Subject: [RFC] memory controller : backgorund reclaim and avoid excessive locking [5/5] lazy page_cgroup free
Posted by KAMEZAWA Hiroyuki on Thu, 14 Feb 2008 08:36:24 GMT
View Forum Message <> Reply to Message

A big lock contetion of memory controller is mz->lru_lock.

This is acquired when
 1. add to lru
 2. remove from lru
 3. scan lru list

It seems 1. and 3. are unavoidable. but 2. can be delayed.

This patch make removing page_cgorup from lru-list be lazy and batched.
(Like pagevec..)

This patch adds a new flag page_cgroup and make lru scan routine
ignores it.

I think this reduces lock contention especially when
 - several tasks are exiting at once.
 - several files are removed at once.

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

```
 mm/memcontrol.c |   86 +++++++++++++++++++++++++++++++++++++++++++++++++++++-------
 1 files changed, 76 insertions(+), 10 deletions(-)

Index: linux-2.6.24-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.24-mm1.orig/mm/memcontrol.c
+++ linux-2.6.24-mm1/mm/memcontrol.c
@@ -100,6 +100,7 @@ struct mem_cgroup_per_zone {
  struct list_head inactive_list;
  unsigned long count[NR_MEM_CGROUP_ZSTAT];
 };
+
 /* Macro for accessing counter */
 #define MEM_CGROUP_ZSTAT(mz, idx) ((mz)->count[(idx)])

@@ -157,9 +158,17 @@ struct mem_cgroup {
   atomic_t reclaimers;
   wait_queue_head_t waitq;
  } throttle;
+ /*
```

```
+  * For lazy freeing (not GC)
+  */
+ struct {
+  struct mem_cgroup_per_zone *mz;
+  int   num;
+#define GARBAGE_MAXSIZE  (16)
+  struct page_cgroup *vec[GARBAGE_MAXSIZE];
+ } garbage[NR_CPUS];
 };

-
 /*
  * We use the lower bit of the page->page_cgroup pointer as a bit spin
  * lock. We need to ensure that page->page_cgroup is atleast two
@@ -176,12 +185,14 @@ struct page_cgroup {
  struct list_head lru;  /* per cgroup LRU list */
  struct page *page;
  struct mem_cgroup *mem_cgroup;
+ struct mem_cgroup_per_zone *mz; /* now belongs to...*/
  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 */
 #define PAGE_CGROUP_FLAG_ACTIVE (0x2) /* page is active in this cgroup */
+#define PAGE_CGROUP_FLAG_GARBAGE (0x4)  /* this page_cgroup is just a garbage */

 static inline int page_cgroup_nid(struct page_cgroup *pc)
 {
@@ -235,8 +246,12 @@ static inline struct mem_cgroup_per_zone
 page_cgroup_zoneinfo(struct page_cgroup *pc)
 {
  struct mem_cgroup *mem = pc->mem_cgroup;
- int nid = page_cgroup_nid(pc);
- int zid = page_cgroup_zid(pc);
+ int nid, zid;
+
+ if (pc->mz)
+  return pc->mz;
+ nid = page_cgroup_nid(pc);
+ zid = page_cgroup_zid(pc);

  return mem_cgroup_zoneinfo(mem, nid, zid);
 }
@@ -360,8 +375,10 @@ static struct page_cgroup *clear_page_cg
  /* lock and clear */
  lock_page_cgroup(page);
  ret = page_get_page_cgroup(page);
```

```
- if (likely(ret == pc))
+ if (likely(ret == pc)) {
   page_assign_page_cgroup(page, NULL);
+  pc->flags |= PAGE_CGROUP_FLAG_GARBAGE;
+ }
  unlock_page_cgroup(page);
  return ret;
 }
@@ -377,6 +394,7 @@ static void __mem_cgroup_remove_list(str
   MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_INACTIVE) -= 1;

  mem_cgroup_charge_statistics(pc->mem_cgroup, pc->flags, false);
+ pc->mz = NULL;
  list_del_init(&pc->lru);
 }

@@ -392,6 +410,7 @@ static void __mem_cgroup_add_list(struct
   MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_ACTIVE) += 1;
   list_add(&pc->lru, &mz->active_list);
  }
+ pc->mz = mz;
  mem_cgroup_charge_statistics(pc->mem_cgroup, pc->flags, true);
 }

@@ -408,10 +427,12 @@ static void __mem_cgroup_move_lists(stru
  if (active) {
   MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_ACTIVE) += 1;
   pc->flags |= PAGE_CGROUP_FLAG_ACTIVE;
+  pc->mz = mz;
   list_move(&pc->lru, &mz->active_list);
  } else {
   MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_INACTIVE) += 1;
   pc->flags &= ~PAGE_CGROUP_FLAG_ACTIVE;
+  pc->mz = mz;
   list_move(&pc->lru, &mz->inactive_list);
  }
 }
@@ -607,7 +628,6 @@ unsigned long mem_cgroup_isolate_pages(u
    nr_taken++;
   }
  }
-
  list_splice(&pc_list, src);
  spin_unlock(&mz->lru_lock);

@@ -703,6 +723,7 @@ static void drain_all_borrow(struct mem_
  on_each_cpu(drain_local_borrow, mem, 0, 1);
 }
```

```
+
 /*
  * Charge the memory controller for page usage.
  * Return
@@ -820,6 +841,7 @@ retry:
  pc->flags = PAGE_CGROUP_FLAG_ACTIVE;
  if (ctype == MEM_CGROUP_CHARGE_TYPE_CACHE)
   pc->flags |= PAGE_CGROUP_FLAG_CACHE;
+ pc->mz = NULL;

  if (!page || page_cgroup_assign_new_page_cgroup(page, pc)) {
   /*
@@ -872,6 +894,52 @@ int mem_cgroup_cache_charge(struct page
  return ret;
 }

+/* called under irq-off */
+void local_free_garbages(void *data)
+{
+ struct mem_cgroup *mem = data;
+ int cpu = smp_processor_id();
+ struct page_cgroup *pc, *tmp;
+ LIST_HEAD(frees);
+ int i;
+ if (!mem->garbage[cpu].mz)
+  return;
+ spin_lock(&mem->garbage[cpu].mz->lru_lock);
+ for (i = 0; i < mem->garbage[cpu].num; i++) {
+  pc = mem->garbage[cpu].vec[i];
+  __mem_cgroup_remove_list(pc);
+  list_add(&pc->lru, &frees);
+ }
+ mem->garbage[cpu].num = 0;
+ spin_unlock(&mem->garbage[cpu].mz->lru_lock);
+
+ list_for_each_entry_safe(pc, tmp, &frees, lru)
+  kfree(pc);
+}
+
+void __mem_cgroup_free_garbage(struct mem_cgroup *mem,
+    struct page_cgroup *pc)
+{
+ unsigned long flags;
+ int cpu;
+ local_irq_save(flags);
+ cpu = smp_processor_id();
+ if ((pc->mz != mem->garbage[cpu].mz) ||
```

```
+     (mem->garbage[cpu].num == GARBAGE_MAXSIZE))
+  local_free_garbages(mem);
+ mem->garbage[cpu].mz = pc->mz;
+ mem->garbage[cpu].vec[mem->garbage[cpu].num] = pc;
+ mem->garbage[cpu].num++;
+ local_irq_restore(flags);
+}
+
+void all_free_garbages(struct mem_cgroup *mem)
+{
+ on_each_cpu(local_free_garbages, mem, 0, 1);
+}
+
+
+
 /*
  * Uncharging is always a welcome operation, we never complain, simply
  * uncharge. This routine should be called with lock_page_cgroup held
@@ -881,7 +949,6 @@ void mem_cgroup_uncharge(struct page_cgr
  struct mem_cgroup *mem;
  struct mem_cgroup_per_zone *mz;
  struct page *page;
- unsigned long flags;

  /*
   * Check if our page_cgroup is valid
@@ -901,10 +968,7 @@ void mem_cgroup_uncharge(struct page_cgr
    mem = pc->mem_cgroup;
    css_put(&mem->css);
    mem_cgroup_return_and_uncharge(mem, PAGE_SIZE);
-   spin_lock_irqsave(&mz->lru_lock, flags);
-   __mem_cgroup_remove_list(pc);
-   spin_unlock_irqrestore(&mz->lru_lock, flags);
-   kfree(pc);
+    __mem_cgroup_free_garbage(mem, pc);
   }
   lock_page_cgroup(page);
  }
@@ -1050,11 +1114,15 @@ mem_cgroup_force_empty_list(struct mem_c
  if (list_empty(list))
   return;
 retry:
+ all_free_garbages(mem);
  count = FORCE_UNCHARGE_BATCH;
  spin_lock_irqsave(&mz->lru_lock, flags);

  while (--count && !list_empty(list)) {
   pc = list_entry(list->prev, struct page_cgroup, lru);
```

```
+  /* If there are still garbage, exit and retry */
+  if (pc->flags & PAGE_CGROUP_FLAG_GARBAGE)
+   break;
   page = pc->page;
   /* Avoid race with charge */
   atomic_set(&pc->ref_cnt, 0);
@@ -1102,6 +1170,7 @@ int mem_cgroup_force_empty(struct mem_cg
     mem_cgroup_force_empty_list(mem, mz, 0);
    }
   drain_all_borrow(mem);
+  all_free_garbages(mem);
  }
  ret = 0;
 out:
@@ -1416,6 +1485,7 @@ static void mem_cgroup_pre_destroy(struc
  mem_cgroup_force_empty(mem);
  if (mem->daemon.kthread)
   kthread_stop(mem->daemon.kthread);
+ all_free_garbages(mem);
 }

 static void mem_cgroup_destroy(struct cgroup_subsys *ss,
```