Subject: [PATCH] memory cgroup enhancements [0/5] intro
Posted by KAMEZAWA Hiroyuki on Tue, 16 Oct 2007 10:19:49 GMT
View Forum Message <> Reply to Message

This patch set adds
 - force_empty interface, which drops all charges in memory cgroup.
   This enables rmdir() against unused memory cgroup.
 - the memory cgroup statistics accounting.

Based on 2.6.23-mm1 + http://lkml.org/lkml/2007/10/12/53

Changes from previous version is
 - merged comments.
 - based on 2.6.23-mm1
 - removed Charge/Uncharge counter.

[1/5] ... force_empty patch
[2/5] ... remember page is charged as page-cache patch
[3/5] ... remember page is on which list patch
[4/5] ... memory cgroup statistics patch
[5/5] ... show statistics patch

tested on x86-64/fake-NUMA + CONFIG_PREEMPT=y/n (for testing preempt_disable())

Any comments are welcome.

-Kame

Subject: [PATCH] memory cgroup enhancements [1/5] force_empty for memory cgroup
Posted by KAMEZAWA Hiroyuki on Tue, 16 Oct 2007 10:23:41 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 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 |  102 +++++++++++++++++++++++++++++++++++++++++++++++++++++----
 1 file changed, 95 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 +537,70 @@ 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 relcaim page itself, just removes page_cgroup.
+ */
+static void
+mem_cgroup_force_empty_list(struct mem_cgroup *mem, struct list_head *list)
+{
+ struct page_cgroup *pc;
+ struct page *page;
+ int count = SWAP_CLUSTER_MAX;
+ unsigned long flags;
+
+ spin_lock_irqsave(&mem->lru_lock, flags);
+
+ while (!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
+   count = 1; /* being uncharged ? ...do relax */
+
+  if (--count == 0) {
+   spin_unlock_irqrestore(&mem->lru_lock, flags);
+   cond_resched();
+   spin_lock_irqsave(&mem->lru_lock, flags);
+   count = SWAP_CLUSTER_MAX;
+  }
+ }
+ spin_unlock_irqrestore(&mem->lru_lock, flags);
+}
+
+/*
+ * make mem_cgroup's charge to be 0 if there is no binded task.
+ * This enables deleting this mem_cgroup.
+ */
+
+int mem_cgroup_force_empty(struct mem_cgroup *mem)
+{
+ int ret = -EBUSY;
+ css_get(&mem->css);
+ 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 +686,31 @@ 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;
+}
+
+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)
+{
+ static const char buf[2] = "0";
+ return simple_read_from_buffer((void __user *)userbuf, nbytes,
+   ppos, buf, strlen(buf));
+}
```

```
+
+
 static struct cftype mem_cgroup_files[] = {
  {
   .name = "usage_in_bytes",
@@ -650,6 +733,11 @@ 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 [2/5] remember charge as cache
Posted by KAMEZAWA Hiroyuki on Tue, 16 Oct 2007 10:25:10 GMT
View Forum Message <> Reply to Message

Add PCGF_PAGECACHE flag to page_cgroup to remember "this page is
charged as page-cache."
This is very useful for implementing precise accounting in memory cgroup.

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

```
 mm/memcontrol.c |   18 +++++++++++++++---
 1 file changed, 15 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
@@ -83,6 +83,8 @@ 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 PCGF_PAGECACHE  (0x1) /* charged as page-cache */
 };

 enum {
@@ -315,8 +317,8 @@ 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, int is_cache)
 {
  struct mem_cgroup *mem;
  struct page_cgroup *pc;
@@ -418,6 +420,10 @@ noreclaim:
  atomic_set(&pc->ref_cnt, 1);
  pc->mem_cgroup = mem;
  pc->page = page;
+ if (is_cache)
+  pc->flags = PCGF_PAGECACHE;
+ else
+  pc->flags = 0;
  if (page_cgroup_assign_new_page_cgroup(page, pc)) {
   /*
    * an another charge is added to this page already.
@@ -442,6 +448,12 @@ 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, 0);
+}
+
 /*
  * See if the cached pages should be charged at all?
  */
@@ -454,7 +466,7 @@ 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, 1);
  else
   return 0;
 }
```

Subject: [PATCH] memory cgroup enhancements [3/5] record pc is on active list
Posted by KAMEZAWA Hiroyuki on Tue, 16 Oct 2007 10:26:13 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.

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

```
 mm/memcontrol.c |   12 ++++++++----
 1 file changed, 8 insertions(+), 4 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
@@ -85,6 +85,7 @@ struct page_cgroup {
      /* mapped and cached states     */
  int  flags;
 #define PCGF_PAGECACHE  (0x1) /* charged as page-cache */
+#define PCGF_ACTIVE  (0x2) /* this is on cgroup's active list */
 };

 enum {
@@ -208,10 +209,13 @@ 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 |= PCGF_ACTIVE;
   list_move(&pc->lru, &pc->mem_cgroup->active_list);
- else
+ } else {
+  pc->flags &= ~PCGF_ACTIVE;
   list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
+ }
 }
 int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem)
@@ -421,9 +425,9 @@ noreclaim:
  pc->mem_cgroup = mem;
  pc->page = page;
  if (is_cache)
-  pc->flags = PCGF_PAGECACHE;
+  pc->flags = PCGF_PAGECACHE | PCGF_ACTIVE;
  else
-  pc->flags = 0;
+  pc->flags = PCGF_ACTIVE;
  if (page_cgroup_assign_new_page_cgroup(page, pc)) {
   /*
    * an another charge is added to this page already.
```

Subject: [PATCH] memory cgroup enhancements [4/5] memory cgroup statistics

Posted by KAMEZAWA Hiroyuki on Tue, 16 Oct 2007 10:27:10 GMT

View Forum Message <> Reply to Message

Add statistics account infrastructure for memory controller.

Changelog v1 -> v2
 - Removed Charge/Uncharge counter
 - reflected comments.
   - changes __move_lists() args.
   - changes __mem_cgroup_stat_add() name, comment and added VM_BUG_ON

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 |  120 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++----
 1 file changed, 113 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
@@ -35,6 +35,64 @@ 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_PAGECACHE, /* # of pages charged as cache */
+ MEM_CGROUP_STAT_RSS,    /* # of pages charged as rss */
+
+ /*
+  * usage = charge - uncharge.
+  */
+ MEM_CGROUP_STAT_ACTIVE,    /* # of pages in active list */
+ MEM_CGROUP_STAT_INACTIVE,  /* # of pages on inactive list */
+
+ 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];
+};
+
+/*
+ * For batching....mem_cgroup_charge_statistics()(see below).
+ * MUST be called under preempt_disable().
+ */
+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();
+#ifdef CONFIG_PREEMPT
+ VM_BUG_ON(preempt_count() == 0);
+#endif
+ stat->cpustat[cpu].count[idx] += val;
+}
+
+static inline void mem_cgroup_stat_inc(struct mem_cgroup_stat *stat,
+  enum mem_cgroup_stat_index idx)
+{
```

```
+ preempt_disable();
+ __mem_cgroup_stat_add(stat, idx, 1);
+ preempt_enable();
+}
+
+static inline void mem_cgroup_stat_dec(struct mem_cgroup_stat *stat,
+  enum mem_cgroup_stat_index idx)
+{
+ preempt_disable();
+ __mem_cgroup_stat_add(stat, idx, -1);
+ preempt_enable();
+}
+
+
+/*
  * 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 +121,10 @@ struct mem_cgroup {
   */
  spinlock_t lru_lock;
  unsigned long control_type; /* control RSS or RSS+Pagecache */
+ /*
+  * statistics.
+  */
+ struct mem_cgroup_stat stat;
 };

 /*
@@ -96,6 +158,33 @@ enum {
  MEM_CGROUP_TYPE_MAX,
 };

+/*
+ * Batched statistics modification.
+ * We have to modify several values at charge/uncharge..
+ */
+static inline void
+mem_cgroup_charge_statistics(struct mem_cgroup *mem, int flags, int charge)
+{
+ int val = (charge)? 1 : -1;
+ struct mem_cgroup_stat *stat = &mem->stat;
+ preempt_disable();
+
+ if (flags & PCGF_PAGECACHE)
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_PAGECACHE, val);
+ else
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_RSS, val);
```

```
+
+ if (flags & PCGF_ACTIVE)
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE, val);
+ else
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE, val);
+
+ preempt_enable();
+}
+
+
+
+
 static struct mem_cgroup init_mem_cgroup;

 static inline
@@ -209,12 +298,27 @@ clear_page_cgroup(struct page *page, str

 static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
+ int moved = 0;
+ struct mem_cgroup *mem = pc->mem_cgroup;
+
+ if (active && (pc->flags & PCGF_ACTIVE) == 0)
+  moved = 1; /* Move from inactive to active */
+ else if (!active && (pc->flags & PCGF_ACTIVE))
+  moved = -1; /* Move from active to inactive */
+
+ if (moved) {
+  struct mem_cgroup_stat *stat = &mem->stat;
+  preempt_disable();
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE, moved);
+  __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE, -moved);
+  preempt_enable();
+ }
  if (active) {
   pc->flags |= PCGF_ACTIVE;
-  list_move(&pc->lru, &pc->mem_cgroup->active_list);
+  list_move(&pc->lru, &mem->active_list);
  } else {
   pc->flags &= ~PCGF_ACTIVE;
-  list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
+  list_move(&pc->lru, &mem->inactive_list);
  }
 }

@@ -233,15 +337,12 @@ int task_in_mem_cgroup(struct task_struc
  */
 void mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
- struct mem_cgroup *mem;
  if (!pc)
   return;

- mem = pc->mem_cgroup;
-
- spin_lock(&mem->lru_lock);
+ spin_lock(&pc->mem_cgroup->lru_lock);
  __mem_cgroup_move_lists(pc, active);
- spin_unlock(&mem->lru_lock);
+ spin_unlock(&pc->mem_cgroup->lru_lock);
 }

 unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
@@ -440,6 +541,9 @@ noreclaim:
   goto retry;
  }

+ /* Update statistics vector */
+ mem_cgroup_charge_statistics(mem, pc->flags, true);
+
  spin_lock_irqsave(&mem->lru_lock, flags);
  list_add(&pc->lru, &mem->active_list);
  spin_unlock_irqrestore(&mem->lru_lock, flags);
@@ -505,6 +609,7 @@ void mem_cgroup_uncharge(struct page_cgr
    spin_lock_irqsave(&mem->lru_lock, flags);
    list_del_init(&pc->lru);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
+   mem_cgroup_charge_statistics(mem, pc->flags, false);
    kfree(pc);
   }
  }
@@ -577,6 +682,7 @@ mem_cgroup_force_empty_list(struct mem_c
    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
    count = 1; /* 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 [5/5] show statistics by memory.stat file per cgroup
Posted by KAMEZAWA Hiroyuki on Tue, 16 Oct 2007 10:28:10 GMT
View Forum Message <> Reply to Message

Show accounted information of memory cgroup by memory.stat file

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 |   52 ++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 52 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>

@@ -833,6 +834,53 @@ 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_PAGECACHE] = { "page_cache", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_RSS] = { "rss", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_ACTIVE] = { "active", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_INACTIVE] = { "inactive", 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++) {
+  unsigned int cpu;
+  s64 val;
+
+  val = 0;
+  for (cpu = 0; cpu < NR_CPUS; cpu++)
+   val += stat->cpustat[cpu].count[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",
@@ -860,6 +908,10 @@ 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: Re: [PATCH] memory cgroup enhancements [0/5] intro
Posted by Balbir Singh on Tue, 16 Oct 2007 18:20:28 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
```
> This patch set adds
>  - force_empty interface, which drops all charges in memory cgroup.
>    This enables rmdir() against unused memory cgroup.
>  - the memory cgroup statistics accounting.
>
> Based on 2.6.23-mm1 + http://lkml.org/lkml/2007/10/12/53
>
> Changes from previous version is
>  - merged comments.
>  - based on 2.6.23-mm1
>  - removed Charge/Uncharge counter.
>
> [1/5] ... force_empty patch
> [2/5] ... remember page is charged as page-cache patch
> [3/5] ... remember page is on which list patch
> [4/5] ... memory cgroup statistics patch
> [5/5] ... show statistics patch
>
> tested on x86-64/fake-NUMA + CONFIG_PREEMPT=y/n (for testing preempt_disable())
>
> Any comments are welcome.
>
```

Hi, KAMEZAWA-San,

I would prefer these patches to go in once the fixes that you've posted
earlier have gone in (the migration fix series). I am yet to test the
migration fix per-se, but the series seemed quite fine to me. Andrew
could you please pick it up.

> -Kame
>

--
 Warm Regards,
 Balbir Singh
 Linux Technology Center
 IBM, ISTL

Subject: Re: [PATCH] memory cgroup enhancements [0/5] intro
Posted by akpm on Tue, 16 Oct 2007 18:28:43 GMT
View Forum Message <> Reply to Message

On Tue, 16 Oct 2007 23:50:28 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> > [1/5] ... force_empty patch
> > [2/5] ... remember page is charged as page-cache patch
> > [3/5] ... remember page is on which list patch
> > [4/5] ... memory cgroup statistics patch
> > [5/5] ... show statistics patch
> >
> > tested on x86-64/fake-NUMA + CONFIG_PREEMPT=y/n (for testing preempt_disable())
> >
> > Any comments are welcome.
> >
>
> Hi, KAMEZAWA-San,
>
> I would prefer these patches to go in once the fixes that you've posted
> earlier have gone in (the migration fix series). I am yet to test the
> migration fix per-se, but the series seemed quite fine to me. Andrew
> could you please pick it up.

It's in my backlog somewhere. I'm not paying much attention to things
which don't look like 2.6.24 material at present...

Subject: Re: [PATCH] memory cgroup enhancements [0/5] intro
Posted by Balbir Singh on Tue, 16 Oct 2007 18:40:54 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
> On Tue, 16 Oct 2007 23:50:28 +0530
> Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>
>>> [1/5] ... force_empty patch

>>> [2/5] ... remember page is charged as page-cache patch
>>> [3/5] ... remember page is on which list patch
>>> [4/5] ... memory cgroup statistics patch
>>> [5/5] ... show statistics patch
>>>
>>> tested on x86-64/fake-NUMA + CONFIG_PREEMPT=y/n (for testing preempt_disable())
>>>
>>> Any comments are welcome.
>>>
>> Hi, KAMEZAWA-San,
>>
>> I would prefer these patches to go in once the fixes that you've posted
>> earlier have gone in (the migration fix series). I am yet to test the
>> migration fix per-se, but the series seemed quite fine to me. Andrew
>> could you please pick it up.
>
> It's in my backlog somewhere. I'm not paying much attention to things
> which don't look like 2.6.24 material at present...

Aah.. That makes sense.. Thanks for clarifying.

--
 Warm Regards,
 Balbir Singh
 Linux Technology Center
 IBM, ISTL

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

Subject: Re: [PATCH] memory cgroup enhancements [1/5] force_empty for memory cgroup
Posted by David Rientjes on Wed, 17 Oct 2007 04:17:06 GMT
View Forum Message <> Reply to Message

On Tue, 16 Oct 2007, KAMEZAWA Hiroyuki wrote:

> 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.

```
>
```

If there's no tasks in the cgroup, then how can there be any charges
against its memory controller? Is the memory not being uncharged when a
task exits or is moved to another cgroup?

If the only use of this is for rmdir, why not just make it part of the
rmdir operation on the memory cgroup if there are no tasks by default?

```
> Tested and worked well on x86_64/fake-NUMA system.
>
> 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 |  102
+++++++++++++++++++++++++++++++++++++++++++++++++++++----
>  1 file changed, 95 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();
```

Unrelated change?

```
>    }
>   }
>  }
> @@ -543,6 +537,70 @@ 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 relcaim page itself, just removes page_cgroup.
```

Reclaim?

```
> + */
> +static void
> +mem_cgroup_force_empty_list(struct mem_cgroup *mem, struct list_head *list)
> +{
> + struct page_cgroup *pc;
> + struct page *page;
> + int count = SWAP_CLUSTER_MAX;
> + unsigned long flags;
> +
> + spin_lock_irqsave(&mem->lru_lock, flags);
> +
> + while (!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);
```

Don't you need {lock,unlock}_page_cgroup(page) here to avoid a true race
with mem_cgroup_charge() right after you set pc->ref_cnt to 0 here?

```
> +  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
> +   count = 1; /* being uncharged ? ...do relax */
> +
> +  if (--count == 0) {
> +   spin_unlock_irqrestore(&mem->lru_lock, flags);
> +   cond_resched();
> +   spin_lock_irqsave(&mem->lru_lock, flags);
```

Instead of this hack, it's probably easier to just goto a label placed
before spin_lock_irqsave() at the top of the function.

```
> +   count = SWAP_CLUSTER_MAX;
> +  }
> + }
> + spin_unlock_irqrestore(&mem->lru_lock, flags);
> +}
> +
> +/*
> + * make mem_cgroup's charge to be 0 if there is no binded task.
> + * This enables deleting this mem_cgroup.
> + */
> +
> +int mem_cgroup_force_empty(struct mem_cgroup *mem)
> +{
> + int ret = -EBUSY;
> + css_get(&mem->css);
> + 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);
> + }
```

This implementation as a while loop looks very suspect since
mem_cgroup_force_empty_list() uses while (!list_empty(list)) as well.
Perhaps it's just easier here as

```
 if (list_empty(&mem->active_list) && list_empty(&mem->inactive_list))
  return 0;
```

```
> + 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 +686,31 @@ 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;
> +}
> +
> +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)
> +{
> + static const char buf[2] = "0";
> + return simple_read_from_buffer((void __user *)userbuf, nbytes,
> +   ppos, buf, strlen(buf));
```

Reading memory.force_empty is pretty useless, so why allow it to be read
at all?

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

Subject: Re: [PATCH] memory cgroup enhancements [3/5] record pc is on active list
Posted by David Rientjes on Wed, 17 Oct 2007 04:17:24 GMT
View Forum Message <> Reply to Message

On Tue, 16 Oct 2007, KAMEZAWA Hiroyuki wrote:

> Remember page_cgroup is on active_list or not in page_cgroup->flags.
>
> Against 2.6.23-mm1.

```
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>
>
>  mm/memcontrol.c |   12 ++++++++----
>  1 file changed, 8 insertions(+), 4 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
> @@ -85,6 +85,7 @@ struct page_cgroup {
>       /* mapped and cached states     */
>   int  flags;
>  #define PCGF_PAGECACHE  (0x1) /* charged as page-cache */
> +#define PCGF_ACTIVE  (0x2) /* this is on cgroup's active list */
```

Please move these flag #defines out of the struct definition.

Subject: Re: [PATCH] memory cgroup enhancements [1/5] force_empty for memory cgroup
Posted by Balbir Singh on Wed, 17 Oct 2007 05:05:58 GMT

View Forum Message <> Reply to Message

David Rientjes wrote:
> On Tue, 16 Oct 2007, KAMEZAWA Hiroyuki wrote:
>
>> 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.
>>
>
> If there's no tasks in the cgroup, then how can there be any charges
> against its memory controller? Is the memory not being uncharged when a
> task exits or is moved to another cgroup?
>

David,

Since we account even for page and swap cache, there might be pages left
over even after all tasks are gone.

> If the only use of this is for rmdir, why not just make it part of the
> rmdir operation on the memory cgroup if there are no tasks by default?
>

That's a good idea, but sometimes an administrator might want to force
a cgroup empty and start fresh without necessary deleting the cgroup.

```
>> Tested and worked well on x86_64/fake-NUMA system.
>>
>> 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 |  102
+++++++++++++++++++++++++++++++++++++++++++++++++++++++----
>>  1 file changed, 95 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();
>
> Unrelated change?
>
```

Yes

```
>>    }
>>   }
>>  }
>> @@ -543,6 +537,70 @@ 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 relcaim page itself, just removes page_cgroup.
>
> Reclaim?
>
>> + */
>> +static void
>> +mem_cgroup_force_empty_list(struct mem_cgroup *mem, struct list_head *list)
>> +{
>> + struct page_cgroup *pc;
>> + struct page *page;
>> + int count = SWAP_CLUSTER_MAX;
>> + unsigned long flags;
>> +
>> + spin_lock_irqsave(&mem->lru_lock, flags);
>> +
>> + while (!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);
>
> Don't you need {lock,unlock}_page_cgroup(page) here to avoid a true race
> with mem_cgroup_charge() right after you set pc->ref_cnt to 0 here?
>
```

I think clear_page_cgroup() below checks for a possible race.

```
>> +  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
>> +   count = 1; /* being uncharged ? ...do relax */
>> +
>> +  if (--count == 0) {
>> +   spin_unlock_irqrestore(&mem->lru_lock, flags);
>> +   cond_resched();
>> +   spin_lock_irqsave(&mem->lru_lock, flags);
>
> Instead of this hack, it's probably easier to just goto a label placed
> before spin_lock_irqsave() at the top of the function.
>
```

I am not convinced of this hack either, specially the statement of
setting count to SWAP_CLUSTER_MAX.

```
>> +   count = SWAP_CLUSTER_MAX;
>> +  }
>> + }
>> + spin_unlock_irqrestore(&mem->lru_lock, flags);
>> +}
>> +
>> +/*
>> + * make mem_cgroup's charge to be 0 if there is no binded task.
>> + * This enables deleting this mem_cgroup.
>> + */
>> +
>> +int mem_cgroup_force_empty(struct mem_cgroup *mem)
>> +{
>> + int ret = -EBUSY;
>> + css_get(&mem->css);
>> + 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);
>> + }
>
> This implementation as a while loop looks very suspect since
> mem_cgroup_force_empty_list() uses while (!list_empty(list)) as well.
> Perhaps it's just easier here as
>
>  if (list_empty(&mem->active_list) && list_empty(&mem->inactive_list))
>   return 0;
>
```

Do we VM_BUG_ON() in case the lists are not empty after calling
mem_cgroup_force_empty_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 +686,31 @@ 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;
>> +}
>> +
>> +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)
>> +{
>> + static const char buf[2] = "0";
>> + return simple_read_from_buffer((void __user *)userbuf, nbytes,
>> +   ppos, buf, strlen(buf));
>
```

> Reading memory.force_empty is pretty useless, so why allow it to be read
> at all?

I agree, this is not required. I wonder if we could set permissions at
group level to mark this file as *write only*. We could use the new
read_uint and write_uint callbacks for reading/writing integers.

--
 Warm Regards,
 Balbir Singh
 Linux Technology Center
 IBM, ISTL

Subject: Re: [PATCH] memory cgroup enhancements [1/5] force_empty for memory cgroup
Posted by KAMEZAWA Hiroyuki on Wed, 17 Oct 2007 05:16:09 GMT

View Forum Message <> Reply to Message

On Tue, 16 Oct 2007 21:17:06 -0700 (PDT)
David Rientjes <rientjes@google.com> wrote:
> > This is useful to invoke rmdir() against memory cgroup successfully.
> >
>
> If there's no tasks in the cgroup, then how can there be any charges
> against its memory controller? Is the memory not being uncharged when a
> task exits or is moved to another cgroup?
Page Caches are charged as page cache. (even if not mmmaped)
So, empty cgroup can maintain some amount of charges as page cache.

>
> If the only use of this is for rmdir, why not just make it part of the
> rmdir operation on the memory cgroup if there are no tasks by default?
>
This patch is just a step for that. I myself think your way is better.
If we can make a concensus and good code, we can do this in automatic way
by rmdir().
BTW, personally, I uses this *force_empty* for test memory cgroup.
(like drop_caches)

> > Tested and worked well on x86_64/fake-NUMA system.
> >

```
> > 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 |  102
++++++++++++++++++++++++++++++++++++++++++++++++++++++----
> >  1 file changed, 95 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();
>
> Unrelated change?
>
No. force_reclaim causes this race. but not BUG. so printk is removed.


> >    }
> >   }
> >  }
> > @@ -543,6 +537,70 @@ 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 relcaim page itself, just removes page_cgroup.
> 
> Reclaim?
> 
yes..

> > + */
> > +static void
> > +mem_cgroup_force_empty_list(struct mem_cgroup *mem, struct list_head *list)
> > +{
> > + struct page_cgroup *pc;
> > + struct page *page;
> > + int count = SWAP_CLUSTER_MAX;
> > + unsigned long flags;
> > +
> > + spin_lock_irqsave(&mem->lru_lock, flags);
> > +
> > + while (!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);
> 
> Don't you need {lock,unlock}_page_cgroup(page) here to avoid a true race
> with mem_cgroup_charge() right after you set pc->ref_cnt to 0 here?
> 
My patch (already posted) add check like this in mem_cgroup_charge
--
retry:
        lock_page_cgroup(page);
        pc = page_get_page_cgroup(page);
        /*
         * The page_cgroup exists and the page has already been accounted
         */
        if (pc) {
                if (unlikely(!atomic_inc_not_zero(&pc->ref_cnt))) {
                        /* this page is under being uncharged ? */
                        unlock_page_cgroup(page);
                        cpu_relax();
                goto retry;
            } else {
                unlock_page_cgroup(page);
                goto done;
            }
        }
--
```
and clear_page_cgroup() takes a lock.
Then,
  - If atoimc_set(&pc->ref_cnt, 0) goes before mem_cgroup_charge()'s
    atomic_inc_not_zero(&pc->ref_cnt), there is no race.
  - If atomic_inc_not_zero(&pc->ref_cnt) goes before atomic_set(),
    just ignore increment and drop this charge.

```
> > +  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
> > +   count = 1; /* being uncharged ? ...do relax */
> > +
> > +  if (--count == 0) {
> > +   spin_unlock_irqrestore(&mem->lru_lock, flags);
> > +   cond_resched();
> > +   spin_lock_irqsave(&mem->lru_lock, flags);
>
> Instead of this hack, it's probably easier to just goto a label placed
> before spin_lock_irqsave() at the top of the function.
>
```
Hmm, ok. I'll consider this again.

```
> > +   count = SWAP_CLUSTER_MAX;
> > +  }
> > + }
> > + spin_unlock_irqrestore(&mem->lru_lock, flags);
> > +}
> > +
> > +/*
> > + * make mem_cgroup's charge to be 0 if there is no binded task.
> > + * This enables deleting this mem_cgroup.
> > + */
> > +
> > +int mem_cgroup_force_empty(struct mem_cgroup *mem)
> > +{
> > + int ret = -EBUSY;
> > + css_get(&mem->css);
> > + 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);
> > + }
>
> This implementation as a while loop looks very suspect since
> mem_cgroup_force_empty_list() uses while (!list_empty(list)) as well.
> Perhaps it's just easier here as
>
>  if (list_empty(&mem->active_list) && list_empty(&mem->inactive_list))
>   return 0;
>
will fix.


> > + 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 +686,31 @@ 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;
> > +}
> > +
> > +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)
> > +{
> > + static const char buf[2] = "0";
> > + return simple_read_from_buffer((void __user *)userbuf, nbytes,
> > +   ppos, buf, strlen(buf));
>
> Reading memory.force_empty is pretty useless, so why allow it to be read
> at all?
```

Ah, yes. How can I make this as read-only ? just remove .read ?

Thanks,
-Kame

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

---

Subject: Re: [PATCH] memory cgroup enhancements [3/5] record pc is on active list
Posted by KAMEZAWA Hiroyuki on Wed, 17 Oct 2007 05:16:27 GMT

View Forum Message <> Reply to Message

On Tue, 16 Oct 2007 21:17:24 -0700 (PDT)
David Rientjes <rientjes@google.com> wrote:

```
> On Tue, 16 Oct 2007, KAMEZAWA Hiroyuki wrote:
>
> > Remember page_cgroup is on active_list or not in page_cgroup->flags.
> >
> > Against 2.6.23-mm1.
> >
> > Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> > Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>
> >
> >  mm/memcontrol.c |   12 ++++++++----
> >  1 file changed, 8 insertions(+), 4 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
> > @@ -85,6 +85,7 @@ struct page_cgroup {
> >       /* mapped and cached states     */
> >   int  flags;
> >  #define PCGF_PAGECACHE  (0x1) /* charged as page-cache */
> > +#define PCGF_ACTIVE  (0x2) /* this is on cgroup's active list */
>
> Please move these flag #defines out of the struct definition.
>
```
ok

-Kame

Subject: Re: [PATCH] memory cgroup enhancements [0/5] intro
Posted by KAMEZAWA Hiroyuki on Wed, 17 Oct 2007 05:19:10 GMT

View Forum Message <> Reply to Message

On Tue, 16 Oct 2007 11:28:43 -0700
Andrew Morton <akpm@linux-foundation.org> wrote:

> > I would prefer these patches to go in once the fixes that you've posted
> > earlier have gone in (the migration fix series). I am yet to test the
> > migration fix per-se, but the series seemed quite fine to me. Andrew
> > could you please pick it up.
>
> It's in my backlog somewhere. I'm not paying much attention to things
> which don't look like 2.6.24 material at present...
>

Ah, ok. I'll wait and keep this set as RFC for a while.

Thanks,
-Kame

Subject: Re: [PATCH] memory cgroup enhancements [1/5] force_empty for memory cgroup
Posted by KAMEZAWA Hiroyuki on Wed, 17 Oct 2007 05:26:42 GMT
View Forum Message <> Reply to Message

On Wed, 17 Oct 2007 10:35:58 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> > If the only use of this is for rmdir, why not just make it part of the
> > rmdir operation on the memory cgroup if there are no tasks by default?
> >
>
> That's a good idea, but sometimes an administrator might want to force
> a cgroup empty and start fresh without necessary deleting the cgroup.
>
I'll make a "automatic force_empty at rmdir()" patch as another patch depends
on this. If we make concensus that "force_empty interface is redundant", I'll
remove it later.

> I am not convinced of this hack either, specially the statement of
> setting count to SWAP_CLUSTER_MAX.
>
Just because I think there should be "unlock and rest" in this busy loop,
I need some number. Should I define other number ?
as
#define FORCE_RECALIM_BATCH (128)

```
> >> +  /* drop all page_cgroup in inactive_list */
> >> +  mem_cgroup_force_empty_list(mem, &mem->inactive_list);
> >> + }
> >
> > This implementation as a while loop looks very suspect since
> > mem_cgroup_force_empty_list() uses while (!list_empty(list)) as well.
> > Perhaps it's just easier here as
> >
> >  if (list_empty(&mem->active_list) && list_empty(&mem->inactive_list))
> >   return 0;
> >
>
> Do we VM_BUG_ON() in case the lists are not empty after calling
> mem_cgroup_force_empty_list()
>
```
Okay, I will add.

> > Reading memory.force_empty is pretty useless, so why allow it to be read
> > at all?
>

> I agree, this is not required. I wonder if we could set permissions at
> group level to mark this file as *write only*. We could use the new
> read_uint and write_uint callbacks for reading/writing integers.
>
ok, will remove.

Thanks,
-Kame

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

---

Subject: Re: [PATCH] memory cgroup enhancements [1/5] force_empty for memory cgroup
Posted by David Rientjes on Wed, 17 Oct 2007 05:38:18 GMT
View Forum Message <> Reply to Message

On Wed, 17 Oct 2007, KAMEZAWA Hiroyuki wrote:

```
> > > +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)
> > > +{
> > > + static const char buf[2] = "0";
> > > + return simple_read_from_buffer((void __user *)userbuf, nbytes,
> > > +   ppos, buf, strlen(buf));
> >
> > Reading memory.force_empty is pretty useless, so why allow it to be read
> > at all?
>
> Ah, yes. How can I make this as read-only ? just remove .read ?
>
> Thanks,
```

You mean make it write-only? Typically it would be as easy as only
specifying a mode of S_IWUSR so that it can only be written to, the
S_IFREG is already provided by cgroup_add_file().

Unfortunately, cgroups do not appear to allow that. It hardcodes
the permissions of 0644 | S_IFREG into the cgroup_create_file() call from
cgroup_add_file(), which is a bug. Cgroup files should be able to be
marked as read-only or write-only depending on their semantics.

So until that bug gets fixed and you're allowed to pass your own file modes to cgroup_add_files(), you'll have to provide the read function.

  David

Subject: Re: [PATCH] memory cgroup enhancements [1/5] force_empty for memory cgroup

Posted by KAMEZAWA Hiroyuki on Wed, 17 Oct 2007 05:50:09 GMT

View Forum Message <> Reply to Message

On Tue, 16 Oct 2007 22:38:18 -0700 (PDT)
David Rientjes <rientjes@google.com> wrote:
> > Thanks,
>
> You mean make it write-only? Typically it would be as easy as only
> specifying a mode of S_IWUSR so that it can only be written to, the
> S_IFREG is already provided by cgroup_add_file().
>
> Unfortunately, cgroups do not appear to allow that. It hardcodes
> the permissions of 0644 | S_IFREG into the cgroup_create_file() call from
> cgroup_add_file(), which is a bug. Cgroup files should be able to be
> marked as read-only or write-only depending on their semantics.
>
> So until that bug gets fixed and you're allowed to pass your own file
> modes to cgroup_add_files(), you'll have to provide the read function.
>
Hmm. it seems I have to read current cgroup code more.
Thank you for advice.

Regards,
-Kame