

---

Subject: [PATCH] memory cgroup enhancements take 4 [0/8] intro  
Posted by [KAMEZAWA Hiroyuki](#) on Wed, 31 Oct 2007 10:22:13 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Hi, this set is for enhancements for memory cgroup I have now.  
Tested on x86\_64 and passed some tests.  
All are against 2.6.23-mm1 + previous memory cgroup bugfix patches.

Any comments are welcome.

Patch contents:

[1/8] .... fix zone handling in try\_to\_free\_mem\_cgroup\_page  
This is bug fix.

[2/8] .... force\_empty interface for dropping all account in empty cgroup  
enhancements for easy deleting empty cgroup which was used  
for memory control. Without this, deleting will fail  
in many case.

[3/8] .... remember "a page is charged as page cache"  
record as what a page is charged.

[4/8] .... remember "a page is on active list of cgroup or not"  
for future use. (can be skipped.)  
will be useful for reclaim routine enhance

[5/8] .... add status accounting function for memory cgroup  
infrastructure for accounting.  
will be used in memory.stat file

[6/8] .... add memory.stat file  
showing # of RSS and CACHEes by memory.stat file  
and other \*memory specific\* data in future.

[7/8] .... pre destroy handler  
add cgroup pre\_destroy handler before calling destroy handler.

[8/8] .... implicit force\_empty at rmdir()  
call force\_empty in pre\_destroy handler.  
This allows rmdir() to success always if cgroup is empty.

Reflected all comments against take3 and dropped zonestat.

Thanks,  
-Kame

---

---

Subject: [PATCH] memory cgroup enhancements take 4 [2/8] force\_empty interface for dropping all account in em

Posted by [KAMEZAWA Hiroyuki](#) on Wed, 31 Oct 2007 10:25:18 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

This patch adds an interface "memory.force\_empty".  
Any write to this file will drop all charges in this cgroup if there is no task under.

```
%echo 1 > /...../memory.force_empty
```

will drop all charges of memory cgroup if cgroup's tasks is empty.

This is useful to invoke rmdir() against memory cgroup successfully.

Tested and worked well on x86\_64/fake-NUMA system.

Changelog v4 -> v5:

- added comments to mem\_cgroup\_force\_empty()
- made mem\_force\_empty\_read return -EINVAL
- cleanup mem\_cgroup\_force\_empty\_list()
- removed SWAP\_CLUSTER\_MAX

Changelog v3 -> v4:

- adjusted to 2.6.23-mm1
- fixed typo
- changes buf[2]="0" to static const

Changelog v2 -> v3:

- changed the name from force\_reclaim to force\_empty.

Changelog v1 -> v2:

- added a new interface force\_reclaim.
- changes spin\_lock to spin\_lock\_irqsave().

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

```
mm/memcontrol.c | 110 ++++++-----
1 file changed, 103 insertions(+), 7 deletions(-)
```

Index: devel-2.6.23-mm1/mm/memcontrol.c

```

=====
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -480,6 +480,7 @@ void mem_cgroup_uncharge(struct page_cgr
    page = pc->page;
    /*
     * get page->cgroup and clear it under lock.
+   * force_empty can drop page->cgroup without checking refcnt.
     */
    if (clear_page_cgroup(page, pc) == pc) {
        mem = pc->mem_cgroup;
@@ -489,13 +490,6 @@ void mem_cgroup_uncharge(struct page_cgr
    list_del_init(&pc->lru);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
    kfree(pc);
- } else {
-     /*
-     * Note: This will be removed when force-empty patch is
-     * applied. just show warning here.
-     */
-     printk(KERN_ERR "Race in mem_cgroup_uncharge() ?");
-     dump_stack();
- }
- }
- }
@@ -543,6 +537,76 @@ retry:
    return;
}

+/*
+ * This routine traverse page_cgroup in given list and drop them all.
+ * This routine ignores page_cgroup->ref_cnt.
+ * *And* this routine doesn't reclaim page itself, just removes page_cgroup.
+ */
+#define FORCE_UNCHARGE_BATCH (128)
+static void
+mem_cgroup_force_empty_list(struct mem_cgroup *mem, struct list_head *list)
+{
+    struct page_cgroup *pc;
+    struct page *page;
+    int count;
+    unsigned long flags;
+
+    +
+    +retry:
+    + count = FORCE_UNCHARGE_BATCH;
+    + spin_lock_irqsave(&mem->lru_lock, flags);
+    +
+    + while (--count && !list_empty(list)) {

```

```

+ pc = list_entry(list->prev, struct page_cgroup, lru);
+ page = pc->page;
+ /* Avoid race with charge */
+ atomic_set(&pc->ref_cnt, 0);
+ if (clear_page_cgroup(page, pc) == pc) {
+   css_put(&mem->css);
+   res_counter_uncharge(&mem->res, PAGE_SIZE);
+   list_del_init(&pc->lru);
+   kfree(pc);
+ } else /* being uncharged ? ...do relax */
+   break;
+ }
+ spin_unlock_irqrestore(&mem->lru_lock, flags);
+ if (!list_empty(list)) {
+   cond_resched();
+   goto retry;
+ }
+ return;
+}
+
+/*
+ * make mem_cgroup's charge to be 0 if there is no task.
+ * This enables deleting this mem_cgroup.
+ */
+
+int mem_cgroup_force_empty(struct mem_cgroup *mem)
+{
+ int ret = -EBUSY;
+ css_get(&mem->css);
+ /*
+  * page reclaim code (kswapd etc..) will move pages between
+  * active_list <-> inactive_list while we don't take a lock.
+  * So, we have to do loop here until all lists are empty.
+  */
+ while (!(list_empty(&mem->active_list) &&
+   list_empty(&mem->inactive_list))) {
+   if (atomic_read(&mem->css.cgroup->count) > 0)
+     goto out;
+   /* drop all page_cgroup in active_list */
+   mem_cgroup_force_empty_list(mem, &mem->active_list);
+   /* drop all page_cgroup in inactive_list */
+   mem_cgroup_force_empty_list(mem, &mem->inactive_list);
+ }
+ ret = 0;
+out:
+ css_put(&mem->css);
+ return ret;
+}

```

```

+
+
+
int mem_cgroup_write_strategy(char *buf, unsigned long long *tmp)
{
    *tmp = memparse(buf, &buf);
@@ -628,6 +692,33 @@ static ssize_t mem_control_type_read(str
    ppos, buf, s - buf);
}

+
+static ssize_t mem_force_empty_write(struct cgroup *cont,
+ struct cftype *cft, struct file *file,
+ const char __user *userbuf,
+ size_t nbytes, loff_t *ppos)
+{
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
+ int ret;
+ ret = mem_cgroup_force_empty(mem);
+ if (!ret)
+ ret = nbytes;
+ return ret;
+}
+
+/*
+ * Note: This should be removed if cgroup supports write-only file.
+ */
+
+static ssize_t mem_force_empty_read(struct cgroup *cont,
+ struct cftype *cft,
+ struct file *file, char __user *userbuf,
+ size_t nbytes, loff_t *ppos)
+{
+ return -EINVAL;
+}
+
+
+static struct cftype mem_cgroup_files[] = {
+ {
+     .name = "usage_in_bytes",
@@ -650,6 +741,11 @@ static struct cftype mem_cgroup_files[]
+     .write = mem_control_type_write,
+     .read = mem_control_type_read,
+ },
+ {
+     .name = "force_empty",
+     .write = mem_force_empty_write,
+     .read = mem_force_empty_read,

```

```
+ },  
};
```

```
static struct mem_cgroup init_mem_cgroup;
```

---

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

---

---

Subject: [PATCH] memory cgroup enhancements take 4 [3/8] remember "a page is charged as page cache"

Posted by [KAMEZAWA Hiroyuki](#) on Wed, 31 Oct 2007 10:28:48 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Add a flag to page\_cgroup to remember "this page is charged as cache."  
cache here includes page caches and swap cache.  
This is useful for implementing precise accounting in memory cgroup.  
TODO:  
    distinguish page-cache and swap-cache

Changelog v2 -> v3

- added enum for mem\_cgroup\_charge\_type\_common(...charge\_type)
- renamed #define PCGF\_XXX\_XXX to PAGE\_CGROUP\_FLAG\_XXX

Changelog v1 -> v2

- moved #define to out-side of struct definition

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

Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

mm/memcontrol.c | 24 ++++++-----  
1 file changed, 21 insertions(+), 3 deletions(-)

Index: devel-2.6.23-mm1/mm/memcontrol.c

```
=====
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -83,7 +83,9 @@ struct page_cgroup {
    struct mem_cgroup *mem_cgroup;
    atomic_t ref_cnt; /* Helpful when pages move b/w */
    /* mapped and cached states */
+ int flags;
};
+#define PAGE_CGROUP_FLAG_CACHE (0x1) /* charged as cache */
```

```

enum {
    MEM_CGROUP_TYPE_UNSPEC = 0,
@@ -93,6 +95,11 @@ enum {
    MEM_CGROUP_TYPE_MAX,
};

+enum charge_type {
+ MEM_CGROUP_CHARGE_TYPE_CACHE = 0,
+ MEM_CGROUP_CHARGE_TYPE_MAPPED,
+};
+
static struct mem_cgroup init_mem_cgroup;

static inline
@@ -315,8 +322,8 @@ unsigned long mem_cgroup_isolate_pages(u
    * 0 if the charge was successful
    * < 0 if the cgroup is over its limit
    */
-int mem_cgroup_charge(struct page *page, struct mm_struct *mm,
-    gfp_t gfp_mask)
+static int mem_cgroup_charge_common(struct page *page, struct mm_struct *mm,
+    gfp_t gfp_mask, enum charge_type ctype)
{
    struct mem_cgroup *mem;
    struct page_cgroup *pc;
@@ -418,6 +425,9 @@ noreclaim:
    atomic_set(&pc->ref_cnt, 1);
    pc->mem_cgroup = mem;
    pc->page = page;
+ pc->flags = 0;
+ if (ctype == MEM_CGROUP_CHARGE_TYPE_CACHE)
+ pc->flags |= PAGE_CGROUP_FLAG_CACHE;
    if (page_cgroup_assign_new_page_cgroup(page, pc)) {
/*
    * an another charge is added to this page already.
@@ -442,6 +452,13 @@ err:
    return -ENOMEM;
}

+int mem_cgroup_charge(struct page *page, struct mm_struct *mm,
+    gfp_t gfp_mask)
+{
+ return mem_cgroup_charge_common(page, mm, gfp_mask,
+    MEM_CGROUP_CHARGE_TYPE_MAPPED);
+}
+
/*
    * See if the cached pages should be charged at all?

```

```

*/
@@ -454,7 +471,8 @@ int mem_cgroup_cache_charge(struct page

    mem = rcu_dereference(mm->mem_cgroup);
    if (mem->control_type == MEM_CGROUP_TYPE_ALL)
-   return mem_cgroup_charge(page, mm, gfp_mask);
+   return mem_cgroup_charge_common(page, mm, gfp_mask,
+   MEM_CGROUP_CHARGE_TYPE_CACHE);
    else
        return 0;
}

```

---

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

---



---

Subject: [PATCH] memory cgroup enhancements take 4 [4/8] remember "a page is on active list of cgroup or  
Posted by [KAMEZAWA Hiroyuki](#) on Wed, 31 Oct 2007 10:29:26 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Remember page\_cgroup is on active\_list or not in page\_cgroup->flags.

Against 2.6.23-mm1.

Changelog v2->v3

- renamed #define PCGF\_ACTIVE to PAGE\_CGROUP\_FLAG\_ACTIVE.

Changelog v1->v2

- moved #define to out-side of struct definition

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

Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

mm/memcontrol.c | 10 ++++++---  
1 file changed, 7 insertions(+), 3 deletions(-)

Index: devel-2.6.23-mm1/mm/memcontrol.c

```

=====
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -86,6 +86,7 @@ struct page_cgroup {
    int flags;
};
#define PAGE_CGROUP_FLAG_CACHE (0x1) /* charged as cache */
+#define PAGE_CGROUP_FLAG_ACTIVE (0x2) /* page is active in this cgroup */

```

```

enum {
    MEM_CGROUP_TYPE_UNSPEC = 0,
@@ -213,10 +214,13 @@ clear_page_cgroup(struct page *page, str

static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
{
- if (active)
+ if (active) {
+ pc->flags |= PAGE_CGROUP_FLAG_ACTIVE;
+ list_move(&pc->lru, &pc->mem_cgroup->active_list);
- else
+ } else {
+ pc->flags &= ~PAGE_CGROUP_FLAG_ACTIVE;
+ list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
+ }
}

int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem)
@@ -425,7 +429,7 @@ noreclaim:
    atomic_set(&pc->ref_cnt, 1);
    pc->mem_cgroup = mem;
    pc->page = page;
- pc->flags = 0;
+ pc->flags = PAGE_CGROUP_FLAG_ACTIVE;
    if (ctype == MEM_CGROUP_CHARGE_TYPE_CACHE)
        pc->flags |= PAGE_CGROUP_FLAG_CACHE;
    if (page_cgroup_assign_new_page_cgroup(page, pc)) {

```

---

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

---



---

Subject: [PATCH] memory cgroup enhancements take 4 [5/8] add status accounting function for memory cgroup

Posted by [KAMEZAWA Hiroyuki](#) on Wed, 31 Oct 2007 10:30:46 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Add statistics account infrastructure for memory controller.  
All account information is stored per-cpu and caller will not have to take lock or use atomic ops.  
This will be used by memory.stat file later.

CACHE includes swapcache now. I'd like to divide it to PAGECACHE and SWAPCACHE later.

This patch adds 3 functions for accounting.

- \* \_\_mem\_cgroup\_stat\_add() ... for usual routine.
- \* \_\_mem\_cgroup\_stat\_add\_safe ... for calling under irq\_disabled section.
- \* mem\_cgroup\_read\_stat() ... for reading stat value.
- \* renamed PAGECACHE to CACHE (because it may include swapcache \*now\*)

#### Changelog v3 -> v4

- fixed typo.
- removed unused inc/dec interface.
- added \_\_mem\_cgroup\_stat\_add\_safe() for accounting under safe situation.
- moved callers of mem\_cgroup\_charge\_statistics() under irq disabled section.
- added mem\_cgroup\_read\_stat()

#### Changelog v2 -> v3

- adjusted to rename of #define PAGE\_CGROUP\_FLAG....
- dropped ACTIVE/INACTIVE counter.  
They should be accounted against per zone. Then, using pcp counter, we need array of NR\_CPU \* MAX\_NUMNODES \* NR\_ZONES against all stats. This is too big for statistics \*per-memory-cgroup\*.  
ACTIVE/INACTIVE counter is added as per-zone statistics later.

#### Changelog v1 -> v2

- Removed Charge/Uncharge counter
- reflected comments.
- changes \_\_move\_lists() args.
- changes \_\_mem\_cgroup\_stat\_add() name, comment and added VM\_BUGON

#### Changes from original:

- divided into 2 patch (account and show info)
- changed from u64 to s64
- added mem\_cgroup\_stat\_add() and batched statistics modification logic.
- removed stat init code because mem\_cgroup is allocated by kzalloc().

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

Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

mm/memcontrol.c | 82

+++++  
1 file changed, 82 insertions(+)

Index: devel-2.6.23-mm1/mm/memcontrol.c

=====

--- devel-2.6.23-mm1.orig/mm/memcontrol.c

+++ devel-2.6.23-mm1/mm/memcontrol.c

@@ -35,6 +35,59 @@ struct cgroup\_subsys mem\_cgroup\_subsys;  
static const int MEM\_CGROUP\_RECLAIM\_RETRIES = 5;

/\*

```

+ * Statistics for memory cgroup.
+ */
+enum mem_cgroup_stat_index {
+ /*
+ * For MEM_CONTAINER_TYPE_ALL, usage = pagecache + rss.
+ */
+ MEM_CGROUP_STAT_CACHE, /* # of pages charged as cache */
+ MEM_CGROUP_STAT_RSS, /* # of pages charged as rss */
+
+ MEM_CGROUP_STAT_NSTATS,
+};
+
+struct mem_cgroup_stat_cpu {
+ s64 count[MEM_CGROUP_STAT_NSTATS];
+} ____cacheline_aligned_in_smp;
+
+struct mem_cgroup_stat {
+ struct mem_cgroup_stat_cpu cpustat[NR_CPUS];
+};
+
+/*
+ * modifies value with disabling preempt.
+ */
+static inline void __mem_cgroup_stat_add(struct mem_cgroup_stat *stat,
+      enum mem_cgroup_stat_index idx, int val)
+{
+ int cpu = smp_processor_id();
+ preempt_disable();
+ stat->cpustat[cpu].count[idx] += val;
+ preempt_enable();
+}
+
+/*
+ * For accounting under irq disable, no need for increment preempt count.
+ */
+static inline void __mem_cgroup_stat_add_safe(struct mem_cgroup_stat *stat,
+      enum mem_cgroup_stat_index idx, int val)
+{
+ int cpu = smp_processor_id();
+ stat->cpustat[cpu].count[idx] += val;
+}
+
+static inline s64 mem_cgroup_read_stat(struct mem_cgroup_stat *stat,
+      enum mem_cgroup_stat_index idx)
+{
+ int cpu;
+ s64 ret = 0;
+ for_each_possible_cpu(cpu)

```

```

+ ret += stat->cpustat[cpu].count[idx];
+ return ret;
+}
+
+/*
+ * The memory controller data structure. The memory controller controls both
+ * page cache and RSS per cgroup. We would eventually like to provide
+ * statistics based on the statistics developed by Rik Van Riel for clock-pro,
@@ -63,6 +116,10 @@ struct mem_cgroup {
+ */
+ spinlock_t lru_lock;
+ unsigned long control_type; /* control RSS or RSS+Pagecache */
+ /*
+ * statistics.
+ */
+ struct mem_cgroup_stat stat;
+ };

+ /*
@@ -101,6 +158,27 @@ enum charge_type {
+ MEM_CGROUP_CHARGE_TYPE_MAPPED,
+ };

+ /*
+ * Always modified under lru lock. Then, not necessary to preempt_disable()
+ */
+ static inline void
+ mem_cgroup_charge_statistics(struct mem_cgroup *mem, int flags, bool charge)
+ {
+ int val = (charge)? 1 : -1;
+ struct mem_cgroup_stat *stat = &mem->stat;
+ VM_BUG_ON(!irqs_disabled());
+
+ if (flags & PAGE_CGROUP_FLAG_CACHE)
+ __mem_cgroup_stat_add_safe(stat,
+ MEM_CGROUP_STAT_CACHE, val);
+ else
+ __mem_cgroup_stat_add_safe(stat, MEM_CGROUP_STAT_RSS, val);
+
+ }
+
+
+
+
+ static struct mem_cgroup init_mem_cgroup;

+ static inline
@@ -445,6 +523,8 @@ noreclaim:

```

```

}

spin_lock_irqsave(&mem->lru_lock, flags);
+ /* Update statistics vector */
+ mem_cgroup_charge_statistics(mem, pc->flags, true);
  list_add(&pc->lru, &mem->active_list);
  spin_unlock_irqrestore(&mem->lru_lock, flags);

@@ -510,6 +590,7 @@ void mem_cgroup_uncharge(struct page_cgr
  res_counter_uncharge(&mem->res, PAGE_SIZE);
  spin_lock_irqsave(&mem->lru_lock, flags);
  list_del_init(&pc->lru);
+ mem_cgroup_charge_statistics(mem, pc->flags, false);
  spin_unlock_irqrestore(&mem->lru_lock, flags);
  kfree(pc);
}
@@ -586,6 +667,7 @@ retry:
  css_put(&mem->css);
  res_counter_uncharge(&mem->res, PAGE_SIZE);
  list_del_init(&pc->lru);
+ mem_cgroup_charge_statistics(mem, pc->flags, false);
  kfree(pc);
} else /* being uncharged ? ...do relax */

```

---

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

---



---

Subject: [PATCH] memory cgroup enhancements take 4 [6/8] add memory.stat file  
Posted by [KAMEZAWA Hiroyuki](#) on Wed, 31 Oct 2007 10:31:36 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Show accounted information of memory cgroup by memory.stat file

Changelog v2->v3  
- make use of mem\_cgroup\_read\_stat() at printing.  
Changelog v1->v2  
- dropped Charge/Uncharge entry.

Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>  
Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

mm/memcontrol.c | 48 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++  
1 file changed, 48 insertions(+)

Index: devel-2.6.23-mm1/mm/memcontrol.c

```

=====
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -28,6 +28,7 @@
#include <linux/swap.h>
#include <linux/spinlock.h>
#include <linux/fs.h>
+#include <linux/seq_file.h>

#include <asm/uaccess.h>

@@ -823,6 +824,48 @@ static ssize_t mem_force_empty_read(stru
}

+static const struct mem_cgroup_stat_desc {
+ const char *msg;
+ u64 unit;
+} mem_cgroup_stat_desc[] = {
+ [MEM_CGROUP_STAT_CACHE] = { "cache", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_RSS] = { "rss", PAGE_SIZE, },
+};
+
+static int mem_control_stat_show(struct seq_file *m, void *arg)
+{
+ struct cgroup *cont = m->private;
+ struct mem_cgroup *mem_cont = mem_cgroup_from_cont(cont);
+ struct mem_cgroup_stat *stat = &mem_cont->stat;
+ int i;
+
+ for (i = 0; i < ARRAY_SIZE(stat->cpustat[0].count); i++) {
+ s64 val;
+
+ val = mem_cgroup_read_stat(stat, i);
+ val *= mem_cgroup_stat_desc[i].unit;
+ seq_printf(m, "%s %lld\n", mem_cgroup_stat_desc[i].msg, val);
+ }
+ return 0;
+}
+
+static const struct file_operations mem_control_stat_file_operations = {
+ .read = seq_read,
+ .llseek = seq_lseek,
+ .release = single_release,
+};
+
+static int mem_control_stat_open(struct inode *unused, struct file *file)
+{

```

```

+ /* XXX __d_cont */
+ struct cgroup *cont = file->f_dentry->d_parent->d_fsdata;
+
+ file->f_op = &mem_control_stat_file_operations;
+ return single_open(file, mem_control_stat_show, cont);
+}
+
+
+
static struct cftype mem_cgroup_files[] = {
{
.name = "usage_in_bytes",
@@ -850,6 +893,10 @@ static struct cftype mem_cgroup_files[]
.write = mem_force_empty_write,
.read = mem_force_empty_read,
},
+ {
+ .name = "stat",
+ .open = mem_control_stat_open,
+ },
};

static struct mem_cgroup init_mem_cgroup;

```

---

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

---



---

Subject: [PATCH] memory cgroup enhancements take 4 [7/8] add pre destroy handler

Posted by [KAMEZAWA Hiroyuki](#) on Wed, 31 Oct 2007 10:32:07 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

My main purpose of this patch is for memory controller..

This patch adds a handler "pre\_destroy" to cgroup\_subsys.  
It is called before cgroup\_rmdir() checks all subsys's refcnt.

I think this is useful for subsys which have some extra refs even if there are no tasks in cgroup. By adding pre\_destroy(), the kernel keeps the rule "destroy() against subsystem is called only when refcnt=0." and allows css ref to be used by other objects than tasks.

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

include/linux/cgroup.h | 1 +  
kernel/cgroup.c | 7 +++++++  
2 files changed, 8 insertions(+)

Index: devel-2.6.23-mm1/include/linux/cgroup.h

```
=====
--- devel-2.6.23-mm1.orig/include/linux/cgroup.h
+++ devel-2.6.23-mm1/include/linux/cgroup.h
@@ -233,6 +233,7 @@ int cgroup_is_descendant(const struct cg
struct cgroup_subsys {
    struct cgroup_subsys_state *(*create)(struct cgroup_subsys *ss,
        struct cgroup *cont);
+ void (*pre_destroy)(struct cgroup_subsys *ss, struct cgroup *cont);
    void (*destroy)(struct cgroup_subsys *ss, struct cgroup *cont);
    int (*can_attach)(struct cgroup_subsys *ss,
        struct cgroup *cont, struct task_struct *tsk);
```

Index: devel-2.6.23-mm1/kernel/cgroup.c

```
=====
--- devel-2.6.23-mm1.orig/kernel/cgroup.c
+++ devel-2.6.23-mm1/kernel/cgroup.c
@@ -2158,6 +2158,13 @@ static int cgroup_rmdir(struct inode *un
    parent = cont->parent;
    root = cont->root;
    sb = root->sb;
+ /*
+  * Notify subsyses that rmdir() request comes.
+  */
+ for_each_subsys(root, ss) {
+ if ((cont->subsys[ss->subsys_id]) && ss->pre_destroy)
+ ss->pre_destroy(ss, cont);
+ }

    if (cgroup_has_css_refs(cont)) {
        mutex_unlock(&cgroup_mutex);
```

---

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

---

---

Subject: [PATCH] memory cgroup enhancements take 4 [8/8] implicit force empty at rmdir()  
Posted by [KAMEZAWA Hiroyuki](#) on Wed, 31 Oct 2007 10:32:56 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

This patch adds pre\_destroy handler for mem\_cgroup and try to make mem\_cgroup empty at rmdir().

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

mm/memcontrol.c | 8 +++++++  
1 file changed, 8 insertions(+)

Index: devel-2.6.23-mm1/mm/memcontrol.c

```
=====
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -923,6 +923,13 @@ mem_cgroup_create(struct cgroup_subsys *
    return &mem->css;
}

+static void mem_cgroup_pre_destroy(struct cgroup_subsys *ss,
+    struct cgroup *cont)
+{
+    struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
+    mem_cgroup_force_empty(mem);
+}
+
static void mem_cgroup_destroy(struct cgroup_subsys *ss,
    struct cgroup *cont)
{
@@ -974,6 +981,7 @@ struct cgroup_subsys mem_cgroup_subsys =
    .name = "memory",
    .subsys_id = mem_cgroup_subsys_id,
    .create = mem_cgroup_create,
+    .pre_destroy = mem_cgroup_pre_destroy,
    .destroy = mem_cgroup_destroy,
    .populate = mem_cgroup_populate,
    .attach = mem_cgroup_move_task,
```

---

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

---

---

Subject: Re: [PATCH] memory cgroup enhancements take 4 [5/8] add status  
accounting function for memory cgroup  
Posted by [akpm](#) on Wed, 31 Oct 2007 22:12:34 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

On Wed, 31 Oct 2007 19:30:46 +0900  
KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:

> Add statistics account infrastructure for memory controller.

```

> All account information is stored per-cpu and caller will not have
> to take lock or use atomic ops.
> This will be used by memory.stat file later.
>
> CACHE includes swapcache now. I'd like to divide it to
> PAGECACHE and SWAPCACHE later.
>
> ...
>
> --- devel-2.6.23-mm1.orig/mm/memcontrol.c
> +++ devel-2.6.23-mm1/mm/memcontrol.c
> @@ -35,6 +35,59 @@ struct cgroup_subsys mem_cgroup_subsys;
> static const int MEM_CGROUP_RECLAIM_RETRIES = 5;
>
> /*
>  * Statistics for memory cgroup.
>  */
> enum mem_cgroup_stat_index {
>  /*
>   * For MEM_CONTAINER_TYPE_ALL, usage = pagecache + rss.
>   */
>  MEM_CGROUP_STAT_CACHE, /* # of pages charged as cache */
>  MEM_CGROUP_STAT_RSS, /* # of pages charged as rss */
>  +
>  MEM_CGROUP_STAT_NSTATS,
> };
> +
> struct mem_cgroup_stat_cpu {
>  s64 count[MEM_CGROUP_STAT_NSTATS];
> } ____cacheline_aligned_in_smp;
> +
> struct mem_cgroup_stat {
>  struct mem_cgroup_stat_cpu cpustat[NR_CPUS];
> };
> +
> /*
>  * modifies value with disabling preempt.
>  */
> static inline void __mem_cgroup_stat_add(struct mem_cgroup_stat *stat,
>  enum mem_cgroup_stat_index idx, int val)
> {
>  int cpu = smp_processor_id();
>  preempt_disable();
>  stat->cpustat[cpu].count[idx] += val;
>  preempt_enable();
> }

```

This is clearly doing `smp_processor_id()` in preemptible code. (or the

preempt\_disable() just isn't needed). I fixed it up.

Please ensure that you test with all runtime debugging options enabled - you should have seen a warning here.

```
> +/*
> + * For accounting under irq disable, no need for increment preempt count.
> + */
> +static inline void __mem_cgroup_stat_add_safe(struct mem_cgroup_stat *stat,
> + enum mem_cgroup_stat_index idx, int val)
> +{
> + int cpu = smp_processor_id();
> + stat->cpustat[cpu].count[idx] += val;
> +}
```

There's a wild amount of inlining in that file. Please, just don't do it - inline is a highly specialised thing and is rarely needed.

When I removed the obviously-wrong inline statements, the size of mm/memcontrol.o went from 3823 bytes down to 3495.

It also caused this:

mm/memcontrol.c:65: warning: '\_\_mem\_cgroup\_stat\_add' defined but not used

so I guess I'll just remove that.

---

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

---

---

Subject: Re: [PATCH] memory cgroup enhancements take 4 [5/8] add status accounting function for memory cgroup  
Posted by [KAMEZAWA Hiroyuki](#) on Thu, 01 Nov 2007 00:29:56 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

At first, thank you for review.

On Wed, 31 Oct 2007 15:12:34 -0700  
Andrew Morton <akpm@linux-foundation.org> wrote:  
> > +static inline void \_\_mem\_cgroup\_stat\_add(struct mem\_cgroup\_stat \*stat,  
> > + enum mem\_cgroup\_stat\_index idx, int val)  
> > +{  
> > + int cpu = smp\_processor\_id();  
> > + preempt\_disable();  
> > + stat->cpustat[cpu].count[idx] += val;

> > + preempt\_enable();  
> > +}  
>  
> This is clearly doing smp\_processor\_id() in preemptible code. (or the  
> preempt\_disable() just isn't needed). I fixed it up.  
>  
Thanks,  
> Please ensure that you test with all runtime debugging options enabled -  
> you should have seen a warning here.  
>  
Sorry, I'll take care.

> > +/\*  
> > + \* For accounting under irq disable, no need for increment preempt count.  
> > + \*/  
> > +static inline void \_\_mem\_cgroup\_stat\_add\_safe(struct mem\_cgroup\_stat \*stat,  
> > + enum mem\_cgroup\_stat\_index idx, int val)  
> > +{  
> > + int cpu = smp\_processor\_id();  
> > + stat->cpustat[cpu].count[idx] += val;  
> > +}  
>  
> There's a wild amount of inlining in that file. Please, just don't do it -  
> inline is a highly specialised thing and is rarely needed.  
>  
> When I removed the obviously-wrong inline statements, the size of  
> mm/memcontrol.o went from 3823 bytes down to 3495.  
>  
> It also caused this:  
>  
> mm/memcontrol.c:65: warning: '\_\_mem\_cgroup\_stat\_add' defined but not used  
>  
> so I guess I'll just remove that.  
>  
ok. I'll add again if it is needed again.

Thanks,  
-Kame

---

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

---