  if (val >= 100) {
+   ret = -EINVAL;
+   goto exit;
+  }
+  freezer->duty.ratio = val;
+  break;
+ case FREEZER_PERIOD:
+  do_div(val, 100);
+  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);
+
+ }
+exit:
+ cgroup_unlock();
+ return ret;
+}
+
 static struct cftype files[] = {
  {
   .name = "state",
   .read_seq_string = freezer_read,
   .write_string = freezer_write,
  },
+ {
+  .name = "duty_ratio_pct",
+  .private = FREEZER_DUTY_RATIO,
+  .read_u64 = freezer_read_duty_ratio,
+  .write_u64 = freezer_write_param,
+ },
+ {
+  .name = "period_ms",
+  .private = FREEZER_PERIOD,
+  .read_u64 = freezer_read_period,
+  .write_u64 = freezer_write_param,
+ },
+
 };

 static int freezer_populate(struct cgroup_subsys *ss, struct cgroup *cgroup)
--
1.7.0.4
```

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