Subject: [RFC][for -mm] memory cgroup enhancements take3 [9/9] per zone stat
Posted by KAMEZAWA Hiroyuki on Tue, 30 Oct 2007 11:23:32 GMT
View Forum Message <> Reply to Message

Add per-zone x per-cpu counter for accounting # of active/inactive
This array can be big if MAX_NUMNODE is very large, so we need
some cares.

This "active/inactive" information should be maintained per zone
(can be used for page reclaim code later, I think).

Memory cgroup total active/inactive information is shown in memory.stat
file.

This patch changes early_init from 1 to 0 for using kmalloc/vmalloc at boot.

Changelog v1 -> v2:
 - changed from per-node to per-zone.
 - just count acitve/inactive

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

 mm/memcontrol.c | 171
++--
 1 file changed, 165 insertions(+), 6 deletions(-)

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
@@ -29,6 +29,7 @@
 #include <linux/spinlock.h>
 #include <linux/fs.h>
 #include <linux/seq_file.h>
+#include <linux/vmalloc.h>

 #include <asm/uaccess.h>

@@ -56,6 +57,31 @@ struct mem_cgroup_stat {
  struct mem_cgroup_stat_cpu cpustat[NR_CPUS];
 };

+
+/*
+ * Per-zone statistics.
+ * Please be carefull. The array can be very big on envrionments whic has
+ * very big MAX_NUMNODES . Adding new stat member to this will eat much memory.
+ * Only Active/Inactive may be sutiable.

```
+ */
+enum mem_cgroup_zonestat_index {
+ MEM_CGROUP_ZONESTAT_ACTIVE,
+ MEM_CGROUP_ZONESTAT_INACTIVE,
+ MEM_CGROUP_ZONESTAT_NUM,
+};
+
+#ifdef CONFIG_NUMA
+#define PERZONE_ARRAY_SIZE (MAX_NUMNODES *MAX_NR_ZONES)
+#else
+#define PERZONE_ARRAY_SIZE (MAX_NR_ZONES)
+#endif
+struct mem_cgroup_zonestat_cpu {
+ s64 count[PERZONE_ARRAY_SIZE][MEM_CGROUP_ZONESTAT_NUM];
+};
+struct mem_cgroup_zonestat {
+ struct mem_cgroup_zonestat_cpu *cpustat[NR_CPUS];
+};
+
 /*
  * For batching....mem_cgroup_charge_statistics()(see below).
  * MUST be called under preempt_disable().
@@ -86,7 +112,30 @@ static inline void mem_cgroup_stat_dec(s
  preempt_enable();
 }

+static inline void __mem_cgroup_zonestat_add(struct mem_cgroup_zonestat *zstat,
+  enum mem_cgroup_zonestat_index idx, int val, int pos)
+{
+ int cpu = smp_processor_id();
+ zstat->cpustat[cpu]->count[pos][idx] += val;
+}

+static inline void __mem_cgroup_zonesta_dec(struct mem_cgroup_zonestat *zstat,
+  enum mem_cgroup_zonestat_index idx, int val, int pos)
+{
+ int cpu = smp_processor_id();
+ zstat->cpustat[cpu]->count[pos][idx] -= val;
+}
+
+static inline s64 mem_cgroup_count_zonestat(struct mem_cgroup_zonestat *zstat,
+  int nid, int zid, int idx)
+{
+ int cpu;
+ int pos = nid * MAX_NR_ZONES + zid;
+ s64 ret = 0;
+ for_each_possible_cpu(cpu)
+  ret += zstat->cpustat[cpu]->count[pos][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
@@ -120,6 +169,7 @@ struct mem_cgroup {
   * statistics.
   */
  struct mem_cgroup_stat stat;
+ struct mem_cgroup_zonestat zonestat;
 };

 /*
@@ -141,6 +191,8 @@ struct page_cgroup {
  atomic_t ref_cnt;  /* Helpful when pages move b/w  */
      /* mapped and cached states     */
  int  flags;
+ int nid;
+ int zone_id;
 };
 #define PAGE_CGROUP_FLAG_CACHE (0x1) /* charged as cache */
 #define PAGE_CGROUP_FLAG_ACTIVE (0x2) /* page is active in this cgroup */
@@ -158,15 +210,32 @@ enum charge_type {
  MEM_CGROUP_CHARGE_TYPE_MAPPED = 0,
 };

+#ifdef CONFIG_NUMA
+static inline int page_cgroup_to_zonestat_index(struct page_cgroup *pc)
+{
+ return pc->nid * MAX_NR_ZONES + pc->zone_id;
+}
+#else
+static inline int page_cgroup_to_zonestat_index(struct page_cgroup *pc)
+{
+ return pc->znone_id;
+}
+#endif
+
+
 /*
  * Batched statistics modification.
  * We have to modify several values at charge/uncharge..
  */
 static inline void
-mem_cgroup_charge_statistics(struct mem_cgroup *mem, int flags, int charge)
+mem_cgroup_charge_statistics(struct page_cgroup *pc, int charge)
 {
  int val = (charge)? 1 : -1;
```

```
+ struct mem_cgroup *mem = pc->mem_cgroup;
+ int flags = pc->flags;
  struct mem_cgroup_stat *stat = &mem->stat;
+ struct mem_cgroup_zonestat *zonestat = &mem->zonestat;
+ int index = page_cgroup_to_zonestat_index(pc);
  preempt_disable();

  if (flags & PAGE_CGROUP_FLAG_CACHE)
@@ -174,6 +243,13 @@ mem_cgroup_charge_statistics(struct mem_
  else
   __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_RSS, val);

+ if (flags & PAGE_CGROUP_FLAG_ACTIVE)
+  __mem_cgroup_zonestat_add(zonestat, MEM_CGROUP_ZONESTAT_ACTIVE,
+    val, index);
+ else
+  __mem_cgroup_zonestat_add(zonestat,
+    MEM_CGROUP_ZONESTAT_INACTIVE,
+    val, index);
  preempt_enable();
 }

@@ -293,6 +369,23 @@ clear_page_cgroup(struct page *page, str

 static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
+ int direction = 0;
+
+ if (active && !(pc->flags & PAGE_CGROUP_FLAG_ACTIVE))
+  direction = 1; /*from inactive to acvive */
+ if (!active && (pc->flags & PAGE_CGROUP_FLAG_ACTIVE))
+  direction = -1;
+
+ if (direction) {
+  struct mem_cgroup_zonestat *zstat = &pc->mem_cgroup->zonestat;
+  int index = page_cgroup_to_zonestat_index(pc);
+  preempt_disable();
+  __mem_cgroup_zonestat_add(zstat, MEM_CGROUP_ZONESTAT_ACTIVE,
+   direction, index);
+  __mem_cgroup_zonestat_add(zstat, MEM_CGROUP_ZONESTAT_INACTIVE,
+   direction, index);
+ }
+
  if (active) {
   pc->flags |= PAGE_CGROUP_FLAG_ACTIVE;
   list_move(&pc->lru, &pc->mem_cgroup->active_list);
@@ -509,6 +602,8 @@ noreclaim:
  pc->mem_cgroup = mem;
```

```
  pc->page = page;
  pc->flags = PAGE_CGROUP_FLAG_ACTIVE;
+ pc->nid = page_to_nid(page);
+ pc->zone_id = page_zonenum(page);
  if (ctype == MEM_CGROUP_CHARGE_TYPE_CACHE)
   pc->flags |= PAGE_CGROUP_FLAG_CACHE;
  if (page_cgroup_assign_new_page_cgroup(page, pc)) {
@@ -524,7 +619,7 @@ noreclaim:
  }

  /* Update statistics vector */
- mem_cgroup_charge_statistics(mem, pc->flags, true);
+ mem_cgroup_charge_statistics(pc, true);

  spin_lock_irqsave(&mem->lru_lock, flags);
  list_add(&pc->lru, &mem->active_list);
@@ -591,9 +686,10 @@ void mem_cgroup_uncharge(struct page_cgr
    css_put(&mem->css);
    res_counter_uncharge(&mem->res, PAGE_SIZE);
    spin_lock_irqsave(&mem->lru_lock, flags);
+   /* mem is valid while doing this. */
+   mem_cgroup_charge_statistics(pc, false);
    list_del_init(&pc->lru);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
-   mem_cgroup_charge_statistics(mem, pc->flags, false);
    kfree(pc);
   }
  }
@@ -669,7 +765,7 @@ 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);
+   mem_cgroup_charge_statistics(pc, false);
    kfree(pc);
   } else  /* being uncharged ? ...do relax */
    break;
@@ -833,11 +929,20 @@ static const struct mem_cgroup_stat_desc
  [MEM_CGROUP_STAT_RSS] = { "rss", PAGE_SIZE, },
 };

+static const struct mem_cgroup_zstat_desc {
+ const char *msg;
+ u64 unit;
+} mem_cgroup_zstat_desc[] = {
+ [MEM_CGROUP_ZONESTAT_ACTIVE] = {"active", PAGE_SIZE},
+ [MEM_CGROUP_ZONESTAT_INACTIVE] = {"inactive", PAGE_SIZE},
+};
```

```
+
 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;
+ struct mem_cgroup_zonestat *zstat = &mem_cont->zonestat;
  int i;

  for (i = 0; i < ARRAY_SIZE(stat->cpustat[0].count); i++) {
@@ -850,6 +955,16 @@ static int mem_control_stat_show(struct
   val *= mem_cgroup_stat_desc[i].unit;
   seq_printf(m, "%s %lld\n", mem_cgroup_stat_desc[i].msg, val);
  }
+ for (i = 0; i < MEM_CGROUP_ZONESTAT_NUM; i++) {
+  int nid, z;
+  s64 val = 0;
+  for_each_node_state(nid, N_POSSIBLE)
+   for (z = 0; z < MAX_NR_ZONES; z++)
+    val += mem_cgroup_count_zonestat(zstat, nid,
+         z, i);
+  val += mem_cgroup_zstat_desc[i].unit;
+  seq_printf(m, "%s %lld\n", mem_cgroup_zstat_desc[i].msg, val);
+ }
  return 0;
 }

@@ -905,10 +1020,33 @@ static struct cftype mem_cgroup_files[]

 static struct mem_cgroup init_mem_cgroup;

+static struct mem_cgroup_zonestat_cpu *
+__alloc_mem_cgroup_zonestat(int nid)
+{
+ struct mem_cgroup_zonestat_cpu *mczc;
+ if (sizeof(*mczc) < PAGE_SIZE)
+  mczc = kmalloc_node(sizeof(*mczc), GFP_KERNEL, nid);
+ else
+  mczc = vmalloc_node(sizeof(*mczc), nid);
+ return mczc;
+}
+
+static void __free_mem_cgroup_zonestat(struct mem_cgroup_zonestat_cpu *mczc)
+{
+ if (!mczc)
+  return;
+ if (sizeof(*mczc) < PAGE_SIZE)
+  kfree(mczc);
```

```
+ else
+  vfree(mczc);
+ return;
+}
+
 static struct cgroup_subsys_state *
 mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
 {
  struct mem_cgroup *mem;
+ int cpu;

  if (unlikely((cont->parent) == NULL)) {
   mem = &init_mem_cgroup;
@@ -924,7 +1062,23 @@ 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;
+
+ for_each_possible_cpu(cpu) {
+  int nid = cpu_to_node(cpu);
+  struct mem_cgroup_zonestat_cpu *mczc;
+  mczc = __alloc_mem_cgroup_zonestat(nid);
+  if (!mczc)
+   goto free_err;
+  memset(mczc, sizeof(*mczc), 0);
+  mem->zonestat.cpustat[cpu] = mczc;
+ }
  return &mem->css;
+free_err:
+ for_each_possible_cpu(cpu)
+  __free_mem_cgroup_zonestat(mem->zonestat.cpustat[cpu]);
+ if (mem != &init_mem_cgroup)
+  kfree(mem);
+ return NULL;
 }

 static void mem_cgroup_pre_destroy(struct cgroup_subsys *ss,
@@ -937,7 +1091,12 @@ static void mem_cgroup_pre_destroy(struc
 static void mem_cgroup_destroy(struct cgroup_subsys *ss,
     struct cgroup *cont)
 {
- kfree(mem_cgroup_from_cont(cont));
+ int cpu;
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
+ for_each_possible_cpu(cpu)
+  __free_mem_cgroup_zonestat(mem->zonestat.cpustat[cpu]);
+
+ kfree(mem);
```

```
 }

 static int mem_cgroup_populate(struct cgroup_subsys *ss,
@@ -989,5 +1148,5 @@ struct cgroup_subsys mem_cgroup_subsys =
  .destroy = mem_cgroup_destroy,
  .populate = mem_cgroup_populate,
  .attach = mem_cgroup_move_task,
- .early_init = 1,
+ .early_init = 0,
 };
```

_____

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