Subject: [PATCH v3] cgroup/freezer: add per freezer duty ratio control
Posted by jacob.jun.pan on Mon, 07 Feb 2011 18:13:02 GMT
View Forum Message <> Reply to Message

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

Changelog:
v3: - added documentation

v2:
 - use read_u64/write_u64 instead of string read/write
 - use period in second instead of ms
 - renamed duty_ratio to duty_ratio_pct, be more specific
 - removed locking around period and duty_ratio reads

Jacob Pan (1):
  cgroup/freezer: add per freezer duty ratio control

 Documentation/cgroups/freezer-subsystem.txt | 23 ++++++
 kernel/cgroup_freezer.c | 109 +++++++++++++++++++++++++++-
 2 files changed, 130 insertions(+), 2 deletions(-)

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

Subject: [PATCH v3] cgroup/freezer: add per freezer duty ratio control
Posted by jacob.jun.pan on Mon, 07 Feb 2011 18:13:03 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 5 > freezer.period_sec

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

 Documentation/cgroups/freezer-subsystem.txt | 23 ++++++
 kernel/cgroup_freezer.c | 109 ++++++++++++++++++++++++++-
 2 files changed, 130 insertions(+), 2 deletions(-)

diff --git a/Documentation/cgroups/freezer-subsystem.txt
b/Documentation/cgroups/freezer-subsystem.txt
index 41f37fe..2bc1b98 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 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 5 > freezer.period_sec
+
+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..928f2ab 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 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 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 +75,35 @@ 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_pct_ms) {
+   if (try_to_freeze_cgroup(cgroup, freezer))
+    pr_info("cannot freeze\n");
+   msleep(freezer->duty.period_pct_ms *
+    freezer->duty.ratio);
+   unfreeze_cgroup(cgroup, freezer);
+   msleep(freezer->duty.period_pct_ms *
```

```
+    (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 +191,11 @@ 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 +327,16 @@ 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;
@@ -368,19 +423,69 @@ static int freezer_write(struct cgroup *cgroup,
```

```
  return retval;
 }

+static int freezer_write_duty_ratio(struct cgroup *cgroup, struct cftype *cft,
+  u64 val)
+{
+ if (!cgroup_lock_live_group(cgroup))
+  return -ENODEV;
+ cgroup_freezer(cgroup)->duty.ratio = val;
+ cgroup_unlock();
+ wake_up(&freezer_wait);
+
+ return 0;
+}
+
+static int freezer_write_period(struct cgroup *cgroup, struct cftype *cft,
+  u64 val)
+{
+ if (!cgroup_lock_live_group(cgroup))
+  return -ENODEV;
+ cgroup_freezer(cgroup)->duty.period_pct_ms = val * 10;
+ cgroup_unlock();
+ wake_up(&freezer_wait);
+
+ return 0;
+}
+
 static struct cftype files[] = {
  {
   .name = "state",
   .read_seq_string = freezer_read,
   .write_string = freezer_write,
  },
+ {
+  .name = "duty_ratio_pct",
+  .read_u64 = freezer_read_duty_ratio,
+  .write_u64 = freezer_write_duty_ratio,
+ },
+ {
+  .name = "period_sec",
+  .read_u64 = freezer_read_period,
+  .write_u64 = 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))
+  pr_debug("%s failed to create %s\n", __func__, thread_name);
+
+ 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