Subject: [PATCH 1/1, v9] cgroup/freezer: add per freezer duty ratio control
Posted by jacob.jun.pan on Tue, 15 Feb 2011 22:09:10 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 period and
percentage of frozen time.

This patch introduces two file nodes under cgroup
freezer.period_second
freezer.frozen_time_percentage

Usage example: set period to be 5 seconds and frozen duty ratio 90%
[root@localhost aoa]# echo 90 > freezer.frozen_time_percentage
[root@localhost aoa]# echo 5 > freezer.period_second

Signed-off-by: Jacob Pan <jacob.jun.pan@linux.intel.com>

 Documentation/cgroups/freezer-subsystem.txt | 33 ++++++
 kernel/cgroup_freezer.c | 155 ++++++

```
+THAWED.
+The granularity of the period is second, due to the fact that small
+period may result in greater overhead and causes stability issue since
+common workqueue is used here to perform the management.
+
+Zero value of period_second is not allowed, the default period is 1
+second.
+
+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.frozen_time_percentage
+[root@localhost ]# echo 5 > freezer.period_second
+
+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..32beacb 100644
--- a/kernel/cgroup_freezer.c
+++ b/kernel/cgroup_freezer.c
@@ -28,12 +28,31 @@ enum freezer_state {
  CGROUP_FROZEN,
 };

+enum duty_ratio_params {
+ FREEZER_DUTY_RATIO = 0,
+ FREEZER_PERIOD,
+};
+
+struct freezer_duty {
+ u32 ratio; /* percentage of time frozen */
+ u32 period_pct_ms; /* one percent of the period in miliseconds */
+ u32 enabled;
+};
+
 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 */
  spinlock_t lock; /* protects _writes_ to state */
```

```
 };

+static int freezer_change_state(struct cgroup *cgroup,
+    enum freezer_state goal_state);
+static void unfreeze_cgroup(struct cgroup *cgroup, struct freezer *freezer);
+static int try_to_freeze_cgroup(struct cgroup *cgroup, struct freezer *freezer);
+static void update_if_frozen(struct cgroup *cgroup, struct freezer *freezer);
+
 static inline struct freezer *cgroup_freezer(
   struct cgroup *cgroup)
 {
@@ -63,6 +82,35 @@ int cgroup_freezing_or_frozen(struct task_struct *task)
  return result;
 }

+static void freezer_work_fn(struct work_struct *work)
+{
+ struct freezer *freezer;
+ unsigned long delay_jiffies = 0;
+
+ freezer = container_of(work, struct freezer, freezer_work.work);
+ spin_lock_irq(&freezer->lock);
+ /* toggle between THAWED and FROZEN state. */
+ update_if_frozen(freezer->css.cgroup, freezer);
+ switch (freezer->state) {
+ case CGROUP_FROZEN:
+ case CGROUP_FREEZING:
+  delay_jiffies = msecs_to_jiffies((100 - freezer->duty.ratio) *
+      freezer->duty.period_pct_ms);
+  unfreeze_cgroup(freezer->css.cgroup, freezer);
+  break;
+ case CGROUP_THAWED:
+  delay_jiffies = msecs_to_jiffies(freezer->duty.ratio *
+      freezer->duty.period_pct_ms);
+  try_to_freeze_cgroup(freezer->css.cgroup, freezer);
+  break;
+ default:
+  BUG();
+ }
+
+ schedule_delayed_work(&freezer->freezer_work, delay_jiffies);
+ spin_unlock_irq(&freezer->lock);
+}
+
 /*
  * cgroups_write_string() limits the size of freezer state strings to
  * CGROUP_LOCAL_BUFFER_SIZE
@@ -144,13 +192,19 @@ static struct cgroup_subsys_state *freezer_create(struct
```

```
cgroup_subsys *ss,

  spin_lock_init(&freezer->lock);
  freezer->state = CGROUP_THAWED;
+ freezer->duty.period_pct_ms = 10; /* default to 1 second period */
  return &freezer->css;
 }

 static void freezer_destroy(struct cgroup_subsys *ss,
        struct cgroup *cgroup)
 {
- kfree(cgroup_freezer(cgroup));
+ struct freezer *freezer;
+
+ freezer = cgroup_freezer(cgroup);
+ if (freezer->duty.enabled)
+  cancel_delayed_work_sync(&freezer->freezer_work);
+ kfree(freezer);
 }

 /*
@@ -282,6 +336,16 @@ 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 / 10;
+}
+
 static int try_to_freeze_cgroup(struct cgroup *cgroup, struct freezer *freezer)
 {
  struct cgroup_iter it;
@@ -347,20 +411,56 @@ out:
  return retval;
 }

+static void freezer_check_duty_cycling(struct freezer *freezer, int set_state)
+{
+ if (freezer->duty.enabled) {
+  if (freezer->duty.ratio && freezer->duty.ratio < 100)
+   return;
+  pr_debug("stopping duty ratio mode\n");
```

```
+  cancel_delayed_work_sync(&freezer->freezer_work);
+  freezer->duty.enabled = 0;
+  if (set_state) {
+   update_if_frozen(freezer->css.cgroup, freezer);
+   if (freezer->duty.ratio == 100)
+    try_to_freeze_cgroup(freezer->css.cgroup,
+      freezer);
+   else if (freezer->duty.ratio == 0)
+    unfreeze_cgroup(freezer->css.cgroup, freezer);
+   else
+    BUG();
+  }
+ } else if (freezer->duty.ratio < 100 && freezer->duty.ratio) {
+  pr_debug("starting duty ratio mode\n");
+  INIT_DELAYED_WORK(&freezer->freezer_work, freezer_work_fn);
+  freezer->duty.enabled = 1;
+  schedule_delayed_work(&freezer->freezer_work, 0);
+ }
+}
+
 static int freezer_write(struct cgroup *cgroup,
     struct cftype *cft,
     const char *buffer)
 {
  int retval;
  enum freezer_state goal_state;
+ struct freezer *freezer;

- if (strcmp(buffer, freezer_state_strs[CGROUP_THAWED]) == 0)
+ freezer = cgroup_freezer(cgroup);
+ if (strcmp(buffer, freezer_state_strs[CGROUP_THAWED]) == 0) {
   goal_state = CGROUP_THAWED;
- else if (strcmp(buffer, freezer_state_strs[CGROUP_FROZEN]) == 0)
+  freezer->duty.ratio = 0;
+ } else if (strcmp(buffer, freezer_state_strs[CGROUP_FROZEN]) == 0) {
+  freezer->duty.ratio = 100;
   goal_state = CGROUP_FROZEN;
+ }
  else
   return -EINVAL;

+ /* we should stop duty ratio toggling if user wants to
+  * force change to a valid state.
+  */
+ freezer_check_duty_cycling(freezer, 0);
+
  if (!cgroup_lock_live_group(cgroup))
   return -ENODEV;
```

```
  retval = freezer_change_state(cgroup, goal_state);
@@ -368,12 +468,61 @@ static int freezer_write(struct cgroup *cgroup,
  return retval;
 }

+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;
+
+ spin_lock_irq(&freezer->lock);
+ switch (cft->private) {
+ case FREEZER_DUTY_RATIO:
+  if (val > 100) {
+   ret = -EINVAL;
+   goto exit;
+  }
+  freezer->duty.ratio = val;
+  break;
+ case FREEZER_PERIOD:
+  if (!val) {
+   ret = -EINVAL;
+   goto exit;
+  }
+  freezer->duty.period_pct_ms = val * 10;
+  break;
+ default:
+  BUG();
+ }
+ spin_unlock_irq(&freezer->lock);
+ freezer_check_duty_cycling(freezer, 1);
+exit:
+ cgroup_unlock();
+ return ret;
+}
+
 static struct cftype files[] = {
  {
   .name = "state",
   .read_seq_string = freezer_read,
   .write_string = freezer_write,
  },
+ {
```

```
+  .name = "frozen_time_percentage",
+  .private = FREEZER_DUTY_RATIO,
+  .read_u64 = freezer_read_duty_ratio,
+  .write_u64 = freezer_write_param,
+ },
+ {
+  .name = "period_second",
+  .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