Subject: [RFC] [PATCH] memory controller background reclamation
Posted by yamamoto on Mon, 22 Oct 2007 09:43:49 GMT
View Forum Message <> Reply to Message

hi,

i implemented background reclamation for your memory controller and
did a few benchmarks with and without it. any comments?

YAMAMOTO Takashi

"time make -j4 bzImage" in a cgroup with 64MB limit:

 without patch:
  real   22m22.389s
  user   13m35.163s
  sys   6m12.283s

  memory.failcnt after a run: 106647

 with patch:
  real   19m25.860s
  user   14m10.549s
  sys   6m13.831s

  memory.failcnt after a run: 35366

"for x in 1 2 3;do time dd if=/dev/zero bs=1M count=300|tail;done"
in a cgroup with 16MB limit:

 without patch:
  300+0 records in
  300+0 records out
  314572800 bytes (315 MB) copied, 19.0058 seconds, 16.6 MB/s
  u 0.00s s 0.67s e 19.01s maj 90 min 2021 in 0 out 0
  u 0.48s s 7.22s e 93.13s maj 20475 min 210903 in 0 out 0
  300+0 records in
  300+0 records out
  314572800 bytes (315 MB) copied, 14.8394 seconds, 21.2 MB/s
  u 0.00s s 0.63s e 14.84s maj 53 min 1392 in 0 out 0
  u 0.48s s 7.00s e 94.39s maj 20125 min 211145 in 0 out 0
  300+0 records in
  300+0 records out
  314572800 bytes (315 MB) copied, 14.0635 seconds, 22.4 MB/s
  u 0.01s s 0.59s e 14.07s maj 50 min 1385 in 0 out 0
  u 0.57s s 6.93s e 91.54s maj 20359 min 210911 in 0 out 0

memory.failcnt after a run: 8804

with patch:
300+0 records in
300+0 records out
314572800 bytes (315 MB) copied, 5.94875 seconds, 52.9 MB/s
u 0.00s s 0.67s e 5.95s maj 206 min 5393 in 0 out 0
u 0.45s s 6.63s e 46.07s maj 21350 min 210252 in 0 out 0
300+0 records in
300+0 records out
314572800 bytes (315 MB) copied, 8.16441 seconds, 38.5 MB/s
u 0.00s s 0.60s e 8.17s maj 240 min 4614 in 0 out 0
u 0.45s s 6.56s e 54.56s maj 22298 min 209255 in 0 out 0
300+0 records in
300+0 records out
314572800 bytes (315 MB) copied, 4.60967 seconds, 68.2 MB/s
u 0.00s s 0.60s e 4.64s maj 196 min 4733 in 0 out 0
u 0.46s s 6.53s e 49.28s maj 22223 min 209384 in 0 out 0

memory.failcnt after a run: 1589

--- linux-2.6.23-rc8-mm2-cgkswapd/mm/memcontrol.c.BACKUP 2007-10-01 17:19:57.000000000 +0900
+++ linux-2.6.23-rc8-mm2-cgkswapd/mm/memcontrol.c 2007-10-22 13:46:01.000000000 +0900
@@ -27,6 +27,7 @@
#include <linux/rcupdate.h>
#include <linux/swap.h>
#include <linux/spinlock.h>
+#include <linux/workqueue.h>
#include <linux/fs.h>

#include <asm/uaccess.h>
@@ -63,6 +64,8 @@ struct mem_cgroup {
*/
spinlock_t lru_lock;
unsigned long control_type; /* control RSS or RSS+Pagecache */
+
+ struct work_struct reclaim_work;
};

/*
@@ -95,6 +98,9 @@ enum {

static struct mem_cgroup init_mem_cgroup;

```
+static DEFINE_MUTEX(mem_cgroup_workqueue_init_lock);
+static struct workqueue_struct *mem_cgroup_workqueue;
+
 static inline
 struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont)
 {
@@ -250,6 +256,69 @@ unsigned long mem_cgroup_isolate_pages(u
  return nr_taken;
 }

+static int
+mem_cgroup_need_reclaim(struct mem_cgroup *mem)
+{
+ struct res_counter * const cnt = &mem->res;
+ int doreclaim;
+ unsigned long flags;
+
+ /* XXX should be in res_counter */
+ /* XXX should not hardcode a watermark */
+ spin_lock_irqsave(&cnt->lock, flags);
+ doreclaim = cnt->usage > cnt->limit / 4 * 3;
+ spin_unlock_irqrestore(&cnt->lock, flags);
+
+ return doreclaim;
+}
+
+static void
+mem_cgroup_schedule_reclaim_if_necessary(struct mem_cgroup *mem)
+{
+
+ if (mem_cgroup_workqueue == NULL) {
+  BUG_ON(mem->css.cgroup->parent != NULL);
+  return;
+ }
+
+ if (work_pending(&mem->reclaim_work))
+  return;
+
+ if (!mem_cgroup_need_reclaim(mem))
+  return;
+
+ css_get(&mem->css); /* XXX need some thoughts wrt cgroup removal. */
+ /*
+  * XXX workqueue is not an ideal mechanism for our purpose.
+  * revisit later.
+  */
+ if (!queue_work(mem_cgroup_workqueue, &mem->reclaim_work))
+  css_put(&mem->css);
```

```
+}
+
+static void
+mem_cgroup_reclaim(struct work_struct *work)
+{
+ struct mem_cgroup * const mem =
+     container_of(work, struct mem_cgroup, reclaim_work);
+ int batch_count = 128; /* XXX arbitrary */
+
+ for (; batch_count > 0; batch_count--) {
+  if (!mem_cgroup_need_reclaim(mem))
+   break;
+  /*
+   * XXX try_to_free_foo is not a correct mechanism to
+   * use here.  eg. ALLOCSTALL counter
+   * revisit later.
+   */
+  if (!try_to_free_mem_cgroup_pages(mem, GFP_KERNEL))
+   break;
+ }
+ if (batch_count == 0)
+  mem_cgroup_schedule_reclaim_if_necessary(mem);
+ css_put(&mem->css);
+}
+
 /*
  * Charge the memory controller for page usage.
  * Return
@@ -311,6 +380,9 @@ int mem_cgroup_charge(struct page *page,
   */
  while (res_counter_charge(&mem->res, PAGE_SIZE)) {
   bool is_atomic = gfp_mask & GFP_ATOMIC;
+
+  mem_cgroup_schedule_reclaim_if_necessary(mem);
+
   /*
    * We cannot reclaim under GFP_ATOMIC, fail the charge
    */
@@ -551,8 +623,18 @@ mem_cgroup_create(struct cgroup_subsys *
  if (unlikely((cont->parent) == NULL)) {
   mem = &init_mem_cgroup;
   init_mm.mem_cgroup = mem;
- } else
+ } else {
+  /* XXX too late for the top-level cgroup */
+  if (mem_cgroup_workqueue == NULL) {
+   mutex_lock(&mem_cgroup_workqueue_init_lock);
+   if (mem_cgroup_workqueue == NULL) {
```

```
+    mem_cgroup_workqueue =
+        create_workqueue("mem_cgroup");
+   }
+   mutex_unlock(&mem_cgroup_workqueue_init_lock);
+  }
   mem = kzalloc(sizeof(struct mem_cgroup), GFP_KERNEL);
+ }

  if (mem == NULL)
   return NULL;
@@ -562,6 +644,7 @@ mem_cgroup_create(struct cgroup_subsys *
  INIT_LIST_HEAD(&mem->inactive_list);
  spin_lock_init(&mem->lru_lock);
  mem->control_type = MEM_CGROUP_TYPE_ALL;
+ INIT_WORK(&mem->reclaim_work, mem_cgroup_reclaim);
  return &mem->css;
 }
```

_____

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