Subject: [RFC] [PATCH] memory controller statistics
Posted by yamamoto on Fri, 07 Sep 2007 03:39:42 GMT
View Forum Message <> Reply to Message

hi,

i implemented some statistics for your memory controller.

it's tested with 2.6.23-rc2-mm2 + memory controller v7.
i think it can be applied to 2.6.23-rc4-mm1 as well.

YAMOMOTO Takshi

todo: something like nr_active/inactive in /proc/vmstat.

```
--- ./mm/memcontrol.c.BACKUP 2007-08-29 17:13:09.000000000 +0900
+++ ./mm/memcontrol.c 2007-09-06 16:26:13.000000000 +0900
@@ -24,6 +24,7 @@
 #include <linux/page-flags.h>
 #include <linux/bit_spinlock.h>
 #include <linux/rcupdate.h>
+#include <linux/seq_file.h>
 #include <linux/swap.h>
 #include <linux/spinlock.h>
 #include <linux/fs.h>
@@ -33,6 +34,54 @@
 struct container_subsys mem_container_subsys;
 static const int MEM_CONTAINER_RECLAIM_RETRIES = 5;

+enum mem_container_stat_index {
+ /*
+  * for MEM_CONTAINER_TYPE_ALL, usage == pagecache + rss
+  */
+ MEMCONT_STAT_PAGECACHE,
+ MEMCONT_STAT_RSS,
+
+ /*
+  * redundant; usage == charge - uncharge
+  */
+ MEMCONT_STAT_CHARGE,
+ MEMCONT_STAT_UNCHARGE,
+
+ /*
+  * mostly for debug
+  */
+ MEMCONT_STAT_ISOLATE,
+ MEMCONT_STAT_ISOLATE_FAIL,
+ MEMCONT_STAT_NSTATS,
```

```
+};
+
+static const char * const mem_container_stat_desc[] = {
+ [MEMCONT_STAT_PAGECACHE] = "page_cache",
+ [MEMCONT_STAT_RSS] = "rss",
+ [MEMCONT_STAT_CHARGE] = "charge",
+ [MEMCONT_STAT_UNCHARGE] = "uncharge",
+ [MEMCONT_STAT_ISOLATE] = "isolate",
+ [MEMCONT_STAT_ISOLATE_FAIL] = "isolate_fail",
+};
+
+struct mem_container_stat {
+ atomic_t count[MEMCONT_STAT_NSTATS];
+};
+
+static void mem_container_stat_inc(struct mem_container_stat * stat,
+   enum mem_container_stat_index idx)
+{
+
+ atomic_inc(&stat->count[idx]);
+}
+
+static void mem_container_stat_dec(struct mem_container_stat * stat,
+   enum mem_container_stat_index idx)
+{
+
+ atomic_dec(&stat->count[idx]);
+}
+
 /*
  * The memory controller data structure. The memory controller controls both
  * page cache and RSS per container. We would eventually like to provide
@@ -62,6 +111,7 @@ struct mem_container {
   */
  spinlock_t lru_lock;
  unsigned long control_type; /* control RSS or RSS+Pagecache */
+ struct mem_container_stat stat;
 };

 /*
@@ -72,6 +122,12 @@ struct mem_container {
 #define PAGE_CONTAINER_LOCK_BIT  0x0
 #define PAGE_CONTAINER_LOCK   (1 << PAGE_CONTAINER_LOCK_BIT)

+/* XXX hack; shouldn't be here.  it really belongs to struct page_container. */
+#define PAGE_CONTAINER_CACHE_BIT  0x1
+#define PAGE_CONTAINER_CACHE   (1 << PAGE_CONTAINER_CACHE_BIT)
+
```

```
+#define PAGE_CONTAINER_FLAGS (PAGE_CONTAINER_LOCK |
PAGE_CONTAINER_CACHE)
+
 /*
  * A page_container page is associated with every page descriptor. The
  * page_container helps us identify information about the container
@@ -134,9 +190,9 @@ static inline int page_container_locked(
     &page->page_container);
 }

-void page_assign_page_container(struct page *page, struct page_container *pc)
+static void page_assign_page_container_flags(struct page *page, int flags,
+    struct page_container *pc)
 {
- int locked;

  /*
   * While resetting the page_container we might not hold the
@@ -145,14 +201,20 @@ void page_assign_page_container(struct p
   */
  if (pc)
   VM_BUG_ON(!page_container_locked(page));
- locked = (page->page_container & PAGE_CONTAINER_LOCK);
- page->page_container = ((unsigned long)pc | locked);
+ flags |= (page->page_container & PAGE_CONTAINER_LOCK);
+ page->page_container = ((unsigned long)pc | flags);
+}
+
+void page_assign_page_container(struct page *page, struct page_container *pc)
+{
+
+ page_assign_page_container_flags(page, 0, pc);
 }

 struct page_container *page_get_page_container(struct page *page)
 {
  return (struct page_container *)
-  (page->page_container & ~PAGE_CONTAINER_LOCK);
+  (page->page_container & ~PAGE_CONTAINER_FLAGS);
 }

 void __always_inline lock_page_container(struct page *page)
@@ -203,6 +265,7 @@ unsigned long mem_container_isolate_page
  LIST_HEAD(pc_list);
  struct list_head *src;
  struct page_container *pc;
+ struct mem_container_stat *stat = &mem_cont->stat;
```

```
  if (active)
   src = &mem_cont->active_list;
@@ -244,6 +307,9 @@ unsigned long mem_container_isolate_page
   if (__isolate_lru_page(page, mode) == 0) {
    list_move(&page->lru, dst);
    nr_taken++;
+   mem_container_stat_inc(stat, MEMCONT_STAT_ISOLATE);
+  } else {
+   mem_container_stat_inc(stat, MEMCONT_STAT_ISOLATE_FAIL);
   }
  }

@@ -260,9 +326,11 @@ unsigned long mem_container_isolate_page
  * 0 if the charge was successful
  * < 0 if the container is over its limit
  */
-int mem_container_charge(struct page *page, struct mm_struct *mm)
+static int mem_container_charge_common(struct page *page, struct mm_struct *mm,
+    int is_cache)
 {
  struct mem_container *mem;
+ struct mem_container_stat *stat;
  struct page_container *pc, *race_pc;
  unsigned long flags;
  unsigned long nr_retries = MEM_CONTAINER_RECLAIM_RETRIES;
@@ -360,7 +428,16 @@ int mem_container_charge(struct page *pa
  atomic_set(&pc->ref_cnt, 1);
  pc->mem_container = mem;
  pc->page = page;
- page_assign_page_container(page, pc);
+ page_assign_page_container_flags(page,
+     is_cache ? PAGE_CONTAINER_CACHE : 0, pc);
+
+ stat = &mem->stat;
+ if (is_cache) {
+  mem_container_stat_inc(stat, MEMCONT_STAT_PAGECACHE);
+ } else {
+  mem_container_stat_inc(stat, MEMCONT_STAT_RSS);
+ }
+ mem_container_stat_inc(stat, MEMCONT_STAT_CHARGE);

  spin_lock_irqsave(&mem->lru_lock, flags);
  list_add(&pc->lru, &mem->active_list);
@@ -377,6 +454,12 @@ err:
  return -ENOMEM;
 }

+int mem_container_charge(struct page *page, struct mm_struct *mm)
```

```
+{
+
+ return mem_container_charge_common(page, mm, 0);
+}
+
 /*
  * See if the cached pages should be charged at all?
  */
@@ -388,7 +471,7 @@ int mem_container_cache_charge(struct pa

  mem = rcu_dereference(mm->mem_container);
  if (mem->control_type == MEM_CONTAINER_TYPE_ALL)
-  return mem_container_charge(page, mm);
+  return mem_container_charge_common(page, mm, 1);
  else
   return 0;
 }
@@ -411,15 +494,29 @@ void mem_container_uncharge(struct page_
   return;

  if (atomic_dec_and_test(&pc->ref_cnt)) {
+  struct mem_container_stat *stat;
+  int is_cache;
+
   page = pc->page;
   lock_page_container(page);
   mem = pc->mem_container;
   css_put(&mem->css);
+  /* XXX */
+  is_cache = (page->page_container & PAGE_CONTAINER_CACHE) != 0;
   page_assign_page_container(page, NULL);
   unlock_page_container(page);
   res_counter_uncharge(&mem->res, 1);

+  stat = &mem->stat;
+  if (is_cache) {
+   mem_container_stat_dec(stat, MEMCONT_STAT_PAGECACHE);
+  } else {
+   mem_container_stat_dec(stat, MEMCONT_STAT_RSS);
+  }
+  mem_container_stat_inc(stat, MEMCONT_STAT_UNCHARGE);
+
   spin_lock_irqsave(&mem->lru_lock, flags);
+  BUG_ON(list_empty(&pc->lru));
   list_del_init(&pc->lru);
   spin_unlock_irqrestore(&mem->lru_lock, flags);
   kfree(pc);
@@ -496,6 +593,44 @@ static ssize_t mem_control_type_read(str
```

```
    ppos, buf, s - buf);
 }

+static void mem_container_stat_init(struct mem_container_stat *stat)
+{
+ int i;
+
+ for (i = 0; i < ARRAY_SIZE(stat->count); i++) {
+  atomic_set(&stat->count[i], 0);
+ }
+}
+
+static int mem_control_stat_show(struct seq_file *m, void *arg)
+{
+ struct container *cont = m->private;
+ struct mem_container *mem_cont = mem_container_from_cont(cont);
+ struct mem_container_stat *stat = &mem_cont->stat;
+ int i;
+
+ for (i = 0; i < ARRAY_SIZE(stat->count); i++) {
+  seq_printf(m, "%s %u\n", mem_container_stat_desc[i],
+       (unsigned int)atomic_read(&stat->count[i]));
+ }
+ return 0;
+}
+
+static const struct file_operations mem_control_stat_file_operations = {
+ .read = seq_read,
+ .llseek = seq_lseek,
+ .release = single_release,
+};
+
+static int mem_control_stat_open(struct inode *unused, struct file *file)
+{
+ /* XXX __d_cont */
+ struct container *cont = file->f_dentry->d_parent->d_fsdata;
+
+ file->f_op = &mem_control_stat_file_operations;
+ return single_open(file, mem_control_stat_show, cont);
+}
+
 static struct cftype mem_container_files[] = {
  {
   .name = "usage",
@@ -518,6 +653,10 @@ static struct cftype mem_container_files
   .write = mem_control_type_write,
   .read = mem_control_type_read,
  },
```

```
+ {
+  .name = "stat",
+  .open = mem_control_stat_open,
+ },
 };

 static struct mem_container init_mem_container;
@@ -541,6 +680,7 @@ mem_container_create(struct container_su
  INIT_LIST_HEAD(&mem->inactive_list);
  spin_lock_init(&mem->lru_lock);
  mem->control_type = MEM_CONTAINER_TYPE_ALL;
+ mem_container_stat_init(&mem->stat);
  return &mem->css;
 }
```

_____

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