Subject: [RFC][for -mm] memory controller enhancements for reclaiming take2 [5/8] throttling simultaneous cal
Posted by KAMEZAWA Hiroyuki on Mon, 03 Dec 2007 09:38:04 GMT
View Forum Message <> Reply to Message

Add throttling direct reclaim.

Trying heavy workload under memory controller, you'll see too much
iowait and system seems heavy. (This is not good.... memory controller
is usually used for isolating system workload)
And too much memory are reclaimed.

This patch adds throttling function for direct reclaim.
Currently, num_online_cpus/(4) + 1 threads can do direct memory reclaim
under memory controller.

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

```
 mm/memcontrol.c |   42 ++++++++++++++++++++++++++++++++++++++++++-
 1 file changed, 41 insertions(+), 1 deletion(-)

Index: linux-2.6.24-rc3-mm2/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc3-mm2.orig/mm/memcontrol.c
+++ linux-2.6.24-rc3-mm2/mm/memcontrol.c
@@ -36,6 +36,10 @@
 struct cgroup_subsys mem_cgroup_subsys;
 static const int MEM_CGROUP_RECLAIM_RETRIES = 5;

+/* Page reclaim throttle parameter */
+#define MEM_CGROUP_THROTTLE_RECLAIM_FACTOR (4)
+
+
 /*
  * Statistics for memory cgroup.
  */
@@ -133,6 +137,8 @@ struct mem_cgroup {

  unsigned long control_type; /* control RSS or RSS+Pagecache */
  int prev_priority; /* for recording reclaim priority */
+ atomic_t reclaimers; /* # of processes which calls reclaim */
+ wait_queue_head_t waitq;
  /*
   * statistics.
   */
@@ -565,6 +571,27 @@ unsigned long mem_cgroup_isolate_pages(u
  return nr_taken;
```

```
 }

+static void mem_cgroup_wait_reclaim(struct mem_cgroup *mem)
+{
+ DEFINE_WAIT(wait);
+ while (1) {
+  prepare_to_wait(&mem->waitq, &wait, TASK_INTERRUPTIBLE);
+  if (res_counter_check_under_limit(&mem->res)) {
+   finish_wait(&mem->waitq, &wait);
+   break;
+  }
+  schedule();
+  finish_wait(&mem->waitq, &wait);
+ }
+}
+
+/* throttle simlutaneous LRU scan */
+static int mem_cgroup_throttle_reclaim(struct mem_cgroup *mem)
+{
+ int limit = num_online_cpus()/MEM_CGROUP_THROTTLE_RECLAIM_FACTOR + 1;
+ return atomic_add_unless(&mem->reclaimers, 1, limit);
+}
+
 /*
  * Charge the memory controller for page usage.
  * Return
@@ -635,14 +662,24 @@ retry:
   */
  while (res_counter_charge(&mem->res, PAGE_SIZE)) {
   bool is_atomic = gfp_mask & GFP_ATOMIC;
+  int  ret;
   /*
    * We cannot reclaim under GFP_ATOMIC, fail the charge
    */
   if (is_atomic)
    goto noreclaim;

-  if (try_to_free_mem_cgroup_pages(mem, gfp_mask))
+  if (mem_cgroup_throttle_reclaim(mem)) {
+   ret = try_to_free_mem_cgroup_pages(mem, gfp_mask);
+   atomic_dec(&mem->reclaimers);
+   if (waitqueue_active(&mem->waitq))
+    wake_up_all(&mem->waitq);
+   if (ret)
+    continue;
+  } else {
+   mem_cgroup_wait_reclaim(mem);
    continue;
```

```
+ }

  /*
   * try_to_free_mem_cgroup_pages() might not give us a full
@@ -1169,6 +1206,9 @@ mem_cgroup_create(struct cgroup_subsys *
 mem->control_type = MEM_CGROUP_TYPE_ALL;
 memset(&mem->info, 0, sizeof(mem->info));

+ atomic_set(&mem->reclaimers, 0);
+ init_waitqueue_head(&mem->waitq);
+
 for_each_node_state(node, N_POSSIBLE)
  if (alloc_mem_cgroup_per_zone_info(mem, node))
   goto free_out;
```

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