Subject: Re: [RFC] [PATCH] memory controller background reclamation
Posted by yamamoto on Mon, 26 Nov 2007 23:46:19 GMT
View Forum Message <> Reply to Message

> Using numbers like 128 make the code unreadable. I prefer something
> like MEM_CGROUP_BATCH_COUNT since its more readable than 128. If we ever
> propagate batch_count to other dependent functions, I'd much rather do
> it with a well defined name.

here's a new version.

changes from the previous:
- define a constant. (MEM_CGROUP_BG_RECLAIM_BATCH_COUNT)
  no functional changes.
- don't increment ALLOCSTALL vm event for mem cgroup because
  it isn't appropriate esp. for background reclamation.
  introduce MEM_CGROUP_STAT_ALLOCSTALL instead.
  (its value is same as memory.failcnt unless GFP_ATOMIC.)

YAMAMOTO Takashi

Signed-off-by: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

```
--- linux-2.6.24-rc2-mm1-kame-pd/include/linux/res_counter.h.BACKUP 2007-11-14
16:05:48.000000000 +0900
+++ linux-2.6.24-rc2-mm1-kame-pd/include/linux/res_counter.h 2007-11-22 15:14:32.000000000
+0900
@@ -32,6 +32,13 @@ struct res_counter {
   * the number of unsuccessful attempts to consume the resource
   */
  unsigned long long failcnt;
+
+ /*
+  * watermarks
+  */
+ unsigned long long high_watermark;
+ unsigned long long low_watermark;
+
  /*
   * the lock to protect all of the above.
   * the routines below consider this to be IRQ-safe
@@ -66,6 +73,8 @@ enum {
  RES_USAGE,
  RES_LIMIT,
  RES_FAILCNT,
+ RES_HIGH_WATERMARK,
```

```
+ RES_LOW_WATERMARK,
 };

 /*
@@ -124,4 +133,26 @@ static inline bool res_counter_check_und
  return ret;
 }

+static inline bool res_counter_below_low_watermark(struct res_counter *cnt)
+{
+ bool ret;
+ unsigned long flags;
+
+ spin_lock_irqsave(&cnt->lock, flags);
+ ret = cnt->usage < cnt->low_watermark;
+ spin_unlock_irqrestore(&cnt->lock, flags);
+ return ret;
+}
+
+static inline bool res_counter_above_high_watermark(struct res_counter *cnt)
+{
+ bool ret;
+ unsigned long flags;
+
+ spin_lock_irqsave(&cnt->lock, flags);
+ ret = cnt->usage > cnt->high_watermark;
+ spin_unlock_irqrestore(&cnt->lock, flags);
+ return ret;
+}
+
 #endif
--- linux-2.6.24-rc2-mm1-kame-pd/kernel/res_counter.c.BACKUP 2007-11-14
16:05:52.000000000 +0900
+++ linux-2.6.24-rc2-mm1-kame-pd/kernel/res_counter.c 2007-11-22 15:14:32.000000000 +0900
@@ -17,6 +17,8 @@ void res_counter_init(struct res_counter
 {
  spin_lock_init(&counter->lock);
  counter->limit = (unsigned long long)LLONG_MAX;
+ counter->high_watermark = (unsigned long long)LLONG_MAX;
+ counter->low_watermark = (unsigned long long)LLONG_MAX;
 }

 int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
@@ -69,6 +71,10 @@ res_counter_member(struct res_counter *c
   return &counter->limit;
  case RES_FAILCNT:
   return &counter->failcnt;
+ case RES_HIGH_WATERMARK:
```

```
+  return &counter->high_watermark;
+ case RES_LOW_WATERMARK:
+  return &counter->low_watermark;
  };

  BUG();
@@ -99,6 +105,7 @@ ssize_t res_counter_write(struct res_cou
  int ret;
  char *buf, *end;
  unsigned long long tmp, *val;
+ unsigned long flags;

  buf = kmalloc(nbytes + 1, GFP_KERNEL);
  ret = -ENOMEM;
@@ -122,9 +129,29 @@ ssize_t res_counter_write(struct res_cou
    goto out_free;
  }

+ spin_lock_irqsave(&counter->lock, flags);
  val = res_counter_member(counter, member);
+ /* ensure low_watermark <= high_watermark <= limit */
+ switch (member) {
+ case RES_LIMIT:
+  if (tmp < counter->high_watermark)
+   goto out_locked;
+  break;
+ case RES_HIGH_WATERMARK:
+  if (tmp > counter->limit || tmp < counter->low_watermark)
+   goto out_locked;
+  break;
+ case RES_LOW_WATERMARK:
+  if (tmp > counter->high_watermark)
+   goto out_locked;
+  break;
+ }
  *val = tmp;
+ BUG_ON(counter->high_watermark > counter->limit);
+ BUG_ON(counter->low_watermark > counter->high_watermark);
  ret = nbytes;
+out_locked:
+ spin_unlock_irqrestore(&counter->lock, flags);
 out_free:
  kfree(buf);
 out:
--- linux-2.6.24-rc2-mm1-kame-pd/mm/vmscan.c.BACKUP 2007-11-20 13:11:09.000000000
+0900
+++ linux-2.6.24-rc2-mm1-kame-pd/mm/vmscan.c 2007-11-27 08:09:51.000000000 +0900
@@ -1333,7 +1333,6 @@ static unsigned long do_try_to_free_page
```

```
  unsigned long lru_pages = 0;
  int i;

- count_vm_event(ALLOCSTALL);
  /*
   * mem_cgroup will not do shrink_slab.
   */
@@ -1432,6 +1431,7 @@ unsigned long try_to_free_pages(struct z
   .isolate_pages = isolate_pages_global,
  };

+ count_vm_event(ALLOCSTALL);
  return do_try_to_free_pages(zones, gfp_mask, &sc);
 }

--- linux-2.6.24-rc2-mm1-kame-pd/mm/memcontrol.c.BACKUP 2007-11-20 13:11:09.000000000
+0900
+++ linux-2.6.24-rc2-mm1-kame-pd/mm/memcontrol.c 2007-11-27 08:27:10.000000000 +0900
@@ -28,6 +28,7 @@
 #include <linux/rcupdate.h>
 #include <linux/swap.h>
 #include <linux/spinlock.h>
+#include <linux/workqueue.h>
 #include <linux/fs.h>
 #include <linux/seq_file.h>

@@ -35,6 +36,7 @@

 struct cgroup_subsys mem_cgroup_subsys;
 static const int MEM_CGROUP_RECLAIM_RETRIES = 5;
+static const int MEM_CGROUP_BG_RECLAIM_BATCH_COUNT = 128; /* XXX arbitrary */

 /*
  * Statistics for memory cgroup.
@@ -45,6 +47,7 @@ enum mem_cgroup_stat_index {
   */
  MEM_CGROUP_STAT_CACHE,     /* # of pages charged as cache */
  MEM_CGROUP_STAT_RSS,    /* # of pages charged as rss */
+ MEM_CGROUP_STAT_ALLOCSTALL,/* allocation stalled due to memory.limit */

  MEM_CGROUP_STAT_NSTATS,
 };
@@ -138,6 +141,10 @@ struct mem_cgroup {
   * statistics.
   */
  struct mem_cgroup_stat stat;
+ /*
+  * background reclamation.
```

```
+  */
+ struct work_struct reclaim_work;
 };

 /*
@@ -240,6 +247,21 @@ static unsigned long mem_cgroup_get_all_

 static struct mem_cgroup init_mem_cgroup;

+static DEFINE_MUTEX(mem_cgroup_workqueue_init_lock);
+static struct workqueue_struct *mem_cgroup_workqueue;
+
+static void mem_cgroup_create_workqueue(void)
+{
+
+ if (mem_cgroup_workqueue != NULL)
+  return;
+
+ 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);
+}
+
 static inline
 struct mem_cgroup *mem_cgroup_from_cont(struct cgroup *cont)
 {
@@ -566,6 +588,45 @@ unsigned long mem_cgroup_isolate_pages(u
  return nr_taken;
 }

+static void
+mem_cgroup_schedule_reclaim(struct mem_cgroup *mem)
+{
+
+ if (mem_cgroup_workqueue == NULL) {
+  BUG_ON(mem->css.cgroup->parent != NULL);
+  return;
+ }
+
+ if (work_pending(&mem->reclaim_work))
+  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 = MEM_CGROUP_BG_RECLAIM_BATCH_COUNT;
+
+ for (; batch_count > 0; batch_count--) {
+  if (res_counter_below_low_watermark(&mem->res))
+   break;
+  if (!try_to_free_mem_cgroup_pages(mem, GFP_KERNEL))
+   break;
+ }
+ if (batch_count == 0)
+  mem_cgroup_schedule_reclaim(mem);
+ css_put(&mem->css);
+}
+
 /*
  * Charge the memory controller for page usage.
  * Return
@@ -631,6 +692,12 @@ retry:
  rcu_read_unlock();

  /*
+  * schedule background reclaim if we are above the high watermark.
+  */
+ if (res_counter_above_high_watermark(&mem->res))
+  mem_cgroup_schedule_reclaim(mem);
+
+ /*
   * If we created the page_cgroup, we should free it on exceeding
   * the cgroup limit.
   */
@@ -642,6 +709,10 @@ retry:
   if (is_atomic)
    goto noreclaim;

+  preempt_disable();
+  __mem_cgroup_stat_add_safe(&mem->stat,
+      MEM_CGROUP_STAT_ALLOCSTALL, 1);
+  preempt_enable();
   if (try_to_free_mem_cgroup_pages(mem, gfp_mask))
    continue;
```

```
@@ -939,9 +1010,16 @@ static ssize_t mem_cgroup_write(struct c
     struct file *file, const char __user *userbuf,
     size_t nbytes, loff_t *ppos)
 {
- return res_counter_write(&mem_cgroup_from_cont(cont)->res,
+ ssize_t ret;
+
+ ret = res_counter_write(&mem_cgroup_from_cont(cont)->res,
     cft->private, userbuf, nbytes, ppos,
     mem_cgroup_write_strategy);
+
+ if (ret >= 0 && cft->private == RES_HIGH_WATERMARK)
+  mem_cgroup_create_workqueue();
+
+ return ret;
 }

 static ssize_t mem_control_type_write(struct cgroup *cont,
@@ -1031,6 +1109,7 @@ static const struct mem_cgroup_stat_desc
 } mem_cgroup_stat_desc[] = {
  [MEM_CGROUP_STAT_CACHE] = { "cache", PAGE_SIZE, },
  [MEM_CGROUP_STAT_RSS] = { "rss", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_ALLOCSTALL] = { "allocstall", 1, },
 };

 static int mem_control_stat_show(struct seq_file *m, void *arg)
@@ -1097,6 +1176,18 @@ static struct cftype mem_cgroup_files[]
   .read = mem_cgroup_read,
  },
  {
+  .name = "high_watermark_in_bytes",
+  .private = RES_HIGH_WATERMARK,
+  .write = mem_cgroup_write,
+  .read = mem_cgroup_read,
+ },
+ {
+  .name = "low_watermark_in_bytes",
+  .private = RES_LOW_WATERMARK,
+  .write = mem_cgroup_write,
+  .read = mem_cgroup_read,
+ },
+ {
   .name = "control_type",
   .write = mem_control_type_write,
   .read = mem_control_type_read,
@@ -1161,6 +1252,8 @@ mem_cgroup_create(struct cgroup_subsys *
   if (alloc_mem_cgroup_per_zone_info(mem, node))
```

```
    goto free_out;

+ INIT_WORK(&mem->reclaim_work, mem_cgroup_reclaim);
+
  return &mem->css;
 free_out:
  for_each_node_state(node, N_POSSIBLE)
```

_____

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