
Subject: Re: Unable to remove control groups on 2.6.23-rc8-mm1
Posted by [KAMEZAWA Hiroyuki](#) on Wed, 26 Sep 2007 21:37:36 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Wed, 26 Sep 2007 08:46:20 -0700
Badari Pulavarty <pbadari@us.ibm.com> wrote:

> On Wed, 2007-09-26 at 18:14 +0900, KAMEZAWA Hiroyuki wrote:

> > This is an experimental patch for drop pages in empty cgroup.

> > comments ?

>

> Hmm.. Patch doesn't seems to help :(

>

> elm3b155:/dev/cgroup/xxx # cat memory.usage_in_bytes

> 65536

> elm3b155:/dev/cgroup/xxx # sync

> elm3b155:/dev/cgroup/xxx # sync

> elm3b155:/dev/cgroup/xxx # cat memory.usage_in_bytes

> 65536

>

sorry for no explanation.

try

==

%sync

%echo -n 1 > /path_to_group/memory.drop_in_force // drop all rmaining pages.

==

Anyway I'll brush up it and post again.

Thanks,

-Kame

> Thanks,

> Badari

>

> >

> > ==

> > An experimental patch.

> >

> > Drop all pages in memcontrol cgroup if cgroup's task is empty.

> > Please type "sync" before try to drop. Unless sync, maybe -EBUSY will return.

> >

> > Problem: not handle mlocked pages now.

> >

> >

> > Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

```

>>
>> ---
>> include/linux/memcontrol.h | 13 ++++-
>> mm/memcontrol.c | 113 ++++++
>> mm/vmscan.c | 16 ++++-
>> 3 files changed, 140 insertions(+), 2 deletions(-)
>>
>> Index: linux-2.6.23-rc8-mm1/mm/memcontrol.c
>> =====
>> --- linux-2.6.23-rc8-mm1.orig/mm/memcontrol.c
>> +++ linux-2.6.23-rc8-mm1/mm/memcontrol.c
>> @@ -63,6 +63,7 @@ struct mem_cgroup {
>>  */
>>  spinlock_t lru_lock;
>>  unsigned long control_type; /* control RSS or RSS+Pagecache */
>>  + unsigned long force_drop;
>> };
>>
>> /*
>> @@ -135,6 +136,31 @@ static inline int page_cgroup_locked(str
>>  &page->page_cgroup);
>> }
>>
>> +static inline unsigned long long mem_cgroup_usage(struct mem_cgroup *mem)
>> +{
>> + return mem->res.usage;
>> +}
>> +
>> +int mem_cgroup_reclaim_end(struct mem_cgroup *mem)
>> +{
>> + if (!mem)
>> + return 0;
>> + if (mem_cgroup_usage(mem) == 0)
>> + return 1;
>> + return 0;
>> +}
>> +
>> +int mem_cgroup_force_reclaim(struct mem_cgroup *mem)
>> +{
>> + if (!mem)
>> + return 0;
>> + /* Need more precise check if LRU is separated. */
>> + if (mem->force_drop)
>> + return 1;
>> + else
>> + return 0;
>> +}
>> +

```

```

>> void page_assign_page_cgroup(struct page *page, struct page_cgroup *pc)
>> {
>>     int locked;
>>     @@ -437,6 +463,52 @@ void mem_cgroup_uncharge(struct page_cgr
>> }
>> }
>>
>> +/*
>> + * Drop all pages
>> + * # of tasks in this cgroup must be 0 before call this.
>> + */
>> +int mem_cgroup_drop(struct mem_cgroup *mem)
>> +{
>> +
>> + unsigned long long before;
>> + struct cgroup *cg = mem->css.cgroup;
>> + int ret = -EBUSY;
>> + unsigned long expire = jiffies + 30 * HZ; /* just pseudo value */
>> +
>> + css_get(&mem->css);
>> +retry:
>> + /* disallow if there is the task. */
>> + if (atomic_read(&cg->count))
>> +     goto end;
>> + /*
>> + * We have to call try_to_free_mem_cgroup_pages() several times.
>> + * Especially when there is write-back pages.
>> + */
>> + if (time_after(jiffies, expire))
>> +     goto end;
>> +
>> + before = mem_cgroup_usage(mem);
>> +
>> + if (before == 0) {
>> +     ret = 0;
>> +     goto end;
>> + }
>> + mem->force_drop = 1;
>> + if (try_to_free_mem_cgroup_pages(mem, GFP_HIGHUSER_MOVABLE) == 0)
>> +     congestion_wait(WRITE, HZ/10);
>> + mem->force_drop = 0;
>> +
>> + /* made some progress */
>> + if (mem_cgroup_usage(mem) <= before)
>> +     goto retry;
>> +
>> +end:
>> + css_put(&mem->css);

```

```

>> + return ret;
>> +}
>> +
>> +
>> +
>> int mem_cgroup_write_strategy(char *buf, unsigned long long *tmp)
>> {
>> *tmp = memparse(buf, &buf);
>> @@ -522,6 +594,41 @@ static ssize_t mem_control_type_read(str
>> ppos, buf, s - buf);
>> }
>>
>> +static ssize_t mem_drop_type_write(struct cgroup *cont,
>> + struct cftype *cft, struct file *file,
>> + const char __user *userbuf,
>> + size_t nbytes, loff_t *pos)
>> +{
>> + struct mem_cgroup *mem;
>> + int ret;
>> + char *buf, *end;
>> + unsigned long tmp;
>> +
>> + mem = mem_cgroup_from_cont(cont);
>> + buf = kmalloc(nbytes + 1, GFP_KERNEL);
>> + ret = -ENOMEM;
>> + if (buf == NULL)
>> + goto out;
>> + buf[nbytes] = 0;
>> + ret = -EFAULT;
>> + if (copy_from_user(buf, userbuf, nbytes))
>> + goto out_free;
>> + ret = -EINVAL;
>> + tmp = simple_strtoul(buf, &end, 10);
>> +
>> + if (*end != '\0')
>> + goto out_free;
>> + if (tmp) {
>> + ret = mem_cgroup_drop(mem);
>> + if (!ret)
>> + ret = nbytes;
>> + }
>> +out_free:
>> + kfree(buf);
>> +out:
>> + return ret;
>> +}
>> +
>> static struct cftype mem_cgroup_files[] = {

```

```

>> {
>> .name = "usage_in_bytes",
>> @@ -544,6 +651,11 @@ static struct cftype mem_cgroup_files[]
>> .write = mem_control_type_write,
>> .read = mem_control_type_read,
>> },
>> + {
>> + .name = "drop_in_force",
>> + .write = mem_drop_type_write,
>> + .read = mem_cgroup_read,
>> + },
>> };
>>
>> static struct mem_cgroup init_mem_cgroup;
>> @@ -567,6 +679,7 @@ mem_cgroup_create(struct cgroup_subsys *
>> INIT_LIST_HEAD(&mem->inactive_list);
>> spin_lock_init(&mem->lru_lock);
>> mem->control_type = MEM_CGROUP_TYPE_ALL;
>> + mem->force_drop = 0;
>> return &mem->css;
>> }
>>
>> Index: linux-2.6.23-rc8-mm1/include/linux/memcontrol.h
>> =====
>> --- linux-2.6.23-rc8-mm1.orig/include/linux/memcontrol.h
>> +++ linux-2.6.23-rc8-mm1/include/linux/memcontrol.h
>> @@ -47,6 +47,10 @@ extern int mem_cgroup_cache_charge(struct
>> gfp_t gfp_mask);
>> extern struct mem_cgroup *mm_cgroup(struct mm_struct *mm);
>>
>> +/* called when page reclaim has no progress in mem cgroup */
>> +extern int mem_cgroup_reclaim_end(struct mem_cgroup *mem);
>> +extern int mem_cgroup_force_reclaim(struct mem_cgroup *mem);
>> +
>> static inline void mem_cgroup_uncharge_page(struct page *page)
>> {
>> mem_cgroup_uncharge(page_get_page_cgroup(page));
>> @@ -102,7 +106,14 @@ static inline struct mem_cgroup *mm_cgrou
>> {
>> return NULL;
>> }
>> -
>> +static inline int mem_cgroup_force_reclaim(struct mem_cgroup *mem)
>> +{
>> + return 0;
>> +}
>> +static inline int mem_cgroup_reclaim_end(struct mem_cgroup *mem)
>> +{

```

```

>> + return 0;
>> +}
>> #endif /* CONFIG_CGROUP_MEM_CONT */
>>
>> #endif /* _LINUX_MEMCONTROL_H */
>> Index: linux-2.6.23-rc8-mm1/mm/vmscan.c
>> =====
>> --- linux-2.6.23-rc8-mm1.orig/mm/vmscan.c
>> +++ linux-2.6.23-rc8-mm1/mm/vmscan.c
>> @@ -1168,6 +1168,13 @@ static unsigned long shrink_zone(int pri
>>     zone->nr_scan_inactive = 0;
>>     else
>>         nr_inactive = 0;
>> + /* TODO: we need to know # of pages to be reclaimed per group */
>> + if (mem_cgroup_force_reclaim(sc->mem_cgroup)) {
>> +     if (!nr_active)
>> +         nr_active = sc->swap_cluster_max;
>> +     if (!nr_inactive)
>> +         nr_inactive = sc->swap_cluster_max;
>> + }
>>
>>     while (nr_active || nr_inactive) {
>>         if (nr_active) {
>> @@ -1256,6 +1263,7 @@ static unsigned long do_try_to_free_page
>>         int ret = 0;
>>         unsigned long total_scanned = 0;
>>         unsigned long nr_reclaimed = 0;
>> + unsigned long progress;
>>         struct reclaim_state *reclaim_state = current->reclaim_state;
>>         unsigned long lru_pages = 0;
>>         int i;
>> @@ -1276,7 +1284,8 @@ static unsigned long do_try_to_free_page
>>         sc->nr_scanned = 0;
>>         if (!priority)
>>             disable_swap_token();
>> - nr_reclaimed += shrink_zones(priority, zones, sc);
>> + progress = shrink_zones(priority, zones, sc);
>> + nr_reclaimed += progress;
>>         /*
>>          * Don't shrink slabs when reclaiming memory from
>>          * over limit cgroups
>> @@ -1292,6 +1301,11 @@ static unsigned long do_try_to_free_page
>>         ret = 1;
>>         goto out;
>>     }
>> + if (progress == 0 &&
>> +     mem_cgroup_reclaim_end(sc->mem_cgroup)) {
>> +     ret = 1;

```

```
> > + goto out;
> > + }
> >
> > /*
> >  * Try to write back as many pages as we just scanned. This
> >
>
>
```

Containers mailing list
Containers@lists.linux-foundation.org
<https://lists.linux-foundation.org/mailman/listinfo/containers>
