Subject: [RFC][for -mm] memory controller enhancements for NUMA [9/10] zone reclaim vs. cgroup reclaim isola
Posted by KAMEZAWA Hiroyuki on Wed, 14 Nov 2007 08:55:28 GMT
View Forum Message <> Reply to Message

When using memory controller, there are 2 levels of memory reclaim.
 1. zone memory relcaim because of system/zone memory shortage.
 2. memory cgroup memory reclaim because of hitting limit.

These two can be distinguished by sc->mem_cgroup parameter.
(we can use macro "scan_global_lru")

This patch tries to make memory cgroup reclaim routine avoid affecting system/zone memory reclaim. This patch inserts if (scan_global_lru(sc)) and hook to memory_cgroup reclaim support functions.

This patch can be a help for isolating system lru activity and group lru activity and shows what additional functions are necessary.

 * mem_cgroup_calc_mapped_ratio() ... calculate mapped ratio for cgroup.
 * mem_cgroup_reclaim_imbalance() ... calculate active/inactive balance in
    cgroup.
 * mem_cgroup_calc_reclaim_active() ... calclate the number of active pages to
   be scanned in this priority in mem_cgroup.

 * mem_cgroup_calc_reclaim_inactive() ... calclate the number of inactive pages
   to be scanned in this priority in mem_cgroup.

 * mem_cgroup_all_unreclaimable() .. checks cgroup's page is all unreclaimable
        or not.
 * mem_cgroup_get_reclaim_priority() ...
 * mem_cgroup_note_reclaim_priority() ... record reclaim priority (temporal)
 * mem_cgroup_remember_reclaim_priority()
       .... record reclaim priority as
      zone->prev_priority.
      This value is used for calc reclaim_mapped.

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

```
 mm/vmscan.c |  155 ++++++++++++++++++++++++++++++++++++++++++++++++++++++--------------------
 1 file changed, 105 insertions(+), 50 deletions(-)

Index: linux-2.6.24-rc2-mm1/mm/vmscan.c
===================================================================
--- linux-2.6.24-rc2-mm1.orig/mm/vmscan.c
+++ linux-2.6.24-rc2-mm1/mm/vmscan.c
@@ -863,7 +863,8 @@ static unsigned long shrink_inactive_lis
   __mod_zone_page_state(zone, NR_ACTIVE, -nr_active);
```

```
   __mod_zone_page_state(zone, NR_INACTIVE,
       -(nr_taken - nr_active));
-  zone->pages_scanned += nr_scan;
+  if (scan_global_lru(sc))
+   zone->pages_scanned += nr_scan;
   spin_unlock_irq(&zone->lru_lock);

   nr_scanned += nr_scan;
@@ -951,22 +952,30 @@ static inline int zone_is_near_oom(struc

 /*
  * Determine we should try to reclaim mapped pages.
+ * This is called only when sc->mem_cgroup is NULL.
  */
-static int calc_reclaim_mapped(struct zone *zone, int priority, int swappiness)
+static int calc_reclaim_mapped(struct scan_control *sc, struct zone *zone,
+    int priority)
 {
  long mapped_ratio;
  long distress;
  long swap_tendency;
  long imbalance;
  int reclaim_mapped;
+ int prev_priority;

- if (zone_is_near_oom(zone))
+ if (scan_global_lru(sc) && zone_is_near_oom(zone))
   return 1;
  /*
   * `distress' is a measure of how much trouble we're having
   * reclaiming pages.  0 -> no problems.  100 -> great trouble.
   */
- distress = 100 >> min(zone->prev_priority, priority);
+ if (scan_global_lru(sc))
+  prev_priority = zone->prev_priority;
+ else
+  prev_priority = mem_cgroup_get_reclaim_priority(sc->mem_cgroup);
+
+ distress = 100 >> min(prev_priority, priority);

  /*
   * The point of this algorithm is to decide when to start
@@ -974,9 +983,13 @@ static int calc_reclaim_mapped(struct zo
   * how much memory
   * is mapped.
   */
- mapped_ratio = ((global_page_state(NR_FILE_MAPPED) +
-   global_page_state(NR_ANON_PAGES)) * 100) /
```

```
-    vm_total_pages;
+ if (scan_global_lru(sc))
+  mapped_ratio = ((global_page_state(NR_FILE_MAPPED) +
+    global_page_state(NR_ANON_PAGES)) * 100) /
+     vm_total_pages;
+ else
+  mapped_ratio = mem_cgroup_calc_mapped_ratio(sc->mem_cgroup);
+
  /*
   * Now decide how much we really want to unmap some pages.  The
   * mapped ratio is downgraded - just because there's a lot of
@@ -989,7 +1002,7 @@ static int calc_reclaim_mapped(struct zo
   * A 100% value of vm_swappiness overrides this algorithm
   * altogether.
   */
- swap_tendency = mapped_ratio / 2 + distress + swappiness;
+ swap_tendency = mapped_ratio / 2 + distress + sc->swappiness;

  /*
   * If there's huge imbalance between active and inactive
@@ -1003,8 +1016,11 @@ static int calc_reclaim_mapped(struct zo
   * Avoid div by zero with nr_inactive+1, and max resulting
   * value is vm_total_pages.
   */
- imbalance  = zone_page_state(zone, NR_ACTIVE);
- imbalance /= zone_page_state(zone, NR_INACTIVE) + 1;
+ if (scan_global_lru(sc)) {
+  imbalance  = zone_page_state(zone, NR_ACTIVE);
+  imbalance /= zone_page_state(zone, NR_INACTIVE) + 1;
+ } else
+  imbalance = mem_cgroup_reclaim_imbalance(sc->mem_cgroup);

  /*
   * Reduce the effect of imbalance if swappiness is low,
@@ -1074,15 +1090,20 @@ static void shrink_active_list(unsigned
  int reclaim_mapped = 0;

  if (sc->may_swap)
-  reclaim_mapped = calc_reclaim_mapped(zone, priority,
-          sc->swappiness);
+  reclaim_mapped = calc_reclaim_mapped(sc, zone, priority);

  lru_add_drain();
  spin_lock_irq(&zone->lru_lock);
  pgmoved = sc->isolate_pages(nr_pages, &l_hold, &pgscanned, sc->order,
      ISOLATE_ACTIVE, zone,
      sc->mem_cgroup, 1);
- zone->pages_scanned += pgscanned;
```

```
+ /*
+  * zone->pages_scanned is used for detect zone's oom
+  * mem_cgroup remembers nr_scan by itself.
+  */
+ if (scan_global_lru(sc))
+  zone->pages_scanned += pgscanned;
+
  __mod_zone_page_state(zone, NR_ACTIVE, -pgmoved);
  spin_unlock_irq(&zone->lru_lock);

@@ -1175,25 +1196,39 @@ static unsigned long shrink_zone(int pri
  unsigned long nr_to_scan;
  unsigned long nr_reclaimed = 0;

- /*
-  * Add one to `nr_to_scan' just to make sure that the kernel will
-  * slowly sift through the active list.
-  */
- zone->nr_scan_active +=
-  (zone_page_state(zone, NR_ACTIVE) >> priority) + 1;
- nr_active = zone->nr_scan_active;
- if (nr_active >= sc->swap_cluster_max)
-  zone->nr_scan_active = 0;
- else
-  nr_active = 0;
+ if (scan_global_lru(sc)) {
+  /*
+   * Add one to nr_to_scan just to make sure that the kernel
+   * will slowly sift through the active list.
+   */
+  zone->nr_scan_active +=
+   (zone_page_state(zone, NR_ACTIVE) >> priority) + 1;
+  nr_active = zone->nr_scan_active;
+  zone->nr_scan_inactive +=
+   (zone_page_state(zone, NR_INACTIVE) >> priority) + 1;
+  nr_inactive = zone->nr_scan_inactive;
+  if (nr_inactive >= sc->swap_cluster_max)
+   zone->nr_scan_inactive = 0;
+  else
+   nr_inactive = 0;
+
+  if (nr_active >= sc->swap_cluster_max)
+   zone->nr_scan_active = 0;
+  else
+   nr_active = 0;
+ } else {
+  /*
+   * This reclaim occurs not because zone memory shortage but
```

```
+   * because memory controller hits its limit.
+   * Then, don't modify zone relcaim related data.
+   */
+  nr_active = mem_cgroup_calc_reclaim_active(sc->mem_cgroup,
+     zone, priority);
+
+  nr_inactive = mem_cgroup_calc_reclaim_inactive(sc->mem_cgroup,
+     zone, priority);
+ }

- zone->nr_scan_inactive +=
-  (zone_page_state(zone, NR_INACTIVE) >> priority) + 1;
- nr_inactive = zone->nr_scan_inactive;
- if (nr_inactive >= sc->swap_cluster_max)
-  zone->nr_scan_inactive = 0;
- else
-  nr_inactive = 0;

  while (nr_active || nr_inactive) {
   if (nr_active) {
@@ -1238,6 +1273,7 @@ static unsigned long shrink_zones(int pr
  unsigned long nr_reclaimed = 0;
  int i;

+
  sc->all_unreclaimable = 1;
  for (i = 0; zones[i] != NULL; i++) {
   struct zone *zone = zones[i];
@@ -1247,16 +1283,26 @@ static unsigned long shrink_zones(int pr

   if (!cpuset_zone_allowed_hardwall(zone, GFP_KERNEL))
    continue;
+  /*
+   * Take care memory controller reclaiming has minimal influence
+   * to global LRU.
+   */
+  if (scan_global_lru(sc)) {
+   note_zone_scanning_priority(zone, priority);

-  note_zone_scanning_priority(zone, priority);
-
-  if (zone_is_all_unreclaimable(zone) && priority != DEF_PRIORITY)
-   continue; /* Let kswapd poll it */
-
-  sc->all_unreclaimable = 0;
+   if (zone_is_all_unreclaimable(zone) &&
+      priority != DEF_PRIORITY)
+    continue; /* Let kswapd poll it */
```

```
+   sc->all_unreclaimable = 0;
+  } else {
+   sc->all_unreclaimable = 0;
+   mem_cgroup_note_reclaim_priority(sc->mem_cgroup,
+       priority);
+  }

   nr_reclaimed += shrink_zone(priority, zone, sc);
  }
+
  return nr_reclaimed;
 }

@@ -1285,15 +1331,19 @@ static unsigned long do_try_to_free_page
  int i;

  count_vm_event(ALLOCSTALL);
+ /*
+  * mem_cgroup will not do shrink_slab.
+  */
+ if (scan_global_lru(sc)) {
+  for (i = 0; zones[i] != NULL; i++) {
+   struct zone *zone = zones[i];

- for (i = 0; zones[i] != NULL; i++) {
-  struct zone *zone = zones[i];
-
-  if (!cpuset_zone_allowed_hardwall(zone, GFP_KERNEL))
-   continue;
+   if (!cpuset_zone_allowed_hardwall(zone, GFP_KERNEL))
+    continue;

-  lru_pages += zone_page_state(zone, NR_ACTIVE)
-    + zone_page_state(zone, NR_INACTIVE);
+   lru_pages += zone_page_state(zone, NR_ACTIVE)
+     + zone_page_state(zone, NR_INACTIVE);
+  }
  }

  for (priority = DEF_PRIORITY; priority >= 0; priority--) {
@@ -1350,14 +1400,19 @@ out:
   */
  if (priority < 0)
   priority = 0;
- for (i = 0; zones[i] != NULL; i++) {
-  struct zone *zone = zones[i];

-  if (!cpuset_zone_allowed_hardwall(zone, GFP_KERNEL))
```

```
-   continue;
+ if (scan_global_lru(sc)) {
+  for (i = 0; zones[i] != NULL; i++) {
+   struct zone *zone = zones[i];
+
+   if (!cpuset_zone_allowed_hardwall(zone, GFP_KERNEL))
+    continue;
+
+   zone->prev_priority = priority;
+  }
+ } else
+  mem_cgroup_record_reclaim_priority(sc->mem_cgroup, priority);

-  zone->prev_priority = priority;
- }
  return ret;
 }
```