Subject: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [3/10] per-zone acti
Posted by KAMEZAWA Hiroyuki on Tue, 27 Nov 2007 03:00:48 GMT
View Forum Message <> Reply to Message

Counting active/inactive per-zone in memory controller.

This patch adds per-zone status in memory cgroup.
These values are often read (as per-zone value) by page reclaiming.

In current design, per-zone stat is just a unsigned long value and
not an atomic value because they are modified only under lru_lock.
(So, atomic_ops is not necessary.)

This patch adds ACTIVE and INACTIVE per-zone status values.

For handling per-zone status, this patch adds
  struct mem_cgroup_per_zone {
  ...
  }
and some helper functions. This will be useful to add per-zone objects
in mem_cgroup.

This patch turns memory controller's early_init to be 0 for calling
kmalloc() in initialization.

Changelog V2 -> V3
  - fixed comments.

Changelog V1 -> V2
  - added mem_cgroup_per_zone struct.
     This will help following patches to implement per-zone objects and
     pack them into a struct.
  - added __mem_cgroup_add_list() and __mem_cgroup_remove_list()
  - fixed page migration handling.
  - renamed zstat to info (per-zone-info)
    This will be place for per-zone information(lru, lock, ..)
  - use page_cgroup_nid()/zid() funcs.

Acked-by: Balbir Singh <balbir@linux.vnet.ibm.com>
Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

 mm/memcontrol.c | 164 ++---
 1 file changed, 157 insertions(+), 7 deletions(-)

Index: linux-2.6.24-rc3-mm1/mm/memcontrol.c
===

```
--- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-26 16:39:00.000000000 +0900
+++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-26 16:39:02.000000000 +0900
@@ -78,6 +78,31 @@
 }

 /*
+ * per-zone information in memory controller.
+ */
+
+enum mem_cgroup_zstat_index {
+ MEM_CGROUP_ZSTAT_ACTIVE,
+ MEM_CGROUP_ZSTAT_INACTIVE,
+
+ NR_MEM_CGROUP_ZSTAT,
+};
+
+struct mem_cgroup_per_zone {
+ unsigned long count[NR_MEM_CGROUP_ZSTAT];
+};
+/* Macro for accessing counter */
+#define MEM_CGROUP_ZSTAT(mz, idx) ((mz)->count[(idx)])
+
+struct mem_cgroup_per_node {
+ struct mem_cgroup_per_zone zoneinfo[MAX_NR_ZONES];
+};
+
+struct mem_cgroup_lru_info {
+ struct mem_cgroup_per_node *nodeinfo[MAX_NUMNODES];
+};
+
+/*
  * 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,
@@ -101,6 +126,7 @@
  */
  struct list_head active_list;
  struct list_head inactive_list;
+ struct mem_cgroup_lru_info info;
  /*
   * spin_lock to protect the per cgroup LRU
   */
@@ -158,6 +184,7 @@
  MEM_CGROUP_CHARGE_TYPE_MAPPED,
 };

+
 /*
```

```
  * Always modified under lru lock. Then, not necessary to preempt_disable()
  */
@@ -173,7 +200,39 @@
      MEM_CGROUP_STAT_CACHE, val);
  else
    __mem_cgroup_stat_add_safe(stat, MEM_CGROUP_STAT_RSS, val);
+}

+static inline struct mem_cgroup_per_zone *
+mem_cgroup_zoneinfo(struct mem_cgroup *mem, int nid, int zid)
+{
+ if (!mem->info.nodeinfo[nid])
+  return NULL;
+ return &mem->info.nodeinfo[nid]->zoneinfo[zid];
+}
+
+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);
+
+ return mem_cgroup_zoneinfo(mem, nid, zid);
+}
+
+static unsigned long mem_cgroup_get_all_zonestat(struct mem_cgroup *mem,
+      enum mem_cgroup_zstat_index idx)
+{
+ int nid, zid;
+ struct mem_cgroup_per_zone *mz;
+ u64 total = 0;
+
+ for_each_online_node(nid)
+  for (zid = 0; zid < MAX_NR_ZONES; zid++) {
+   mz = mem_cgroup_zoneinfo(mem, nid, zid);
+   total += MEM_CGROUP_ZSTAT(mz, idx);
+  }
+ return total;
 }

 static struct mem_cgroup init_mem_cgroup;
@@ -286,12 +345,51 @@
  return ret;
 }

+static void __mem_cgroup_remove_list(struct page_cgroup *pc)
+{
```

```
+ int from = pc->flags & PAGE_CGROUP_FLAG_ACTIVE;
+ struct mem_cgroup_per_zone *mz = page_cgroup_zoneinfo(pc);
+
+ if (from)
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_ACTIVE) -= 1;
+ else
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_INACTIVE) -= 1;
+
+ mem_cgroup_charge_statistics(pc->mem_cgroup, pc->flags, false);
+ list_del_init(&pc->lru);
+}
+
+static void __mem_cgroup_add_list(struct page_cgroup *pc)
+{
+ int to = pc->flags & PAGE_CGROUP_FLAG_ACTIVE;
+ struct mem_cgroup_per_zone *mz = page_cgroup_zoneinfo(pc);
+
+ if (!to) {
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_INACTIVE) += 1;
+  list_add(&pc->lru, &pc->mem_cgroup->inactive_list);
+ } else {
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_ACTIVE) += 1;
+  list_add(&pc->lru, &pc->mem_cgroup->active_list);
+ }
+ mem_cgroup_charge_statistics(pc->mem_cgroup, pc->flags, true);
+}
+
 static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
+ int from = pc->flags & PAGE_CGROUP_FLAG_ACTIVE;
+ struct mem_cgroup_per_zone *mz = page_cgroup_zoneinfo(pc);
+
+ if (from)
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_ACTIVE) -= 1;
+ else
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_INACTIVE) -= 1;
+
  if (active) {
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_ACTIVE) += 1;
   pc->flags |= PAGE_CGROUP_FLAG_ACTIVE;
   list_move(&pc->lru, &pc->mem_cgroup->active_list);
  } else {
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_INACTIVE) += 1;
   pc->flags &= ~PAGE_CGROUP_FLAG_ACTIVE;
   list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
  }
@@ -511,8 +609,7 @@
```

```
  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);
+ __mem_cgroup_add_list(pc);
  spin_unlock_irqrestore(&mem->lru_lock, flags);

 done:
@@ -576,13 +673,13 @@
    css_put(&mem->css);
    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);
+   __mem_cgroup_remove_list(pc);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
    kfree(pc);
   }
  }
 }
+
 /*
  * Returns non-zero if a page (under migration) has valid page_cgroup member.
  * Refcnt of page_cgroup is incremented.
@@ -614,16 +711,26 @@
 void mem_cgroup_page_migration(struct page *page, struct page *newpage)
 {
  struct page_cgroup *pc;
+ struct mem_cgroup *mem;
+ unsigned long flags;
 retry:
  pc = page_get_page_cgroup(page);
  if (!pc)
   return;
+ mem = pc->mem_cgroup;
  if (clear_page_cgroup(page, pc) != pc)
   goto retry;
+
+ spin_lock_irqsave(&mem->lru_lock, flags);
+
+ __mem_cgroup_remove_list(pc);
  pc->page = newpage;
  lock_page_cgroup(newpage);
  page_assign_page_cgroup(newpage, pc);
  unlock_page_cgroup(newpage);
+ __mem_cgroup_add_list(pc);
+
+ spin_unlock_irqrestore(&mem->lru_lock, flags);
```

```
  return;
 }

@@ -651,10 +758,11 @@
   /* Avoid race with charge */
   atomic_set(&pc->ref_cnt, 0);
   if (clear_page_cgroup(page, pc) == pc) {
+   int active;
    css_put(&mem->css);
+   active = pc->flags & PAGE_CGROUP_FLAG_ACTIVE;
    res_counter_uncharge(&mem->res, PAGE_SIZE);
-   list_del_init(&pc->lru);
-   mem_cgroup_charge_statistics(mem, pc->flags, false);
+   __mem_cgroup_remove_list(pc);
    kfree(pc);
   } else  /* being uncharged ? ...do relax */
    break;
@@ -833,6 +941,17 @@
   seq_printf(m, "%s %lld\n", mem_cgroup_stat_desc[i].msg,
     (long long)val);
  }
+ /* showing # of active pages */
+ {
+  unsigned long active, inactive;
+
+  inactive = mem_cgroup_get_all_zonestat(mem_cont,
+      MEM_CGROUP_ZSTAT_INACTIVE);
+  active = mem_cgroup_get_all_zonestat(mem_cont,
+      MEM_CGROUP_ZSTAT_ACTIVE);
+  seq_printf(m, "active %ld\n", (active) * PAGE_SIZE);
+  seq_printf(m, "inactive %ld\n", (inactive) * PAGE_SIZE);
+ }
  return 0;
 }

@@ -886,12 +1005,25 @@
  },
 };

+static int alloc_mem_cgroup_per_zone_info(struct mem_cgroup *mem, int node)
+{
+ struct mem_cgroup_per_node *pn;
+
+ pn = kmalloc_node(sizeof(*pn), GFP_KERNEL, node);
+ if (!pn)
+  return 1;
+ mem->info.nodeinfo[node] = pn;
+ memset(pn, 0, sizeof(*pn));
```

```
+ return 0;
+}
+
 static struct mem_cgroup init_mem_cgroup;

 static struct cgroup_subsys_state *
 mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
 {
  struct mem_cgroup *mem;
+ int node;

  if (unlikely((cont->parent) == NULL)) {
   mem = &init_mem_cgroup;
@@ -907,7 +1039,19 @@
  INIT_LIST_HEAD(&mem->inactive_list);
  spin_lock_init(&mem->lru_lock);
  mem->control_type = MEM_CGROUP_TYPE_ALL;
+ memset(&mem->info, 0, sizeof(mem->info));
+
+ for_each_node_state(node, N_POSSIBLE)
+  if (alloc_mem_cgroup_per_zone_info(mem, node))
+   goto free_out;
+
  return &mem->css;
+free_out:
+ for_each_node_state(node, N_POSSIBLE)
+  kfree(mem->info.nodeinfo[node]);
+ if (cont->parent != NULL)
+  kfree(mem);
+ return NULL;
 }

 static void mem_cgroup_pre_destroy(struct cgroup_subsys *ss,
@@ -920,6 +1064,12 @@
 static void mem_cgroup_destroy(struct cgroup_subsys *ss,
     struct cgroup *cont)
 {
+ int node;
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
+
+ for_each_node_state(node, N_POSSIBLE)
+  kfree(mem->info.nodeinfo[node]);
+
  kfree(mem_cgroup_from_cont(cont));
 }

@@ -972,5 +1122,5 @@
  .destroy = mem_cgroup_destroy,
```

```
  .populate = mem_cgroup_populate,
  .attach = mem_cgroup_move_task,
- .early_init = 1,
+ .early_init = 0,
 };
```

_____

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