
Subject: [RFC] memory controller : backgorund reclaim and avoid excessive locking
[2/5] background reclaim.

Posted by [KAMEZAWA Hiroyuki](#) on Thu, 14 Feb 2008 08:27:40 GMT

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

A patch for background reclaim based on high-low watermak in res_counter.
The daemon is called as "memcontd", here.

Implements following:

- * If res->usage is higher than res->hwmark, start memcontd.
- * memcontd calls try_to_free_pages.
- * memcontd stops if res->usage is lower than res->lwmark.

Maybe we can add more tunings but no extra params now.

ChangeLog:

- start "memcontd" at first change in hwmark.
(In old verion, it started at cgroup creation.)
- changed "relax" logic in memcontd daemon.

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

mm/memcontrol.c | 112

+++++

1 files changed, 109 insertions(+), 3 deletions(-)

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

=====

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

+++ linux-2.6.24-mm1/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>

@@ -136,6 +138,13 @@ struct mem_cgroup {

 * statistics.

 */

 struct mem_cgroup_stat stat;

+ /*

+ * background reclaim.

+ */

+ struct {

+ wait_queue_head_t waitq;

```

+ struct task_struct *kthread;
+ } daemon;
+ };

/*
@@ -504,6 +513,14 @@ long mem_cgroup_calc_reclaim_inactive(st
    return (nr_inactive >> priority);
}

+static inline void mem_cgroup_schedule_daemon(struct mem_cgroup *mem)
+{
+ if (likely(mem->daemon.kthread) && /* can be NULL at boot */)
+     waitqueue_active(&mem->daemon.waitq)
+ wake_up_interruptible(&mem->daemon.waitq);
+}
+
+
+unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
+    struct list_head *dst,
+    unsigned long *scanned, int order,
@@ -658,6 +675,9 @@ retry:
    congestion_wait(WRITE, HZ/10);
}

+ if (res_counter_above_hwmark(&mem->res))
+ mem_cgroup_schedule_daemon(mem);
+
+ atomic_set(&pc->ref_cnt, 1);
+ pc->mem_cgroup = mem;
+ pc->page = page;
@@ -762,6 +782,50 @@ void mem_cgroup_uncharge_page(struct pag
}

/*
+ * background page reclaim routine for cgroup.
+ */
+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_lwmark(&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);
+ /* Am I in hurry ? */
+ if (!res_counter_above_hwm(&mem->res)) {
+   /*
+    * Extra relaxing..memory reclaim is heavy work.
+    * we don't know there is I/O congestion or not.
+    * So use just relax rather than congeston_wait().
+    * HZ/10 is widely used value under /mm.
+    */
+   schedule_timeout(HZ/10);
+ } else {
+   /* Avoid occupation */
+   yield();
+ }
+ }
+
+ css_put(&mem->css);
+ return 0;
+}
+
+
+/*
+ * Returns non-zero if a page (under migration) has valid page_cgroup member.
+ * Refcnt of page_cgroup is incremented.
+ */
@@ -931,15 +995,40 @@ static ssize_t mem_cgroup_read(struct cg
    NULL);
}

+static DEFINE_MUTEX(modify_param_mutex);
+static ssize_t mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
+    struct file *file, const char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+   - return res_counter_write(&mem_cgroup_from_cont(cont)->res,
+   -   cft->private, userbuf, nbytes, ppos,
+   + int ret;
+   + struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
+   +
+   + mutex_lock(&modify_param_mutex);

```

```

+ /* Attach new background reclaim daemon.
+ This must be done before change values (for easy error handling */
+
+ if (cft->private == RES_HWMARK &&
+     !mem->daemon.kthread) {
+     struct task_struct *thr;
+     thr = kthread_run(mem_cgroup_reclaim_daemon, mem, "memcontd");
+     if (IS_ERR(thr)) {
+         ret = PTR_ERR(thr);
+         goto out;
+     }
+     mem->daemon.kthread = thr;
+ }
+ ret = res_counter_write(&mem->res, cft->private, userbuf, nbytes, ppos,
+     mem_cgroup_write_strategy);
+
+ /* Even on error, don't stop reclaim daemon here. not so problematic. */
+
+out:
+ mutex_unlock(&modify_param_mutex);
+ return ret;
+ }

+
+
+ static ssize_t mem_force_empty_write(struct cgroup *cont,
+     struct cftype *cft, struct file *file,
+     const char __user *userbuf,
+ @@ -1032,6 +1121,20 @@ static struct cftype mem_cgroup_files[]
+     .write = mem_cgroup_write,
+     .read = mem_cgroup_read,
+ },
+
+ {
+     .name = "lwmark_in_bytes",
+     .private = RES_LWMARK,
+     .write = mem_cgroup_write,
+     .read = mem_cgroup_read,
+ },
+
+ {
+     .name = "hwmark_in_bytes",
+     .private = RES_HWMARK,
+     .write = mem_cgroup_write,
+     .read = mem_cgroup_read,
+ },
+
+ {
+     .name = "failcnt",

```

```

        .private = RES_FAILCNT,
@@ -1110,7 +1213,8 @@ mem_cgroup_create(struct cgroup_subsys *
    for_each_node_state(node, N_POSSIBLE)
        if (alloc_mem_cgroup_per_zone_info(mem, node))
            goto free_out;
-
+ init_waitqueue_head(&mem->daemon.waitq);
+ mem->daemon.kthread = NULL;
    return &mem->css;
free_out:
    for_each_node_state(node, N_POSSIBLE)
@@ -1125,6 +1229,8 @@ static void mem_cgroup_pre_destroy(struc
{
    struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
    mem_cgroup_force_empty(mem);
+ if (mem->daemon.kthread)
+ kthread_stop(mem->daemon.kthread);
}

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>
