Subject: [RFC] memory controller : backgorund reclaim and avoid excessive locking [4/5] borrow resource
Posted by KAMEZAWA Hiroyuki on Thu, 14 Feb 2008 08:33:45 GMT
View Forum Message <> Reply to Message

One of contended lock is counter->lock.

Now, counter->usage is changed by PAGE_SIZE. This patch changes
this to be PAGE_SIZE * borrow_factor and cache "borrow" in
per cpu area.

This reduce # of lock against counter->lock.

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

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
@@ -47,7 +47,8 @@ enum mem_cgroup_stat_index {
   */
  MEM_CGROUP_STAT_CACHE,   /* # of pages charged as cache */
  MEM_CGROUP_STAT_RSS,   /* # of pages charged as rss */
-
+ MEM_CGROUP_STAT_BORROW,   /* # of per-cpu borrow resource from
+   global resource */
  MEM_CGROUP_STAT_NSTATS,
 };

@@ -134,6 +135,9 @@ struct mem_cgroup {
  struct mem_cgroup_lru_info info;

  int prev_priority; /* for recording reclaim priority */
+
+ int borrow_unit;   /* size of unit for borrowing resource */
+
  /*
   * statistics.
   */
@@ -611,6 +615,92 @@ unsigned long mem_cgroup_isolate_pages(u
  return nr_taken;
 }

+/* FIXME? we assume that size is always PAGE_SIZE. */
+
+static int mem_cgroup_borrow_and_charge(struct mem_cgroup *mem, int size)
+{

```
+ unsigned long flags;
+ int ret;
+
+ ret = 0;
+
+ local_irq_save(flags);
+ if (mem->borrow_unit) {
+  int cpu;
+  s64 *bwp;
+  cpu = smp_processor_id();
+  bwp = &mem->stat.cpustat[cpu].count[MEM_CGROUP_STAT_BORROW];
+  if (*bwp > size) {
+   *bwp -= size;
+   goto out;
+  }
+  /* try to charge */
+  ret = res_counter_charge(&mem->res, mem->borrow_unit);
+  if (!ret) { /* success */
+   *bwp += (mem->borrow_unit - size);
+   goto out;
+  }
+ }
+ spin_lock(&mem->res.lock);
+ ret = res_counter_charge_locked(&mem->res, size);
+ spin_unlock(&mem->res.lock);
+out:
+ local_irq_restore(flags);
+ return ret;
+}
+
+static void mem_cgroup_return_and_uncharge(struct mem_cgroup *mem, int size)
+{
+ unsigned long flags;
+ int uncharge_size = 0;
+
+ local_irq_save(flags);
+ if (mem->borrow_unit) {
+  int limit = mem->borrow_unit * 2;
+  int cpu;
+  s64 *bwp;
+  cpu = smp_processor_id();
+  bwp = &mem->stat.cpustat[cpu].count[MEM_CGROUP_STAT_BORROW];
+  *bwp += size;
+  if (*bwp > limit) {
+   uncharge_size = *bwp - mem->borrow_unit;
+   *bwp = mem->borrow_unit;
+  }
+ } else
```

```
+  uncharge_size = size;
+
+ if (uncharge_size) {
+  spin_lock(&mem->res.lock);
+  res_counter_uncharge_locked(&mem->res, size);
+  spin_unlock(&mem->res.lock);
+ }
+ local_irq_restore(flags);
+
+ return;
+}
+
+static void drain_local_borrow(void *data)
+{
+ int cpu;
+ int borrow;
+ unsigned long flags;
+ struct mem_cgroup *mem = data;
+
+ local_irq_save(flags);
+ cpu = smp_processor_id();
+ borrow = mem->stat.cpustat[cpu].count[MEM_CGROUP_STAT_BORROW];
+ mem->stat.cpustat[cpu].count[MEM_CGROUP_STAT_BORROW] = 0;
+ spin_lock(&mem->res.lock);
+ res_counter_uncharge_locked(&mem->res, borrow);
+ spin_unlock(&mem->res.lock);
+ local_irq_restore(flags);
+}
+
+static void drain_all_borrow(struct mem_cgroup *mem)
+{
+ on_each_cpu(drain_local_borrow, mem, 0, 1);
+}
+
 /*
  * Charge the memory controller for page usage.
  * Return
@@ -681,7 +771,7 @@ retry:
   * If we created the page_cgroup, we should free it on exceeding
   * the cgroup limit.
   */
- while (res_counter_charge(&mem->res, PAGE_SIZE)) {
+ while (mem_cgroup_borrow_and_charge(mem, PAGE_SIZE)) {
   int ret;
   if (!(gfp_mask & __GFP_WAIT))
    goto out;
@@ -709,6 +799,8 @@ retry:
   if (res_counter_check_under_limit(&mem->res))
```

```
    continue;

+  if (nr_retries < MEM_CGROUP_RECLAIM_RETRIES)
+   drain_all_borrow(mem);
   if (!nr_retries--) {
    mem_cgroup_out_of_memory(mem, gfp_mask);
    goto out;
@@ -805,7 +897,7 @@ void mem_cgroup_uncharge(struct page_cgr
   if (clear_page_cgroup(page, pc) == pc) {
    mem = pc->mem_cgroup;
    css_put(&mem->css);
-   res_counter_uncharge(&mem->res, PAGE_SIZE);
+   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);
@@ -1005,6 +1097,7 @@ int mem_cgroup_force_empty(struct mem_cg
     /* drop all page_cgroup in inactive_list */
     mem_cgroup_force_empty_list(mem, mz, 0);
    }
+  drain_all_borrow(mem);
  }
  ret = 0;
 out:
@@ -1109,12 +1202,29 @@ static u64 mem_throttle_read(struct cgro
  return (u64)mem->throttle.limit;
 }

+static int mem_bulkratio_write(struct cgroup *cont, struct cftype *cft, u64 val)
+{
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
+ int unit = val * PAGE_SIZE;
+ if (unit > (PAGE_SIZE << (MAX_ORDER/2)))
+  return -EINVAL;
+ mem->borrow_unit = unit;
+ return 0;
+}
+
+static u64 mem_bulkratio_read(struct cgroup *cont, struct cftype *cft)
+{
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
+ return (u64)(mem->borrow_unit/PAGE_SIZE);
+}
+
 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, },
+ [MEM_CGROUP_STAT_BORROW] = { "borrow", 1, },
 };

 static int mem_control_stat_show(struct seq_file *m, void *arg)
@@ -1205,6 +1315,11 @@ static struct cftype mem_cgroup_files[]
   .read_uint  = mem_throttle_read,
  },
  {
+  .name = "bulkratio",
+  .write_uint = mem_bulkratio_write,
+  .read_uint = mem_bulkratio_read,
+ },
+ {
   .name = "stat",
   .open = mem_control_stat_open,
  },
@@ -1279,6 +1394,8 @@ mem_cgroup_create(struct cgroup_subsys *
  mem->throttle.limit = 10240; /* maybe enough big for no throttle */
  atomic_set(&mem->throttle.reclaimers, 0);

+ mem->borrow_unit = 0; /* Work at strict/precise mode as default */
+
  return &mem->css;
 free_out:
  for_each_node_state(node, N_POSSIBLE)
```

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