Subject: [RFC][PATCH] memory cgroup enhancements updated [10/10] NUMA aware account
Posted by KAMEZAWA Hiroyuki on Fri, 19 Oct 2007 09:37:30 GMT
View Forum Message <> Reply to Message

This patch is a trial for making stat for memory_cgroup NUMA-aware.

 * dividing per-cpu stat to handle nid information.
 * add nid information to page_cgroup
 * Because init routine has to call kmalloc, init_early is set to be 0.

This is just a trial at this stage. Any comments are welcome.
Works well on my fake NUMA system.
I think we can add "numastat" based on this.

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

```
 mm/memcontrol.c |   97 ++++++++++++++++++++++++++++++++++++++++----------------
 1 file changed, 71 insertions(+), 26 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>

@@ -59,12 +60,18 @@ enum mem_cgroup_stat_index {
  MEM_CGROUP_STAT_NSTATS,
 };

+#ifndef CONFIG_NUMA
 struct mem_cgroup_stat_cpu {
- s64 count[MEM_CGROUP_STAT_NSTATS];
+ s64 count[1][MEM_CGROUP_STAT_NSTATS];
 } ____cacheline_aligned_in_smp;
+#else
+struct mem_cgroup_stat_cpu {
+ s64 count[MAX_NUMNODES][MEM_CGROUP_STAT_NSTATS];
+} ____cacheline_aligned_in_smp;
+#endif
```

```
 struct mem_cgroup_stat {
- struct mem_cgroup_stat_cpu cpustat[NR_CPUS];
+ struct mem_cgroup_stat_cpu *cpustat[NR_CPUS];
 };

 /*
@@ -72,28 +79,28 @@ struct mem_cgroup_stat {
  * MUST be called under preempt_disable().
  */
 static inline void __mem_cgroup_stat_add(struct mem_cgroup_stat *stat,
-                enum mem_cgroup_stat_index idx, int val)
+  enum mem_cgroup_stat_index idx, int nid, int val)
 {
  int cpu = smp_processor_id();
 #ifdef CONFIG_PREEMPT
  VM_BUG_ON(preempt_count() == 0);
 #endif
- stat->cpustat[cpu].count[idx] += val;
+ stat->cpustat[cpu]->count[nid][idx] += val;
 }

 static inline void mem_cgroup_stat_inc(struct mem_cgroup_stat *stat,
-  enum mem_cgroup_stat_index idx)
+  enum mem_cgroup_stat_index idx, int nid)
 {
  preempt_disable();
- __mem_cgroup_stat_add(stat, idx, 1);
+ __mem_cgroup_stat_add(stat, idx, nid, 1);
  preempt_enable();
 }

 static inline void mem_cgroup_stat_dec(struct mem_cgroup_stat *stat,
-  enum mem_cgroup_stat_index idx)
+  enum mem_cgroup_stat_index idx, int nid)
 {
  preempt_disable();
- __mem_cgroup_stat_add(stat, idx, -1);
+ __mem_cgroup_stat_add(stat, idx, nid, -1);
  preempt_enable();
 }

@@ -149,6 +156,7 @@ struct page_cgroup {
  struct list_head lru;  /* per cgroup LRU list */
  struct page *page;
  struct mem_cgroup *mem_cgroup;
+ int nid;
  atomic_t ref_cnt;  /* Helpful when pages move b/w  */
      /* mapped and cached states     */
```

```
  int  flags;
@@ -169,21 +177,23 @@ enum {
   * 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 mem_cgroup *mem, int nid,
+     int flags, int charge)
 {
   int val = (charge)? 1 : -1;
   struct mem_cgroup_stat *stat = &mem->stat;
   preempt_disable();

   if (flags & PCGF_PAGECACHE)
-   __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_PAGECACHE, val);
+   __mem_cgroup_stat_add(stat,
+     MEM_CGROUP_STAT_PAGECACHE, nid, val);
   else
-   __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_RSS, val);
+   __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_RSS, nid, val);

   if (flags & PCGF_ACTIVE)
-   __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE, val);
+   __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE, nid, val);
   else
-   __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE, val);
+   __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE, nid, val);

   preempt_enable();
 }
@@ -315,8 +325,10 @@ static void __mem_cgroup_move_lists(stru
   if (moved) {
     struct mem_cgroup_stat *stat = &mem->stat;
     preempt_disable();
-   __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE, moved);
-   __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE, -moved);
+   __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_ACTIVE,
+        pc->nid, moved);
+   __mem_cgroup_stat_add(stat, MEM_CGROUP_STAT_INACTIVE,
+      pc->nid, -moved);
     preempt_enable();
   }
   if (active) {
@@ -493,10 +505,10 @@ retry:
     bool is_atomic = gfp_mask & GFP_ATOMIC;
     if (is_cache)
      mem_cgroup_stat_inc(&mem->stat,
-     MEM_CGROUP_STAT_FAIL_CACHE);
```

```
+    MEM_CGROUP_STAT_FAIL_CACHE, page_to_nid(page));
   else
    mem_cgroup_stat_inc(&mem->stat,
-    MEM_CGROUP_STAT_FAIL_RSS);
+    MEM_CGROUP_STAT_FAIL_RSS, page_to_nid(page));
   /*
    * We cannot reclaim under GFP_ATOMIC, fail the charge
    */
@@ -537,6 +549,7 @@ noreclaim:
  atomic_set(&pc->ref_cnt, 1);
  pc->mem_cgroup = mem;
  pc->page = page;
+ pc->nid = page_to_nid(page);
  if (is_cache)
   pc->flags = PCGF_PAGECACHE | PCGF_ACTIVE;
  else
@@ -554,7 +567,7 @@ noreclaim:
  }

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

  spin_lock_irqsave(&mem->lru_lock, flags);
  list_add(&pc->lru, &mem->active_list);
@@ -621,7 +634,8 @@ void mem_cgroup_uncharge(struct page_cgr
    spin_lock_irqsave(&mem->lru_lock, flags);
    list_del_init(&pc->lru);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
-   mem_cgroup_charge_statistics(mem, pc->flags, false);
+   mem_cgroup_charge_statistics(mem, pc->nid, pc->flags,
+       false);
    kfree(pc);
   }
  }
@@ -657,6 +671,7 @@ void mem_cgroup_end_migration(struct pag
 void mem_cgroup_page_migration(struct page *page, struct page *newpage)
 {
  struct page_cgroup *pc;
+ struct mem_cgroup *mem;
 retry:
  pc = page_get_page_cgroup(page);
  if (!pc)
@@ -664,6 +679,11 @@ retry:
  if (clear_page_cgroup(page, pc) != pc)
   goto retry;
  pc->page = newpage;
+ pc->nid = page_to_nid(newpage);
```

```
+ mem = pc->mem_cgroup;
+ mem_cgroup_charge_statistics(mem, page_to_nid(page), pc->flags, false);
+ mem_cgroup_charge_statistics(mem,
+    page_to_nid(newpage), pc->flags, true);
  lock_page_cgroup(newpage);
  page_assign_page_cgroup(newpage, pc);
  unlock_page_cgroup(newpage);
@@ -697,7 +717,8 @@ 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(mem, pc->flags, pc->nid,
+     false);
    kfree(pc);
   } else  /* being uncharged ? ...do relax */
    break;
@@ -872,13 +893,16 @@ static int mem_control_stat_show(struct
  struct mem_cgroup_stat *stat = &mem_cont->stat;
  int i;

- for (i = 0; i < ARRAY_SIZE(stat->cpustat[0].count); i++) {
+ for (i = 0; i < MEM_CGROUP_STAT_NSTATS; i++) {
   unsigned int cpu;
+  int node;
   s64 val;

   val = 0;
-  for (cpu = 0; cpu < NR_CPUS; cpu++)
-   val += stat->cpustat[cpu].count[i];
+  for_each_possible_cpu(cpu)
+   for_each_node_state(node, N_POSSIBLE)
+    val += stat->cpustat[cpu]->count[node][i];
+
   val *= mem_cgroup_stat_desc[i].unit;
   seq_printf(m, "%s %lld\n", mem_cgroup_stat_desc[i].msg, val);
  }
@@ -941,12 +965,14 @@ 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;
   init_mm.mem_cgroup = mem;
- } else
+ } else {
```

```
   mem = kzalloc(sizeof(struct mem_cgroup), GFP_KERNEL);
+ }

  if (mem == NULL)
   return NULL;
@@ -956,6 +982,17 @@ 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_stat_cpu *mcsc;
+  if (sizeof(*mcsc) < PAGE_SIZE)
+   mcsc = kmalloc_node(sizeof(*mcsc), GFP_KERNEL, nid);
+  else
+   mcsc = vmalloc_node(sizeof(*mcsc), nid);
+  memset(mcsc, 0, sizeof(*mcsc));
+  mem->stat.cpustat[cpu] = mcsc;
+ }
  return &mem->css;
 }

@@ -969,7 +1006,15 @@ 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));
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
+ int cpu;
+ for_each_possible_cpu(cpu) {
+  if (sizeof(struct mem_cgroup_stat_cpu) < PAGE_SIZE)
+   kfree(mem->stat.cpustat[cpu]);
+  else
+   vfree(mem->stat.cpustat[cpu]);
+ }
+ kfree(mem);
 }

 static int mem_cgroup_populate(struct cgroup_subsys *ss,
@@ -1021,5 +1066,5 @@ 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