
Subject: [RFC 4/4] memcg: NUMA background reclaim
Posted by [KAMEZAWA Hiroyuki](#) on Tue, 27 May 2008 05:06:52 GMT
[View Forum Message](#) <> [Reply to Message](#)

One aspect of difference in reclaim logic between global lru and memcg is

- * global LRU triggers memory reclaim at memory shortage.
- * memcg LRU triggers memory reclaim at excess of usage.

Then, global LRU `_know_` which node we should start reclaim from.

- * start from a node at memory shortage or
- * start from a node where memory allocation is waiting

WRT memcg, it's difficult to find where we should start because there is no memory shortage and LRU is splitted.
(But per-zone-LRU is definitely necessary for scalability.)

This patch tries to determine a node for starting recalim by checking ratio of inactive pages/active pages in a node. And trying to avoid starting from a node with relatively small usage.
Better algorithm is welcome.

Singed-off-by: KAMEZAWA Hiruyuki <kamezawa.hiroyu@jp.fujitsu.com>

Index: mm-2.6.26-rc2-mm1/mm/memcontrol.c

```
=====
--- mm-2.6.26-rc2-mm1.orig/mm/memcontrol.c
+++ mm-2.6.26-rc2-mm1/mm/memcontrol.c
@@ -578,7 +578,7 @@ retry:
     if (state == RES_OVER_LIMIT) {
         if (!(gfp_mask & __GFP_WAIT))
             goto out;
-    if (try_to_free_mem_cgroup_pages(mem, gfp_mask))
+    if (try_to_free_mem_cgroup_pages(mem, -1, gfp_mask))
         goto retry;
     /*
      * try_to_free_mem_cgroup_pages() might not give us a
@@ -801,7 +801,7 @@ int mem_cgroup_shrink_usage(struct mm_st
    rcu_read_unlock());

    do {
-    progress = try_to_free_mem_cgroup_pages(mem, gfp_mask);
+    progress = try_to_free_mem_cgroup_pages(mem, -1, gfp_mask);
    } while (!progress && --retry);

    if (!retry)
@@ -814,7 +814,7 @@ static void mem_cgroup_drop_all_pages(st
    {
        int progress;
```

```

while (!res_counter_empty(&mem->res)) {
- progress = try_to_free_mem_cgroup_pages(mem,
+ progress = try_to_free_mem_cgroup_pages(mem, -1,
    GFP_HIGHUSER_MOVABLE);
if (!progress) /* we did as much as possible */
    break;
@@ -912,6 +912,62 @@ out:
/*
 * background reclaim daemon.
 */
+
+ #ifdef CONFIG_NUMA
+ /*
+ * Because memory controller's memory reclaim doesn't come from memory shortage,
+ * we cannot know which node should be reclaimed in an easy way.
+ * This routine select a node with inactive pages to be a node for starting
+ * scanning.
+ */
+ int __select_best_node(struct mem_cgroup *mem)
+ {
+ int nid;
+ int best_node = -1;
+ unsigned long highest_inactive_ratio = 0;
+ unsigned long active, inactive, inactive_ratio, total, threshold, flags;
+ struct mem_cgroup_per_zone *mz;
+ int zid;
+
+ /*
+ * When a node's memory usage is smaller than
+ * total_usage/num_of_node * 75%, we don't select the node
+ */
+ total = mem->res.usage >> PAGE_SHIFT;
+ threshold = (total / num_node_state(N_HIGH_MEMORY)) * 3 / 4;
+
+ /*
+ * See nodemask.h, N_HIGH_MEMORY means that a node has memory
+ * can be used for user's memory.(i.e. not means HIGHMEM).
+ */
+ for_each_node_state(nid, N_HIGH_MEMORY) {
+ active = 0;
+ inactive = 0;
+
+ for (zid = 0; zid < MAX_NR_ZONES; zid++) {
+ mz = mem_cgroup_zoneinfo(mem, nid, zid);
+ spin_lock_irqsave(&mz->lru_lock, flags);
+ active += MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_ACTIVE);
+ inactive +=
+ MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_INACTIVE);

```

```

+ spin_unlock_irqrestore(&mz->lru_lock, flags);
+ }
+
+ if (active + inactive < threshold)
+ continue;
+ inactive_ratio = (inactive * 100) / (active + 1);
+ if (inactive_ratio > highest_inactive_ratio)
+ best_node = nid;
+ }
+ return best_node;
+}
+#else
+int __select_best_node(struct mem_cgroup *mem)
+{
+ return 0;
+}
+#endif
+
static int mem_cgroup_reclaim_daemon(void *data)
{
    DEFINE_WAIT(wait);
@@ -935,13 +991,9 @@ static int mem_cgroup_reclaim_daemon(voi
    continue;
}
    finish_wait(&mem->daemon.waitq, &wait);
- /*
- * memory resource controller doesn't see NUMA memory usage
- * balancing, because we cannot know what balancing is good.
- * TODO: some annotation or heuristics to detect which node
- * we should start reclaim from.
- */
- ret = try_to_free_mem_cgroup_pages(mem, GFP_HIGHUSER_MOVABLE);
+
+ ret = try_to_free_mem_cgroup_pages(mem,
+ __select_best_node(mem), GFP_HIGHUSER_MOVABLE);

    yield();
}

```

Index: mm-2.6.26-rc2-mm1/mm/vmscan.c

```

=====
--- mm-2.6.26-rc2-mm1.orig/mm/vmscan.c
+++ mm-2.6.26-rc2-mm1/mm/vmscan.c
@@ -1429,7 +1429,7 @@ unsigned long try_to_free_pages(struct z
#ifdef CONFIG_CGROUP_MEM_RES_CTLR

    unsigned long try_to_free_mem_cgroup_pages(struct mem_cgroup *mem_cont,
-    gfp_t gfp_mask)
+    int nid, gfp_t gfp_mask)

```

```

{
  struct scan_control sc = {
    .may_writepage = !laptop_mode,
@@ -1442,9 +1442,11 @@ unsigned long try_to_free_mem_cgroup_pag
  };
  struct zonelist *zonelist;

+ if (nid == -1)
+ nid = numa_node_id();
  sc.gfp_mask = (gfp_mask & GFP_RECLAIM_MASK) |
    (GFP_HIGHUSER_MOVABLE & ~GFP_RECLAIM_MASK);
- zonelist = NODE_DATA(numa_node_id()->node_zonelist);
+ zonelist = NODE_DATA(nid)->node_zonelist;
  return do_try_to_free_pages(zonelist, &sc);
}
#endif

```

Index: mm-2.6.26-rc2-mm1/include/linux/swap.h

```

=====
--- mm-2.6.26-rc2-mm1.orig/include/linux/swap.h
+++ mm-2.6.26-rc2-mm1/include/linux/swap.h
@@ -184,7 +184,7 @@ extern void swap_setup(void);
extern unsigned long try_to_free_pages(struct zonelist *zonelist, int order,
    gfp_t gfp_mask);
extern unsigned long try_to_free_mem_cgroup_pages(struct mem_cgroup *mem,
-    gfp_t gfp_mask);
+    int nid, gfp_t gfp_mask);
extern int __isolate_lru_page(struct page *page, int mode);
extern unsigned long shrink_all_memory(unsigned long nr_pages);
extern int vm_swappiness;

```

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