Subject: Re: [RFC] [PATCH] memory controller statistics
Posted by yamamoto on Wed, 10 Oct 2007 08:19:23 GMT
View Forum Message <> Reply to Message

> On Wed, 10 Oct 2007 10:01:17 +0900 (JST)
> yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
>
> > hi,
> >
> > > > i implemented some statistics for your memory controller.
> >
> > here's a new version.
> >
> > changes from the previous:
> > - make counters per-cpu.
> > - value *= PAGE_SIZE
> >
>
> YAMAMOTO-san, I like this work and the patch seems good.
> But will HUNK with my work to some extent ;)
>
> So, I'd like to merge this patch on my patch set (against -mm) if you don't mind.
> If it's ok, please give your "Signed-off-by" line. Then, I'll merge this to mine.
>
> Thanks,
> -Kame

sure. here's the latest version.

changes from the previous:
 - fix a race in uncharge. (check PCGF_ACTIVE with mem->lru_lock held)

YAMAMOTO Takashi

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

--- linux-2.6.23-rc8-mm2-stat/mm/memcontrol.c.BACKUP 2007-10-01 17:19:57.000000000 +0900
+++ linux-2.6.23-rc8-mm2-stat/mm/memcontrol.c 2007-10-10 12:40:48.000000000 +0900
@@ -25,6 +25,7 @@
 #include <linux/backing-dev.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>

```
@@ -34,6 +35,63 @@
 struct cgroup_subsys mem_cgroup_subsys;
 static const int MEM_CGROUP_RECLAIM_RETRIES = 5;

+enum mem_cgroup_stat_index {
+ /*
+  * for MEM_CONTAINER_TYPE_ALL, usage == pagecache + rss
+  */
+ MEM_CGROUP_STAT_PAGECACHE, /* # of pages charged as cache */
+ MEM_CGROUP_STAT_RSS,  /* # of pages charged as rss */
+
+ /*
+  * redundant; usage == charge - uncharge
+  */
+ MEM_CGROUP_STAT_CHARGE,  /* # of pages charged */
+ MEM_CGROUP_STAT_UNCHARGE, /* # of pages uncharged */
+
+ MEM_CGROUP_STAT_ACTIVE,  /* # of pages on active_list */
+ MEM_CGROUP_STAT_INACTIVE, /* # of pages on inactive_list */
+
+ MEM_CGROUP_STAT_NSTATS,
+};
+
+static const struct mem_cgroup_stat_desc {
+ const char *msg;
+ u64 unit;
+} mem_cgroup_stat_desc[] = {
+ [MEM_CGROUP_STAT_PAGECACHE] = { "page_cache", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_RSS] = { "rss", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_CHARGE] = { "charge", PAGE_SIZE },
+ [MEM_CGROUP_STAT_UNCHARGE] = { "uncharge", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_ACTIVE] = { "active", PAGE_SIZE, },
+ [MEM_CGROUP_STAT_INACTIVE] = { "inactive", PAGE_SIZE, },
+};
+
+struct mem_cgroup_stat_cpu {
+ u64 count[MEM_CGROUP_STAT_NSTATS];
+} ____cacheline_aligned_in_smp;
+
+struct mem_cgroup_stat {
+ struct mem_cgroup_stat_cpu cpustat[NR_CPUS];
+};
+
+static inline void mem_cgroup_stat_inc(struct mem_cgroup_stat * stat,
+   enum mem_cgroup_stat_index idx)
+{
+ unsigned int cpu = get_cpu();
+
```

```
+ stat->cpustat[cpu].count[idx]++;
+ put_cpu();
+}
+
+static inline void mem_cgroup_stat_dec(struct mem_cgroup_stat * stat,
+   enum mem_cgroup_stat_index idx)
+{
+ unsigned int cpu = get_cpu();
+
+ stat->cpustat[cpu].count[idx]--;
+ put_cpu();
+}
+
 /*
  * The memory controller data structure. The memory controller controls both
  * page cache and RSS per cgroup. We would eventually like to provide
@@ -63,6 +121,11 @@ struct mem_cgroup {
   */
  spinlock_t lru_lock;
  unsigned long control_type; /* control RSS or RSS+Pagecache */
+
+ /*
+  * statistics
+  */
+ struct mem_cgroup_stat stat;
 };

 /*
@@ -83,6 +146,9 @@ struct page_cgroup {
  struct mem_cgroup *mem_cgroup;
  atomic_t ref_cnt;  /* Helpful when pages move b/w  */
      /* mapped and cached states     */
+ int flags;
+#define PCGF_PAGECACHE 1 /* charged as page cache */
+#define PCGF_ACTIVE 2 /* on active_list */
 };

 enum {
@@ -164,10 +230,24 @@ static void __always_inline unlock_page_

 static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
- if (active)
-  list_move(&pc->lru, &pc->mem_cgroup->active_list);
- else
-  list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
+ struct mem_cgroup *mem = pc->mem_cgroup;
+ struct mem_cgroup_stat *stat = &mem->stat;
```

```
+
+ if (active) {
+  list_move(&pc->lru, &mem->active_list);
+  if ((pc->flags & PCGF_ACTIVE) == 0) {
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_INACTIVE);
+   mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_ACTIVE);
+   pc->flags |= PCGF_ACTIVE;
+  }
+ } else {
+  list_move(&pc->lru, &mem->inactive_list);
+  if ((pc->flags & PCGF_ACTIVE) != 0) {
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_ACTIVE);
+   mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_INACTIVE);
+   pc->flags &= ~PCGF_ACTIVE;
+  }
+ }
 }

 /*
@@ -256,10 +336,11 @@ unsigned long mem_cgroup_isolate_pages(u
  * 0 if the charge was successful
  * < 0 if the cgroup is over its limit
  */
-int mem_cgroup_charge(struct page *page, struct mm_struct *mm,
-    gfp_t gfp_mask)
+int mem_cgroup_charge_common(struct page *page, struct mm_struct *mm,
+    gfp_t gfp_mask, int is_cache)
 {
  struct mem_cgroup *mem;
+ struct mem_cgroup_stat *stat;
  struct page_cgroup *pc, *race_pc;
  unsigned long flags;
  unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
@@ -365,8 +446,17 @@ noreclaim:
  atomic_set(&pc->ref_cnt, 1);
  pc->mem_cgroup = mem;
  pc->page = page;
+ pc->flags = (is_cache ? PCGF_PAGECACHE : 0) | PCGF_ACTIVE;
  page_assign_page_cgroup(page, pc);

+ stat = &mem->stat;
+ if (is_cache)
+  mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_PAGECACHE);
+ else
+  mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_RSS);
+ mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_CHARGE);
+ mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_ACTIVE);
+
```

```
  spin_lock_irqsave(&mem->lru_lock, flags);
  list_add(&pc->lru, &mem->active_list);
  spin_unlock_irqrestore(&mem->lru_lock, flags);
@@ -382,6 +472,14 @@ err:
  return -ENOMEM;
 }

+int mem_cgroup_charge(struct page *page, struct mm_struct *mm,
+    gfp_t gfp_mask)
+{
+
+ return mem_cgroup_charge_common(page, mm, gfp_mask, 0);
+}
+
+
 /*
  * See if the cached pages should be charged at all?
  */
@@ -394,7 +492,7 @@ int mem_cgroup_cache_charge(struct page

  mem = rcu_dereference(mm->mem_cgroup);
  if (mem->control_type == MEM_CGROUP_TYPE_ALL)
-  return mem_cgroup_charge(page, mm, gfp_mask);
+  return mem_cgroup_charge_common(page, mm, gfp_mask, 1);
  else
   return 0;
 }
@@ -417,17 +515,34 @@ void mem_cgroup_uncharge(struct page_cgr
   return;

  if (atomic_dec_and_test(&pc->ref_cnt)) {
+  struct mem_cgroup_stat *stat;
+  int oflags;
+
   page = pc->page;
   lock_page_cgroup(page);
   mem = pc->mem_cgroup;
-  css_put(&mem->css);
   page_assign_page_cgroup(page, NULL);
   unlock_page_cgroup(page);
   res_counter_uncharge(&mem->res, PAGE_SIZE);

    spin_lock_irqsave(&mem->lru_lock, flags);
+  oflags = pc->flags;
+  BUG_ON(list_empty(&pc->lru));
    list_del_init(&pc->lru);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
+
```

```
+  stat = &mem->stat;
+  if ((oflags & PCGF_ACTIVE) != 0)
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_ACTIVE);
+  else
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_INACTIVE);
+  if ((oflags & PCGF_PAGECACHE) != 0)
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_PAGECACHE);
+  else
+   mem_cgroup_stat_dec(stat, MEM_CGROUP_STAT_RSS);
+  mem_cgroup_stat_inc(stat, MEM_CGROUP_STAT_UNCHARGE);
+
+  css_put(&mem->css);
   kfree(pc);
  }
 }
@@ -517,6 +632,49 @@ static ssize_t mem_control_type_read(str
    ppos, buf, s - buf);
 }

+static void mem_cgroup_stat_init(struct mem_cgroup_stat *stat)
+{
+
+ memset(stat, 0, sizeof(*stat));
+}
+
+static int mem_control_stat_show(struct seq_file *m, void *arg)
+{
+ struct cgroup *cont = m->private;
+ struct mem_cgroup *mem_cont = mem_cgroup_from_cont(cont);
+ struct mem_cgroup_stat *stat = &mem_cont->stat;
+ int i;
+
+ for (i = 0; i < ARRAY_SIZE(stat->cpustat[0].count); i++) {
+  unsigned int cpu;
+  u64 val;
+
+  val = 0;
+  for (cpu = 0; cpu < NR_CPUS; cpu++)
+   val += stat->cpustat[cpu].count[i];
+
+  seq_printf(m, "%s %llu\n", mem_cgroup_stat_desc[i].msg,
+      (unsigned long long)(val * mem_cgroup_stat_desc[i].unit));
+ }
+
+ 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 cgroup *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_cgroup_files[] = {
  {
   .name = "usage_in_bytes",
@@ -539,6 +697,10 @@ static struct cftype mem_cgroup_files[]
   .write = mem_control_type_write,
   .read = mem_control_type_read,
  },
+ {
+  .name = "stat",
+  .open = mem_control_stat_open,
+ },
 };

 static struct mem_cgroup init_mem_cgroup;
@@ -562,6 +724,7 @@ 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;
+ mem_cgroup_stat_init(&mem->stat);
  return &mem->css;
 }
```