Subject: [PATCH] memory cgroup enhancements take 4 [5/8] add status accounting function for memory cgroup
Posted by KAMEZAWA Hiroyuki on Wed, 31 Oct 2007 10:30:46 GMT
View Forum Message <> Reply to Message

Add statistics account infrastructure for memory controller.
All account information is stored per-cpu and caller will not have
to take lock or use atomic ops.
This will be used by memory.stat file later.

CACHE includes swapcache now. I'd like to divide it to
PAGECACHE and SWAPCACHE later.

This patch adds 3 functions for accounting.
 * __mem_cgroup_stat_add() ... for usual routine.
 * __mem_cgroup_stat_add_safe ... for calling under irq_disabled section.
 * mem_cgroup_read_stat() ... for reading stat value.
 * renamed PAGECACHE to CACHE (because it may include swapcache *now*)

Changelog v3 -> v4
 - fixed typo.
 - removed unused inc/dec interface.
 - added __mem_cgroup_stat_add_safe() for accounting under safe situation.
 - moved callers of mem_cgroup_charge_statistics() under irq disabled section.
 - added mem_cgroup_read_stat()

Changelog v2 -> v3
 - adjusted to rename of #define PAGE_CGROUP_FLAG....
 - dropped ACTIVE/INACTIVE counter.
   They should be accounted against per zone. Then, using pcp counter,
   we need array of NR_CPU * MAX_NUMNODES * NR_ZONES against all stats.
   This is too big for statistics *per-memory-cgroup*.
   ACTIVE/INACTIVE counter is added as per-zone statistics later.

Changelog v1 -> v2
 - Removed Charge/Uncharge counter
 - reflected comments.
   - changes __move_lists() args.
   - changes __mem_cgroup_stat_add() name, comment and added VM_BUGON

Changes from original:
 - divided into 2 patch (account and show info)
 - changed from u64 to s64
 - added mem_cgroup_stat_add() and batched statistics modification logic.
 - removed stat init code because mem_cgroup is allocated by kzalloc().

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

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

```
 mm/memcontrol.c |   82
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 1 file changed, 82 insertions(+)

Index: devel-2.6.23-mm1/mm/memcontrol.c
===================================================================
--- devel-2.6.23-mm1.orig/mm/memcontrol.c
+++ devel-2.6.23-mm1/mm/memcontrol.c
@@ -35,6 +35,59 @@ struct cgroup_subsys mem_cgroup_subsys;
 static const int MEM_CGROUP_RECLAIM_RETRIES = 5;

 /*
+ * Statistics for memory cgroup.
+ */
+enum mem_cgroup_stat_index {
+ /*
+  * For MEM_CONTAINER_TYPE_ALL, usage = pagecache + rss.
+  */
+ MEM_CGROUP_STAT_CACHE,     /* # of pages charged as cache */
+ MEM_CGROUP_STAT_RSS,    /* # of pages charged as rss */
+
+ MEM_CGROUP_STAT_NSTATS,
+};
+
+struct mem_cgroup_stat_cpu {
+ s64 count[MEM_CGROUP_STAT_NSTATS];
+} ____cacheline_aligned_in_smp;
+
+struct mem_cgroup_stat {
+ struct mem_cgroup_stat_cpu cpustat[NR_CPUS];
+};
+
+/*
+ * modifies value with disabling preempt.
+ */
+static inline void __mem_cgroup_stat_add(struct mem_cgroup_stat *stat,
+                enum mem_cgroup_stat_index idx, int val)
+{
+ int cpu = smp_processor_id();
+ preempt_disable();
+ stat->cpustat[cpu].count[idx] += val;
+ preempt_enable();
+}
+
+/*
+ * For accounting under irq disable, no need for increment preempt count.
```

```
+ */
+static inline void __mem_cgroup_stat_add_safe(struct mem_cgroup_stat *stat,
+  enum mem_cgroup_stat_index idx, int val)
+{
+ int cpu = smp_processor_id();
+ stat->cpustat[cpu].count[idx] += val;
+}
+
+static inline s64 mem_cgroup_read_stat(struct mem_cgroup_stat *stat,
+  enum mem_cgroup_stat_index idx)
+{
+ int cpu;
+ s64 ret = 0;
+ for_each_possible_cpu(cpu)
+  ret += stat->cpustat[cpu].count[idx];
+ return ret;
+}
+
+/*
  * The memory controller data structure. The memory controller controls both
  * page cache and RSS per cgroup. We would eventually like to provide
  * statistics based on the statistics developed by Rik Van Riel for clock-pro,
@@ -63,6 +116,10 @@ struct mem_cgroup {
   */
  spinlock_t lru_lock;
  unsigned long control_type; /* control RSS or RSS+Pagecache */
+ /*
+  * statistics.
+  */
+ struct mem_cgroup_stat stat;
 };

 /*
@@ -101,6 +158,27 @@ enum charge_type {
  MEM_CGROUP_CHARGE_TYPE_MAPPED,
 };

+/*
+ * Always modified under lru lock. Then, not necessary to preempt_disable()
+ */
+static inline void
+mem_cgroup_charge_statistics(struct mem_cgroup *mem, int flags, bool charge)
+{
+ int val = (charge)? 1 : -1;
+ struct mem_cgroup_stat *stat = &mem->stat;
+ VM_BUG_ON(!irqs_disabled());
+
+ if (flags & PAGE_CGROUP_FLAG_CACHE)
```

```
+  __mem_cgroup_stat_add_safe(stat,
+     MEM_CGROUP_STAT_CACHE, val);
+ else
+  __mem_cgroup_stat_add_safe(stat, MEM_CGROUP_STAT_RSS, val);
+
+}
+
+
+
+
 static struct mem_cgroup init_mem_cgroup;

 static inline
@@ -445,6 +523,8 @@ noreclaim:
  }

  spin_lock_irqsave(&mem->lru_lock, flags);
+ /* Update statistics vector */
+ mem_cgroup_charge_statistics(mem, pc->flags, true);
  list_add(&pc->lru, &mem->active_list);
  spin_unlock_irqrestore(&mem->lru_lock, flags);

@@ -510,6 +590,7 @@ void mem_cgroup_uncharge(struct page_cgr
    res_counter_uncharge(&mem->res, PAGE_SIZE);
    spin_lock_irqsave(&mem->lru_lock, flags);
    list_del_init(&pc->lru);
+   mem_cgroup_charge_statistics(mem, pc->flags, false);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
    kfree(pc);
   }
@@ -586,6 +667,7 @@ retry:
    css_put(&mem->css);
    res_counter_uncharge(&mem->res, PAGE_SIZE);
    list_del_init(&pc->lru);
+   mem_cgroup_charge_statistics(mem, pc->flags, false);
    kfree(pc);
   } else  /* being uncharged ? ...do relax */
```