

Subject: [RFC][for -mm] memory controller enhancements for reclaiming take2 [7/8]
background reclaim for mem

Posted by [KAMEZAWA Hiroyuki](#) on Mon, 03 Dec 2007 09:42:44 GMT

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

High Low watermark for page reclaiming in memory cgroup(1) and background reclaim routine.

- If USAGE is bigger than high watermark, background reclaim starts.
- If USAGE is lower than low watermark, background reclaim stops.

Each value is represented in bytes (See by kernel/res_counter.c)

Changelog v1 -> v2:

- merged high/low patch and background reclaim patch.
- removed automatic adjustment of high/low value.
- cleanups.
- add schedule_timedwait() in background reclaim's busy loop.
(LRU scan can be very heavy workload and consume CPU.)

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

From: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

mm/memcontrol.c | 97 ++++++-----
1 file changed, 92 insertions(+), 5 deletions(-)

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

@@ -30,6 +30,8 @@

#include <linux/spinlock.h>

#include <linux/fs.h>

#include <linux/seq_file.h>

+#include <linux/kthread.h>

+#include <linux/freezer.h>

#include <asm/uaccess.h>

@@ -117,11 +119,6 @@ struct mem_cgroup_lru_info {

- * 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,
- * to help the administrator determine what knobs to tune.
- _ *
- * TODO: Add a water mark for the memory controller. Reclaim will begin when
- * we hit the water mark. May be even add a low water mark, such that
- * no reclaim occurs from a cgroup at it's low water mark, this is

```

- * a feature that will be implemented much later in the future.
*/
struct mem_cgroup {
    struct cgroup_subsys_state css;
@@ -130,6 +127,13 @@ struct mem_cgroup {
    */
    struct res_counter res;
    /*
+ * background reclaim
+ */
+ struct {
+     wait_queue_head_t waitq;
+     struct task_struct *thread;
+ } daemon;
+ /*
+ * Per cgroup active and inactive list, similar to the
+ * per zone LRU lists.
+ */
@@ -401,6 +405,17 @@ static void __mem_cgroup_move_lists(stru
    }
}

+static void
+mem_cgroup_schedule_reclaim(struct mem_cgroup *mem)
+{
+ if (!unlikely(mem->daemon.thread))
+     return;
+ if (!waitqueue_active(&mem->daemon.waitq))
+     return;
+ wake_up_interruptible(&mem->daemon.waitq);
+}
+
+int task_in_mem_cgroup(struct task_struct *task, const struct mem_cgroup *mem)
+{
+     int ret;
@@ -708,6 +723,8 @@ noreclaim:
+     mem_cgroup_out_of_memory(mem, GFP_KERNEL);
+     goto free_pc;
+}
+ if (res_counter_above_high_watermark(&mem->res))
+     mem_cgroup_schedule_reclaim(mem);

+     atomic_set(&pc->ref_cnt, 1);
+     pc->mem_cgroup = mem;
@@ -863,6 +880,42 @@ retry:
+}

```

```

/*
+ * Background page reclaim daemon for memory controller.
+ */
+
+static int mem_cgroup_reclaim_daemon(void *data)
+{
+ DEFINE_WAIT(wait);
+ struct mem_cgroup *mem = data;
+
+ css_get(&mem->css);
+ current->flags |= PF_SWAPWRITE;
+ set_freezable();
+
+ while (!kthread_should_stop()) {
+ prepare_to_wait(&mem->daemon.waitq, &wait, TASK_INTERRUPTIBLE);
+ if (res_counter_below_low_watermark(&mem->res)) {
+ if (!kthread_should_stop()) {
+ schedule();
+ try_to_freeze();
+ }
+ finish_wait(&mem->daemon.waitq, &wait);
+ continue;
+ }
+ finish_wait(&mem->daemon.waitq, &wait);
+ try_to_free_mem_cgroup_pages(mem, GFP_HIGHUSER_MOVABLE);
+ /*
+  * throttle LRU scan for moderate reclaiming.
+  * not congestion wait.
+  */
+ schedule_timeout(HZ/10);
+ }
+
+ css_put(&mem->css);
+ return 0;
+}
+
+/*
+ * This routine traverse page_cgroup in given list and drop them all.
+ * This routine ignores page_cgroup->ref_cnt.
+ **And* this routine doesn't reclaim page itself, just removes page_cgroup.
@@ -1136,6 +1189,18 @@ static struct cftype mem_cgroup_files[]
    .read = mem_control_type_read,
    },
    {
+ .name = "low_watermark_in_bytes",
+ .private = RES_LWMARK,
+ .write = mem_cgroup_write,
+ .read = mem_cgroup_read,

```

```

+ },
+ {
+ .name = "high_watermark_in_bytes",
+ .private = RES_HWMARK,
+ .write = mem_cgroup_write,
+ .read = mem_cgroup_read,
+ },
+ {
+ .name = "force_empty",
+ .write = mem_force_empty_write,
+ .read = mem_force_empty_read,
@@ -1186,6 +1251,16 @@ static void free_mem_cgroup_per_zone_inf

static struct mem_cgroup init_mem_cgroup;

+static int __init mem_cgroup_reclaim_init(void)
+{
+ init_mem_cgroup.daemon.thread = kthread_run(mem_cgroup_reclaim_daemon,
+ &init_mem_cgroup, "memcontd");
+ if (IS_ERR(init_mem_cgroup.daemon.thread))
+ BUG();
+ return 0;
+}
+late_initcall(mem_cgroup_reclaim_init);
+
static struct cgroup_subsys_state *
mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
{
@@ -1213,6 +1288,17 @@ mem_cgroup_create(struct cgroup_subsys *
if (alloc_mem_cgroup_per_zone_info(mem, node))
goto free_out;

+ /* Memory Reclaim Daemon per cgroup */
+ init_waitqueue_head(&mem->daemon.waitq);
+ if (mem != &init_mem_cgroup) {
+ /* Complicated...but we cannot call kthread create here..*/
+ /* init call will later assign kthread */
+ mem->daemon.thread = kthread_run(mem_cgroup_reclaim_daemon,
+ mem, "memcontd");
+ if (IS_ERR(mem->daemon.thread))
+ goto free_out;
+ }
+
return &mem->css;
free_out:
for_each_node_state(node, N_POSSIBLE)
@@ -1227,6 +1313,7 @@ static void mem_cgroup_pre_destroy(struc
{

```

```
    struct mem_cgroup *mem = mem_cgroup_from_cont(cont);  
    mem_cgroup_force_empty(mem);  
+ kthread_stop(mem->daemon.thread);  
}
```

```
static void mem_cgroup_destroy(struct cgroup_subsys *ss,
```

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>
