
Subject: [RFC][for -mm] memory controller enhancements for NUMA [10/10]
per-zone-lru

Posted by [KAMEZAWA Hiroyuki](#) on Wed, 14 Nov 2007 08:57:37 GMT

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

I think there is a consensus that memory controller needs per-zone lru.
But it was postponed that it seems some people tries to modify lru of zones.

Now, "scan" value, in mem_cgroup_isolate_pages(), handling is fixed. So,
demand of implementing per-zone-lru is raised.

This patch implements per-zone lru for memory cgroup.
I think this patch's style implementation can be adjusted to zone's lru
implementation changes if happens.

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

mm/memcontrol.c | 117 +++-----
1 file changed, 100 insertions(+), 17 deletions(-)

Index: linux-2.6.24-rc2-mm1/mm/memcontrol.c

=====

--- linux-2.6.24-rc2-mm1.orig/mm/memcontrol.c

+++ linux-2.6.24-rc2-mm1/mm/memcontrol.c

@@ -101,6 +101,11 @@ struct mem_cgroup_zonestat {
};

```
+struct mc_lru_head {  
+ struct list_head active_list[MAX_NR_ZONES];  
+ struct list_head inactive_list[MAX_NR_ZONES];  
+};  
+  
+/*  
+ * The memory controller data structure. The memory controller controls both  
+ * page cache and RSS per cgroup. We would eventually like to provide  
@@ -123,8 +128,8 @@ struct mem_cgroup {  
+ * per zone LRU lists.  
+ * TODO: Consider making these lists per zone  
+ */  
- struct list_head active_list;  
- struct list_head inactive_list;  
+ struct mc_lru_head *lrus[MAX_NUMNODES];  
+  
+ int prev_priority; /* used in memory reclaim (see zone's one) */  
+ /*  
+ * spin_lock to protect the per cgroup LRU
```

```

@@ -139,8 +144,20 @@ struct mem_cgroup {
    * Per zone statistics (used for memory reclaim)
    */
    struct mem_cgroup_zonestat zstat;
+#ifndef CONFIG_NUMA
+ struct lru_head local_head;
+#endif
};

+struct list_head *mem_cgroup_lru_head(struct mem_cgroup *mem, int nid, int zid,
+   int active)
+{
+ if (active)
+ return &mem->lru[nid]->active_list[zid];
+ else
+ return &mem->lru[nid]->inactive_list[zid];
+}
+
+/*
+ * 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
@@ -360,6 +377,9 @@ static struct page_cgroup *clear_page_cg
static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
{
    int from = pc->flags & PAGE_CGROUP_FLAG_ACTIVE;
+ struct list_head *head;
+
+ head = mem_cgroup_lru_head(pc->mem_cgroup, pc->nid, pc->zid, active);

    if (from && !active) /* active -> inactive */
        mem_cgroup_add_zonestat(pc, MEM_CGROUP_ZSTAT_INACTIVE, 1);
@@ -368,10 +388,10 @@ static void __mem_cgroup_move_lists(stru

    if (active) {
        pc->flags |= PAGE_CGROUP_FLAG_ACTIVE;
- list_move(&pc->lru, &pc->mem_cgroup->active_list);
+ list_move(&pc->lru, head);
    } else {
        pc->flags &= ~PAGE_CGROUP_FLAG_ACTIVE;
- list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
+ list_move(&pc->lru, head);
    }
}

@@ -502,11 +522,10 @@ unsigned long mem_cgroup_isolate_pages(u
LIST_HEAD(pc_list);
struct list_head *src;
struct page_cgroup *pc, *tmp;

```

```

+ int nid = z->zone_pgdat->node_id;
+ int zid = zone_idx(z);

- if (active)
- src = &mem_cont->active_list;
- else
- src = &mem_cont->inactive_list;
+ src = mem_cgroup_lru_head(mem_cont, nid, zid, active);

    spin_lock(&mem_cont->lru_lock);
    scan = 0;
@@ -564,6 +583,7 @@ static int mem_cgroup_charge_common(struct
    struct page_cgroup *pc;
    unsigned long flags;
    unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
+ struct list_head *head;

    /*
     * Should page_cgroup's go to their own slab?
@@ -676,11 +696,12 @@ noreclaim:
    goto retry;
}

+ head = mem_cgroup_lru_head(mem, pc->nid, pc->zid, 1);
    spin_lock_irqsave(&mem->lru_lock, flags);
    mem_cgroup_add_zonestat(pc, MEM_CGROUP_ZSTAT_TOTAL, 1);
    /* Update statistics vector */
    mem_cgroup_charge_statistics(mem, pc->flags, true);
- list_add(&pc->lru, &mem->active_list);
+ list_add(&pc->lru, head);
    spin_unlock_irqrestore(&mem->lru_lock, flags);

done:
@@ -814,6 +835,8 @@ mem_cgroup_force_empty_list(struct mem_c
    int count;
    unsigned long flags;

+ if (list_empty(list))
+ return;
    retry:
    count = FORCE_UNCHARGE_BATCH;
    spin_lock_irqsave(&mem->lru_lock, flags);
@@ -854,20 +877,26 @@ retry:
int mem_cgroup_force_empty(struct mem_cgroup *mem)
{
    int ret = -EBUSY;
+ int node, zid;
    css_get(&mem->css);

```



```

+ struct mc_lru_head *head;
+
+ for_each_node_state(node, N_POSSIBLE) {
+ head = kmalloc_node(sizeof(*head), GFP_KERNEL, node);
+ if (!head)
+ return 1;
+ mem->lrus[node] = head;
+ __memory_cgroup_init_lru_head(head);
+ }
+ return 0;
+}
+
+static void mem_cgroup_free_lru(struct mem_cgroup *mem)
+{
+ int node;
+
+ for_each_node_state(node, N_POSSIBLE)
+ kfree(mem->lrus[node]);
+}
+
+static int mem_cgroup_init_lru(struct mem_cgroup *mem)
+{
+ int zone;
+ mem->lrus[0] = &mem->local_lru;
+ __memory_cgroup_init_lru_head(mem->lrus[0]);
+}
+
+static void mem_cgroup_free_lru(struct mem_cgroup *mem)
+{
+}
+
+static struct mem_cgroup init_mem_cgroup;
+
+static struct cgroup_subsys_state *
@@ -1094,8 +1173,9 @@ mem_cgroup_create(struct cgroup_subsys *
return NULL;

res_counter_init(&mem->res);
- INIT_LIST_HEAD(&mem->active_list);
- INIT_LIST_HEAD(&mem->inactive_list);
+ if (mem_cgroup_init_lru(mem))
+ goto free_out;
+
spin_lock_init(&mem->lru_lock);
mem->control_type = MEM_CGROUP_TYPE_ALL;
memset(&mem->zstat, 0, sizeof(mem->zstat));

```

```

@@ -1111,6 +1191,8 @@ mem_cgroup_create(struct cgroup_subsys *
}
return &mem->css;
free_out:
+ mem_cgroup_free_lru(mem);
+
for_each_node_state(node, N_POSSIBLE)
kfree(mem->zstat.nodestat[node]);
if (cont->parent != NULL)
@@ -1131,6 +1213,7 @@ static void mem_cgroup_destroy(struct cg
int node;
struct mem_cgroup *mem = mem_cgroup_from_cont(cont);

+ mem_cgroup_free_lru(mem);
for_each_node_state(node, N_POSSIBLE)
kfree(mem->zstat.nodestat[node]);

```

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