Subject: [RFC][PATCH] memory controller per zone patches take 2 [0/10] introduction
Posted by KAMEZAWA Hiroyuki on Fri, 16 Nov 2007 10:11:07 GMT
View Forum Message <> Reply to Message

Hi, this is updated version of patch set implementing per-zone on memory cgroup.

I still uses x86_64/fake NUMA (my ia64/NUMA box is under maintainance....)
So, RFC again. (I'd like to do 3rd update in the next week.)

Major Changes from previous one.
- per-zone-lru_lock patch is added.
- all per-zone objects of memory cgroup are treated in same way.
- page migration is handled.
- restructured and cleaned up.

Todo:
- do test on "real" NUMA.
- merge YAMAMOTO-san's background page reclaim patch set on this. (If I can)
- performance measurement at some point
- more cleanup and adding meaningful comments
- confirm added logic in vmscan.c is really sane.

Overview:

All per-zone obects are put into
==
```
struct mem_cgroup_per_zone {
        /*
         * spin_lock to protect the per cgroup LRU
         */
        spinlock_t              lru_lock;
        struct list_head        active_list;
        struct list_head        inactive_list;
        unsigned long count[NR_MEM_CGROUP_ZSTAT];
};
```
==
And this per-zone area is accessed by following functions.
==
```
mem_cgroup_zoneinfo(struct mem_cgroup *mem, int nid, int zid)
page_cgroup_zoneinfo(struct page_cgroup *pc)
```
==

Typical usage is following.
==
```
        mz = page_cgroup_zoneinfo(pc);
        spin_lock_irqsave(&mz->lru_lock, flags);
        __mem_cgroup_add_list(pc);
```

```
        spin_unlock_irqrestore(&mz->lru_lock, flags);
==
```

Thanks,
-Kame

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

---

Subject: [RFC][PATCH] memory controller per zone patches take 2 [1/10] add scan_global_lru() macro
Posted by KAMEZAWA Hiroyuki on Fri, 16 Nov 2007 10:14:45 GMT
View Forum Message <> Reply to Message

add macro scan_global_lru().

This is used to detect which scan_control scans global lru or
mem_cgroup lru. And compiled to be static value (1) when
memory controller is not configured. (maybe good for compiler)

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

```
 mm/vmscan.c |   17 ++++++++++-----
 1 file changed, 12 insertions(+), 5 deletions(-)

Index: linux-2.6.24-rc2-mm1/mm/vmscan.c
===================================================================
--- linux-2.6.24-rc2-mm1.orig/mm/vmscan.c
+++ linux-2.6.24-rc2-mm1/mm/vmscan.c
@@ -127,6 +127,12 @@ long vm_total_pages; /* The total number
 static LIST_HEAD(shrinker_list);
 static DECLARE_RWSEM(shrinker_rwsem);

+#ifdef CONFIG_CGROUP_MEM_CONT
+#define scan_global_lru(sc) (!(sc)->mem_cgroup)
+#else
+#define scan_global_lru(sc) (1)
+#endif
+
 /*
  * Add a shrinker callback to be called from the vm
  */
@@ -1290,11 +1296,12 @@ static unsigned long do_try_to_free_page
    * Don't shrink slabs when reclaiming memory from
```

```
     * over limit cgroups
     */
-  if (sc->mem_cgroup == NULL)
+  if (scan_global_lru(sc)) {
     shrink_slab(sc->nr_scanned, gfp_mask, lru_pages);
-  if (reclaim_state) {
-   nr_reclaimed += reclaim_state->reclaimed_slab;
-   reclaim_state->reclaimed_slab = 0;
+   if (reclaim_state) {
+    nr_reclaimed += reclaim_state->reclaimed_slab;
+    reclaim_state->reclaimed_slab = 0;
+   }
    }
    total_scanned += sc->nr_scanned;
    if (nr_reclaimed >= sc->swap_cluster_max) {
@@ -1321,7 +1328,7 @@ static unsigned long do_try_to_free_page
     congestion_wait(WRITE, HZ/10);
   }
   /* top priority shrink_caches still had more to do? don't OOM, then */
- if (!sc->all_unreclaimable && sc->mem_cgroup == NULL)
+ if (!sc->all_unreclaimable && scan_global_lru(sc))
    ret = 1;
 out:
  /*
```



---

Subject: [RFC][PATCH] memory controller per zone patches take 2 [2/10] add nid/zid function for page_cgroup
Posted by KAMEZAWA Hiroyuki on Fri, 16 Nov 2007 10:15:45 GMT
View Forum Message <> Reply to Message

Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
 mm/memcontrol.c |   10 ++++++++++
 1 file changed, 10 insertions(+)

Index: linux-2.6.24-rc2-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc2-mm1.orig/mm/memcontrol.c
+++ linux-2.6.24-rc2-mm1/mm/memcontrol.c
@@ -135,6 +135,16 @@ struct page_cgroup {
 #define PAGE_CGROUP_FLAG_CACHE (0x1) /* charged as cache */
 #define PAGE_CGROUP_FLAG_ACTIVE (0x2) /* page is active in this cgroup */

```
+static inline int page_cgroup_nid(struct page_cgroup *pc)
+{
+ return page_to_nid(pc->page);
+}
+
+static inline int page_cgroup_zid(struct page_cgroup *pc)
+{
+ return page_zonenum(pc->page);
+}
+
 enum {
  MEM_CGROUP_TYPE_UNSPEC = 0,
  MEM_CGROUP_TYPE_MAPPED,
```

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

---

Subject: [RFC][PATCH] memory controller per zone patches take 2 [3/10] add per zone active/inactive counter t
Posted by KAMEZAWA Hiroyuki on Fri, 16 Nov 2007 10:17:24 GMT
View Forum Message <> Reply to Message

Counting active/inactive per-zone in memory controller.

This patch adds per-zone status in memory cgroup.
These values are often read (as per-zone value) by page reclaiming.

In current design, per-zone stat is just a unsigned long value and
not an atomic value because they are modified only under lru_lock.
(for avoiding atomic_t ops.)

This patch adds ACTIVE and INACTIVE per-zone status values.

For handling per-zone status, this patch adds
```
  struct mem_cgroup_per_zone {
  ...
  }
```
and some helper functions. This will be useful to add per-zone objects
in mem_cgroup.

This patch turns memory controller's early_init to be 0 for calling
kmalloc().

Changelog V1 -> V2
- added mem_cgroup_per_zone struct.

This will help following patches to implement per-zone objects and
pack them into a struct.
- added __mem_cgroup_add_list() and __mem_cgroup_remove_list()
- fixed page migration handling.
- renamed zstat to info (per-zone-info)
This will be place for per-zone information(lru, lock, ..)
- use page_cgroup_nid()/zid() funcs.

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

mm/memcontrol.c | 166 ++++++++++++++++++++++++++++++++++++++++++++++++++++++---
1 file changed, 159 insertions(+), 7 deletions(-)

Index: linux-2.6.24-rc2-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc2-mm1.orig/mm/memcontrol.c
+++ linux-2.6.24-rc2-mm1/mm/memcontrol.c
@@ -78,6 +78,32 @@ static s64 mem_cgroup_read_stat(struct m
}

/*
+ * per-zone information in memory controller.
+ */
+
+enum mem_cgroup_zstat_index {
+ MEM_CGROUP_ZSTAT_ACTIVE,
+ MEM_CGROUP_ZSTAT_INACTIVE, /* Active = Total - Inactive */
+
+ NR_MEM_CGROUP_ZSTAT,
+};
+
+struct mem_cgroup_per_zone {
+ unsigned long count[NR_MEM_CGROUP_ZSTAT];
+};
+/* Macro for accessing counter */
+#define MEM_CGROUP_ZSTAT(mz, idx) ((mz)->count[(idx)])
+
+struct mem_cgroup_per_node {
+ struct mem_cgroup_per_zone zoneinfo[MAX_NR_ZONES];
+};
+
+struct mem_cgroup_lru_info {
+ struct mem_cgroup_per_node *nodeinfo[MAX_NUMNODES];
+};
+
+
+/*

```
  * 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,
@@ -101,6 +127,7 @@ struct mem_cgroup {
   */
  struct list_head active_list;
  struct list_head inactive_list;
+ struct mem_cgroup_lru_info info;
  /*
   * spin_lock to protect the per cgroup LRU
   */
@@ -158,6 +185,7 @@ enum charge_type {
  MEM_CGROUP_CHARGE_TYPE_MAPPED,
 };

+
 /*
  * Always modified under lru lock. Then, not necessary to preempt_disable()
  */
@@ -166,6 +194,7 @@ static void mem_cgroup_charge_statistics
 {
  int val = (charge)? 1 : -1;
  struct mem_cgroup_stat *stat = &mem->stat;
+
  VM_BUG_ON(!irqs_disabled());

  if (flags & PAGE_CGROUP_FLAG_CACHE)
@@ -173,7 +202,39 @@ static void mem_cgroup_charge_statistics
      MEM_CGROUP_STAT_CACHE, val);
  else
   __mem_cgroup_stat_add_safe(stat, MEM_CGROUP_STAT_RSS, val);
+}

+static inline struct mem_cgroup_per_zone *
+mem_cgroup_zoneinfo(struct mem_cgroup *mem, int nid, int zid)
+{
+ if (!mem->info.nodeinfo[nid])
+  return NULL;
+ return &mem->info.nodeinfo[nid]->zoneinfo[zid];
+}
+
+static inline struct mem_cgroup_per_zone *
+page_cgroup_zoneinfo(struct page_cgroup *pc)
+{
+ struct mem_cgroup *mem = pc->mem_cgroup;
+ int nid = page_cgroup_nid(pc);
+ int zid = page_cgroup_zid(pc);
+
```

```
+ return mem_cgroup_zoneinfo(mem, nid, zid);
+}
+
+static unsigned long mem_cgroup_get_all_zonestat(struct mem_cgroup *mem,
+     enum mem_cgroup_zstat_index idx)
+{
+ int nid, zid;
+ struct mem_cgroup_per_zone *mz;
+ u64 total = 0;
+
+ for_each_online_node(nid)
+  for (zid = 0; zid < MAX_NR_ZONES; zid++) {
+   mz = mem_cgroup_zoneinfo(mem, nid, zid);
+   total += MEM_CGROUP_ZSTAT(mz, idx);
+  }
+ return total;
 }

 static struct mem_cgroup init_mem_cgroup;
@@ -286,12 +347,51 @@ static struct page_cgroup *clear_page_cg
  return ret;
 }

+static void __mem_cgroup_remove_list(struct page_cgroup *pc)
+{
+ int from = pc->flags & PAGE_CGROUP_FLAG_ACTIVE;
+ struct mem_cgroup_per_zone *mz = page_cgroup_zoneinfo(pc);
+
+ if (from)
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_ACTIVE) -= 1;
+ else
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_INACTIVE) -= 1;
+
+ mem_cgroup_charge_statistics(pc->mem_cgroup, pc->flags, false);
+ list_del_init(&pc->lru);
+}
+
+static void __mem_cgroup_add_list(struct page_cgroup *pc)
+{
+ int to = pc->flags & PAGE_CGROUP_FLAG_ACTIVE;
+ struct mem_cgroup_per_zone *mz = page_cgroup_zoneinfo(pc);
+
+ if (!to) {
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_INACTIVE) += 1;
+  list_add(&pc->lru, &pc->mem_cgroup->inactive_list);
+ } else {
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_ACTIVE) += 1;
+  list_add(&pc->lru, &pc->mem_cgroup->active_list);
```

```
+ }
+ mem_cgroup_charge_statistics(pc->mem_cgroup, pc->flags, true);
+}
+
 static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
+ int from = pc->flags & PAGE_CGROUP_FLAG_ACTIVE;
+ struct mem_cgroup_per_zone *mz = page_cgroup_zoneinfo(pc);
+
+ if (from)
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_ACTIVE) -= 1;
+ else
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_INACTIVE) -= 1;
+
  if (active) {
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_ACTIVE) += 1;
   pc->flags |= PAGE_CGROUP_FLAG_ACTIVE;
   list_move(&pc->lru, &pc->mem_cgroup->active_list);
  } else {
+  MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_INACTIVE) += 1;
   pc->flags &= ~PAGE_CGROUP_FLAG_ACTIVE;
   list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
  }
@@ -511,8 +611,7 @@ 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);
+ __mem_cgroup_add_list(pc);
  spin_unlock_irqrestore(&mem->lru_lock, flags);

 done:
@@ -576,13 +675,13 @@ 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);
-   list_del_init(&pc->lru);
-   mem_cgroup_charge_statistics(mem, pc->flags, false);
+    __mem_cgroup_remove_list(pc);
    spin_unlock_irqrestore(&mem->lru_lock, flags);
    kfree(pc);
   }
  }
 }
+
 /*
  * Returns non-zero if a page (under migration) has valid page_cgroup member.
```

```
  * Refcnt of page_cgroup is incremented.
@@ -614,16 +713,26 @@ 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;
+ unsigned long flags;
 retry:
  pc = page_get_page_cgroup(page);
  if (!pc)
   return;
+ mem = pc->mem_cgroup;
  if (clear_page_cgroup(page, pc) != pc)
   goto retry;
+
+ spin_lock_irqsave(&mem->lru_lock, flags);
+
+ __mem_cgroup_remove_list(pc);
  pc->page = newpage;
  lock_page_cgroup(newpage);
  page_assign_page_cgroup(newpage, pc);
  unlock_page_cgroup(newpage);
+ __mem_cgroup_add_list(pc);
+
+ spin_unlock_irqrestore(&mem->lru_lock, flags);
  return;
 }

@@ -651,10 +760,11 @@ retry:
   /* Avoid race with charge */
   atomic_set(&pc->ref_cnt, 0);
   if (clear_page_cgroup(page, pc) == pc) {
+   int active;
    css_put(&mem->css);
+   active = pc->flags & PAGE_CGROUP_FLAG_ACTIVE;
    res_counter_uncharge(&mem->res, PAGE_SIZE);
-   list_del_init(&pc->lru);
-   mem_cgroup_charge_statistics(mem, pc->flags, false);
+   __mem_cgroup_remove_list(pc);
    kfree(pc);
   } else  /* being uncharged ? ...do relax */
    break;
@@ -833,6 +943,17 @@ static int mem_control_stat_show(struct
   seq_printf(m, "%s %lld\n", mem_cgroup_stat_desc[i].msg,
    (long long)val);
  }
+ /* showing # of active pages */
+ {
```

```
+  unsigned long active, inactive;
+
+  inactive = mem_cgroup_get_all_zonestat(mem_cont,
+      MEM_CGROUP_ZSTAT_INACTIVE);
+  active = mem_cgroup_get_all_zonestat(mem_cont,
+      MEM_CGROUP_ZSTAT_ACTIVE);
+  seq_printf(m, "active %ld\n", (active) * PAGE_SIZE);
+  seq_printf(m, "inactive %ld\n", (inactive) * PAGE_SIZE);
+ }
   return 0;
  }

@@ -886,12 +1007,25 @@ static struct cftype mem_cgroup_files[]
   },
  };

+static int alloc_mem_cgroup_per_zone_info(struct mem_cgroup *mem, int node)
+{
+ struct mem_cgroup_per_node *pn;
+
+ pn = kmalloc_node(sizeof(*pn), GFP_KERNEL, node);
+ if (!pn)
+  return 1;
+ mem->info.nodeinfo[node] = pn;
+ memset(pn, 0, sizeof(*pn));
+ return 0;
+}
+
 static struct mem_cgroup init_mem_cgroup;

 static struct cgroup_subsys_state *
 mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
 {
  struct mem_cgroup *mem;
+ int node;

  if (unlikely((cont->parent) == NULL)) {
   mem = &init_mem_cgroup;
@@ -907,7 +1041,19 @@ 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;
+ memset(&mem->info, 0, sizeof(mem->info));
+
+ for_each_node_state(node, N_POSSIBLE)
+  if (alloc_mem_cgroup_per_zone_info(mem, node))
+   goto free_out;
+
```

```
  return &mem->css;
+free_out:
+ for_each_node_state(node, N_POSSIBLE)
+   kfree(mem->info.nodeinfo[node]);
+ if (cont->parent != NULL)
+   kfree(mem);
+ return NULL;
 }

 static void mem_cgroup_pre_destroy(struct cgroup_subsys *ss,
@@ -920,6 +1066,12 @@ static void mem_cgroup_pre_destroy(struc
 static void mem_cgroup_destroy(struct cgroup_subsys *ss,
     struct cgroup *cont)
 {
+ int node;
+ struct mem_cgroup *mem = mem_cgroup_from_cont(cont);
+
+ for_each_node_state(node, N_POSSIBLE)
+   kfree(mem->info.nodeinfo[node]);
+
   kfree(mem_cgroup_from_cont(cont));
 }

@@ -972,5 +1124,5 @@ 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,
 };
```



---

## Subject: [RFC][PATCH] memory controller per zone patches take 2 [4/10] calculate mapped ratio for memory cgro

Posted by KAMEZAWA Hiroyuki on Fri, 16 Nov 2007 10:17:43 GMT

View Forum Message <> Reply to Message

Define function for calculating mapped_ratio in memory cgroup.

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

```
 include/linux/memcontrol.h |   11 ++++++++++-
 mm/memcontrol.c            |   13 +++++++++++++
```

```
 2 files changed, 23 insertions(+), 1 deletion(-)

Index: linux-2.6.24-rc2-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc2-mm1.orig/mm/memcontrol.c
+++ linux-2.6.24-rc2-mm1/mm/memcontrol.c
@@ -423,6 +423,19 @@ void mem_cgroup_move_lists(struct page_c
  spin_unlock(&mem->lru_lock);
 }

+/*
+ * Calculate mapped_ratio under memory controller.
+ */
+int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem)
+{
+ s64 total, rss;
+
+ /* usage is recorded in bytes */
+ total = mem->res.usage >> PAGE_SHIFT;
+ rss = mem_cgroup_read_stat(&mem->stat, MEM_CGROUP_STAT_RSS);
+ return (rss * 100) / total;
+}
+
 unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
       struct list_head *dst,
       unsigned long *scanned, int order,
Index: linux-2.6.24-rc2-mm1/include/linux/memcontrol.h
===================================================================
--- linux-2.6.24-rc2-mm1.orig/include/linux/memcontrol.h
+++ linux-2.6.24-rc2-mm1/include/linux/memcontrol.h
@@ -61,6 +61,12 @@ extern int mem_cgroup_prepare_migration(
 extern void mem_cgroup_end_migration(struct page *page);
 extern void mem_cgroup_page_migration(struct page *page, struct page *newpage);

+/*
+ * For memory reclaim.
+ */
+extern int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem);
+
+
 #else /* CONFIG_CGROUP_MEM_CONT */
 static inline void mm_init_cgroup(struct mm_struct *mm,
       struct task_struct *p)
@@ -132,7 +138,10 @@ mem_cgroup_page_migration(struct page *p
 {
 }

-
+
```

```
+static inline int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem)
+{
+ return 0;
+}
 #endif /* CONFIG_CGROUP_MEM_CONT */

 #endif /* _LINUX_MEMCONTROL_H */
```



---

Subject: [RFC][PATCH] memory controller per zone patches take 2 [5/10] calculate active/inactive balance for
Posted by KAMEZAWA Hiroyuki on Fri, 16 Nov 2007 10:21:49 GMT
View Forum Message <> Reply to Message

Define function for determining active/inactive balance in memory
cgroup.
 * just use res.usage as total value and assumes total = active + inactive.
 * and yes, we can use mem_cgroup_get_all_zonestat(mem, MEM_CGROUP_ZSTAT_ACTIVE)

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

```
 include/linux/memcontrol.h |    8 ++++++++
 mm/memcontrol.c            |   16 ++++++++++++++++
 2 files changed, 24 insertions(+)

Index: linux-2.6.24-rc2-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc2-mm1.orig/mm/memcontrol.c
+++ linux-2.6.24-rc2-mm1/mm/memcontrol.c
@@ -435,6 +435,22 @@ int mem_cgroup_calc_mapped_ratio(struct
  rss = mem_cgroup_read_stat(&mem->stat, MEM_CGROUP_STAT_RSS);
  return (rss * 100) / total;
 }
+/*
+ * Uses mem_cgroup's imbalance instead of zone's lru imbalance.
+ * This will be used for determining whether page out routine try to free
+ * mapped pages or not.
+ */
+int mem_cgroup_reclaim_imbalance(struct mem_cgroup *mem)
+{
+ s64 total, active, inactive;
+
+ /* usage is recorded in bytes */
```

```
+ total = mem->res.usage >> PAGE_SHIFT;
+ inactive = mem_cgroup_get_all_zonestat(mem, MEM_CGROUP_ZSTAT_INACTIVE);
+ active = total - inactive;
+
+ return active / (inactive + 1);
+}

 unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
      struct list_head *dst,
Index: linux-2.6.24-rc2-mm1/include/linux/memcontrol.h
===================================================================
--- linux-2.6.24-rc2-mm1.orig/include/linux/memcontrol.h
+++ linux-2.6.24-rc2-mm1/include/linux/memcontrol.h
@@ -65,6 +65,8 @@ extern void mem_cgroup_page_migration(st
  * For memory reclaim.
  */
 extern int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem);
+extern int mem_cgroup_reclaim_imbalance(struct mem_cgroup *mem);
+


 #else /* CONFIG_CGROUP_MEM_CONT */
@@ -142,6 +144,12 @@ static inline int mem_cgroup_calc_mapped
 {
  return 0;
 }
+
+static inline int mem_cgroup_reclaim_imbalance(struct mem_cgroup *mem)
+{
+ return 0;
+}
+
 #endif /* CONFIG_CGROUP_MEM_CONT */

 #endif /* _LINUX_MEMCONTROL_H */
```



---

Subject: [RFC][PATCH] memory controller per zone patches take 2 [6/10] remember reclaim priority for memory c
Posted by KAMEZAWA Hiroyuki on Fri, 16 Nov 2007 10:23:30 GMT
View Forum Message <> Reply to Message

Define function to remember reclaim priority (as zone->prev_priority)

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

 include/linux/memcontrol.h |   23 ++++++++++++++++++++++
 mm/memcontrol.c            |   20 ++++++++++++++++++++
 2 files changed, 43 insertions(+)

Index: linux-2.6.24-rc2-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc2-mm1.orig/mm/memcontrol.c
+++ linux-2.6.24-rc2-mm1/mm/memcontrol.c
@@ -133,6 +133,7 @@ struct mem_cgroup {
   */
  spinlock_t lru_lock;
  unsigned long control_type; /* control RSS or RSS+Pagecache */
+ int prev_priority; /* for recording reclaim priority */
  /*
   * statistics.
   */
@@ -452,6 +453,25 @@ int mem_cgroup_reclaim_imbalance(struct
  return active / (inactive + 1);
 }

+/*
+ * prev_priority control...this will be used in memory reclaim path.
+ */
+int mem_cgroup_get_reclaim_priority(struct mem_cgroup *mem)
+{
+ return mem->prev_priority;
+}
+
+void mem_cgroup_note_reclaim_priority(struct mem_cgroup *mem, int priority)
+{
+ if (priority < mem->prev_priority)
+  mem->prev_priority = priority;
+}
+
+void mem_cgroup_record_reclaim_priority(struct mem_cgroup *mem, int priority)
+{
+ mem->prev_priority = priority;
+}
+
 unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
      struct list_head *dst,
      unsigned long *scanned, int order,
Index: linux-2.6.24-rc2-mm1/include/linux/memcontrol.h
===================================================================
--- linux-2.6.24-rc2-mm1.orig/include/linux/memcontrol.h
```

```
+++ linux-2.6.24-rc2-mm1/include/linux/memcontrol.h
@@ -67,6 +67,11 @@ extern void mem_cgroup_page_migration(st
 extern int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem);
 extern int mem_cgroup_reclaim_imbalance(struct mem_cgroup *mem);

+extern int mem_cgroup_get_reclaim_priority(struct mem_cgroup *mem);
+extern void mem_cgroup_note_reclaim_priority(struct mem_cgroup *mem,
+       int priority);
+extern void mem_cgroup_record_reclaim_priority(struct mem_cgroup *mem,
+       int priority);


 #else /* CONFIG_CGROUP_MEM_CONT */
@@ -150,6 +155,24 @@ static inline int mem_cgroup_reclaim_imb
  return 0;
 }

+static inline int mem_cgroup_get_reclaim_priority(struct mem_cgroup *mem,
+      int priority)
+{
+ return 0;
+}
+
+static inline void mem_cgroup_note_reclaim_priority(struct mem_cgroup *mem,
+      int priority)
+{
+ return 0;
+}
+
+static inline void mem_cgroup_record_reclaim_priority(struct mem_cgroup *mem,
+      int priority)
+{
+ return 0;
+}
+
 #endif /* CONFIG_CGROUP_MEM_CONT */

 #endif /* _LINUX_MEMCONTROL_H */
```

---



---

## Subject: [RFC][PATCH] memory controller per zone patches take 2 [7/10] calculate reclaim scan number for memo

Define function for calculating the number of scan target on each Zone/LRU.

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

```
 include/linux/memcontrol.h |   15 +++++++++++++++
 mm/memcontrol.c            |   33 +++++++++++++++++++++++++++++++++
 2 files changed, 48 insertions(+)

Index: linux-2.6.24-rc2-mm1/include/linux/memcontrol.h
===================================================================
--- linux-2.6.24-rc2-mm1.orig/include/linux/memcontrol.h
+++ linux-2.6.24-rc2-mm1/include/linux/memcontrol.h
@@ -73,6 +73,10 @@ extern void mem_cgroup_note_reclaim_prio
 extern void mem_cgroup_record_reclaim_priority(struct mem_cgroup *mem,
        int priority);

+extern long mem_cgroup_calc_reclaim_active(struct mem_cgroup *mem,
+    struct zone *zone, int priority);
+extern long mem_cgroup_calc_reclaim_inactive(struct mem_cgroup *mem,
+    struct zone *zone, int priority);

 #else /* CONFIG_CGROUP_MEM_CONT */
 static inline void mm_init_cgroup(struct mm_struct *mm,
@@ -173,6 +177,17 @@ static inline void mem_cgroup_record_rec
  return 0;
 }

+static inline long mem_cgroup_calc_reclaim_active(struct mem_cgroup *mem,
+     struct zone *zone, int priority)
+{
+ return 0;
+}
+
+static inline long mem_cgroup_calc_reclaim_inactive(struct mem_cgroup *mem,
+     struct zone *zone, int priority)
+{
+ return 0;
+}
 #endif /* CONFIG_CGROUP_MEM_CONT */

 #endif /* _LINUX_MEMCONTROL_H */
Index: linux-2.6.24-rc2-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc2-mm1.orig/mm/memcontrol.c
+++ linux-2.6.24-rc2-mm1/mm/memcontrol.c
@@ -472,6 +472,39 @@ void mem_cgroup_record_reclaim_priority(
```

```
 mem->prev_priority = priority;
}

+/*
+ * Calculate # of pages to be scanned in this priority/zone.
+ * See also vmscan.c
+ *
+ * priority starts from "DEF_PRIORITY" and decremented in each loop.
+ * (see include/linux/mmzone.h)
+ */
+
+long mem_cgroup_calc_reclaim_active(struct mem_cgroup *mem,
+       struct zone *zone, int priority)
+{
+ s64 nr_active;
+ int nid = zone->zone_pgdat->node_id;
+ int zid = zone_idx(zone);
+ struct mem_cgroup_per_zone *mz = mem_cgroup_zoneinfo(mem, nid, zid);
+
+ nr_active = MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_ACTIVE);
+ return (long)(nr_active >> priority);
+}
+
+long mem_cgroup_calc_reclaim_inactive(struct mem_cgroup *mem,
+     struct zone *zone, int priority)
+{
+ u64 nr_inactive;
+ int nid = zone->zone_pgdat->node_id;
+ int zid = zone_idx(zone);
+ struct mem_cgroup_per_zone *mz = mem_cgroup_zoneinfo(mem, nid, zid);
+
+ nr_inactive = MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_INACTIVE);
+
+ return (long)(nr_inactive >> priority);
+}
+
 unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
      struct list_head *dst,
      unsigned long *scanned, int order,
```



## Subject: [RFC][PATCH] memory controller per zone patches take 2 [8/10] changes

in vmscan.c
Posted by KAMEZAWA Hiroyuki on Fri, 16 Nov 2007 10:25:12 GMT
View Forum Message <> Reply to Message

When using memory controller, there are 2 levels of memory reclaim.
1. zone memory reclaim because of system/zone memory shortage.
2. memory cgroup memory reclaim because of hitting limit.

These two can be distinguished by sc->mem_cgroup parameter.

This patch tries to make memory cgroup reclaim routine avoid affecting
system/zone memory reclaim. This patch inserts if (!sc->mem_cgroup) and
hook to memory_cgroup reclaim support functions.

This patch can be a help for isolating system lru activity and group lru
activity and shows what additional functions are necessary.

* mem_cgroup_calc_mapped_ratio() ... calculate mapped ratio for cgroup.
* mem_cgroup_reclaim_imbalance() ... calculate active/inactive balance in
cgroup.
* mem_cgroup_calc_reclaim_active() ... calculate the number of active pages to
be scanned in this priority in mem_cgroup.

* mem_cgroup_calc_reclaim_inactive() ... calculate the number of inactive pages
to be scanned in this priority in mem_cgroup.

* mem_cgroup_all_unreclaimable() .. checks cgroup's page is all unreclaimable
or not.
* mem_cgroup_get_reclaim_priority() ...
* mem_cgroup_note_reclaim_priority() ... record reclaim priority (temporal)
* mem_cgroup_remember_reclaim_priority()
.... record reclaim priority as
zone->prev_priority.
This value is used for calc reclaim_mapped.
Changelog:
- merged calc_reclaim_mapped patch in previous version.

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

mm/vmscan.c | 326 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++----------------------
1 file changed, 197 insertions(+), 129 deletions(-)

Index: linux-2.6.24-rc2-mm1/mm/vmscan.c
===================================================================
--- linux-2.6.24-rc2-mm1.orig/mm/vmscan.c
+++ linux-2.6.24-rc2-mm1/mm/vmscan.c
@@ -863,7 +863,8 @@ static unsigned long shrink_inactive_lis
__mod_zone_page_state(zone, NR_ACTIVE, -nr_active);
__mod_zone_page_state(zone, NR_INACTIVE,

```
       -(nr_taken - nr_active));
-  zone->pages_scanned += nr_scan;
+  if (scan_global_lru(sc))
+   zone->pages_scanned += nr_scan;
   spin_unlock_irq(&zone->lru_lock);

   nr_scanned += nr_scan;
@@ -950,6 +951,113 @@ static inline int zone_is_near_oom(struc
 }

 /*
+ * Determine we should try to reclaim mapped pages.
+ * This is called only when sc->mem_cgroup is NULL.
+ */
+static int calc_reclaim_mapped(struct scan_control *sc, struct zone *zone,
+    int priority)
+{
+ long mapped_ratio;
+ long distress;
+ long swap_tendency;
+ long imbalance;
+ int reclaim_mapped;
+ int prev_priority;
+
+ if (scan_global_lru(sc) && zone_is_near_oom(zone))
+  return 1;
+ /*
+  * `distress' is a measure of how much trouble we're having
+  * reclaiming pages.  0 -> no problems.  100 -> great trouble.
+  */
+ if (scan_global_lru(sc))
+  prev_priority = zone->prev_priority;
+ else
+  prev_priority = mem_cgroup_get_reclaim_priority(sc->mem_cgroup);
+
+ distress = 100 >> min(prev_priority, priority);
+
+ /*
+  * The point of this algorithm is to decide when to start
+  * reclaiming mapped memory instead of just pagecache.  Work out
+  * how much memory
+  * is mapped.
+  */
+ if (scan_global_lru(sc))
+  mapped_ratio = ((global_page_state(NR_FILE_MAPPED) +
+    global_page_state(NR_ANON_PAGES)) * 100) /
+     vm_total_pages;
+ else
```

```
+  mapped_ratio = mem_cgroup_calc_mapped_ratio(sc->mem_cgroup);
+
+ /*
+  * Now decide how much we really want to unmap some pages.  The
+  * mapped ratio is downgraded - just because there's a lot of
+  * mapped memory doesn't necessarily mean that page reclaim
+  * isn't succeeding.
+  *
+  * The distress ratio is important - we don't want to start
+  * going oom.
+  *
+  * A 100% value of vm_swappiness overrides this algorithm
+  * altogether.
+  */
+ swap_tendency = mapped_ratio / 2 + distress + sc->swappiness;
+
+ /*
+  * If there's huge imbalance between active and inactive
+  * (think active 100 times larger than inactive) we should
+  * become more permissive, or the system will take too much
+  * cpu before it start swapping during memory pressure.
+  * Distress is about avoiding early-oom, this is about
+  * making swappiness graceful despite setting it to low
+  * values.
+  *
+  * Avoid div by zero with nr_inactive+1, and max resulting
+  * value is vm_total_pages.
+  */
+ if (scan_global_lru(sc)) {
+  imbalance  = zone_page_state(zone, NR_ACTIVE);
+  imbalance /= zone_page_state(zone, NR_INACTIVE) + 1;
+ } else
+  imbalance = mem_cgroup_reclaim_imbalance(sc->mem_cgroup);
+
+ /*
+  * Reduce the effect of imbalance if swappiness is low,
+  * this means for a swappiness very low, the imbalance
+  * must be much higher than 100 for this logic to make
+  * the difference.
+  *
+  * Max temporary value is vm_total_pages*100.
+  */
+ imbalance *= (vm_swappiness + 1);
+ imbalance /= 100;
+
+ /*
+  * If not much of the ram is mapped, makes the imbalance
+  * less relevant, it's high priority we refill the inactive
```

```
+  * list with mapped pages only in presence of high ratio of
+  * mapped pages.
+  *
+  * Max temporary value is vm_total_pages*100.
+  */
+ imbalance *= mapped_ratio;
+ imbalance /= 100;
+
+ /* apply imbalance feedback to swap_tendency */
+ swap_tendency += imbalance;
+
+ /*
+  * Now use this metric to decide whether to start moving mapped
+  * memory onto the inactive list.
+  */
+ if (swap_tendency >= 100)
+  reclaim_mapped = 1;
+
+ return reclaim_mapped;
+}
+
+/*
  * This moves pages from the active list to the inactive list.
  *
  * We move them the other way if the page is referenced by one or more
@@ -966,6 +1074,8 @@ static inline int zone_is_near_oom(struc
  * The downside is that we have to touch page->_count against each page.
  * But we had to alter page->flags anyway.
  */
+
+
 static void shrink_active_list(unsigned long nr_pages, struct zone *zone,
     struct scan_control *sc, int priority)
 {
@@ -979,100 +1089,21 @@ static void shrink_active_list(unsigned
  struct pagevec pvec;
  int reclaim_mapped = 0;

- if (sc->may_swap) {
-  long mapped_ratio;
-  long distress;
-  long swap_tendency;
-  long imbalance;
-
-  if (zone_is_near_oom(zone))
-   goto force_reclaim_mapped;
-
-  /*
```

```
-	 * `distress' is a measure of how much trouble we're having
-	 * reclaiming pages.  0 -> no problems.  100 -> great trouble.
-	 */
-	distress = 100 >> min(zone->prev_priority, priority);
-
-	/*
-	 * The point of this algorithm is to decide when to start
-	 * reclaiming mapped memory instead of just pagecache.  Work out
-	 * how much memory
-	 * is mapped.
-	 */
-	mapped_ratio = ((global_page_state(NR_FILE_MAPPED) +
-	   global_page_state(NR_ANON_PAGES)) * 100) /
-	     vm_total_pages;
-
-	/*
-	 * Now decide how much we really want to unmap some pages.  The
-	 * mapped ratio is downgraded - just because there's a lot of
-	 * mapped memory doesn't necessarily mean that page reclaim
-	 * isn't succeeding.
-	 *
-	 * The distress ratio is important - we don't want to start
-	 * going oom.
-	 *
-	 * A 100% value of vm_swappiness overrides this algorithm
-	 * altogether.
-	 */
-	swap_tendency = mapped_ratio / 2 + distress + sc->swappiness;
-
-	/*
-	 * If there's huge imbalance between active and inactive
-	 * (think active 100 times larger than inactive) we should
-	 * become more permissive, or the system will take too much
-	 * cpu before it start swapping during memory pressure.
-	 * Distress is about avoiding early-oom, this is about
-	 * making swappiness graceful despite setting it to low
-	 * values.
-	 *
-	 * Avoid div by zero with nr_inactive+1, and max resulting
-	 * value is vm_total_pages.
-	 */
-	imbalance  = zone_page_state(zone, NR_ACTIVE);
-	imbalance /= zone_page_state(zone, NR_INACTIVE) + 1;
-
-	/*
-	 * Reduce the effect of imbalance if swappiness is low,
-	 * this means for a swappiness very low, the imbalance
-	 * must be much higher than 100 for this logic to make
```

```
-   * the difference.
-   *
-   * Max temporary value is vm_total_pages*100.
-   */
-  imbalance *= (vm_swappiness + 1);
-  imbalance /= 100;
-
-  /*
-   * If not much of the ram is mapped, makes the imbalance
-   * less relevant, it's high priority we refill the inactive
-   * list with mapped pages only in presence of high ratio of
-   * mapped pages.
-   *
-   * Max temporary value is vm_total_pages*100.
-   */
-  imbalance *= mapped_ratio;
-  imbalance /= 100;
-
-  /* apply imbalance feedback to swap_tendency */
-  swap_tendency += imbalance;
-
-  /*
-   * Now use this metric to decide whether to start moving mapped
-   * memory onto the inactive list.
-   */
-  if (swap_tendency >= 100)
-force_reclaim_mapped:
-   reclaim_mapped = 1;
- }
+ if (sc->may_swap)
+  reclaim_mapped = calc_reclaim_mapped(sc, zone, priority);

  lru_add_drain();
  spin_lock_irq(&zone->lru_lock);
  pgmoved = sc->isolate_pages(nr_pages, &l_hold, &pgscanned, sc->order,
      ISOLATE_ACTIVE, zone,
      sc->mem_cgroup, 1);
- zone->pages_scanned += pgscanned;
+ /*
+  * zone->pages_scanned is used for detect zone's oom
+  * mem_cgroup remembers nr_scan by itself.
+  */
+ if (scan_global_lru(sc))
+  zone->pages_scanned += pgscanned;
+
  __mod_zone_page_state(zone, NR_ACTIVE, -pgmoved);
  spin_unlock_irq(&zone->lru_lock);
```

```
@@ -1165,25 +1196,39 @@ static unsigned long shrink_zone(int pri
  unsigned long nr_to_scan;
  unsigned long nr_reclaimed = 0;

- /*
-  * Add one to `nr_to_scan' just to make sure that the kernel will
-  * slowly sift through the active list.
-  */
- zone->nr_scan_active +=
-  (zone_page_state(zone, NR_ACTIVE) >> priority) + 1;
- nr_active = zone->nr_scan_active;
- if (nr_active >= sc->swap_cluster_max)
-  zone->nr_scan_active = 0;
- else
-  nr_active = 0;
+ if (scan_global_lru(sc)) {
+  /*
+   * Add one to nr_to_scan just to make sure that the kernel
+   * will slowly sift through the active list.
+   */
+  zone->nr_scan_active +=
+   (zone_page_state(zone, NR_ACTIVE) >> priority) + 1;
+  nr_active = zone->nr_scan_active;
+  zone->nr_scan_inactive +=
+   (zone_page_state(zone, NR_INACTIVE) >> priority) + 1;
+  nr_inactive = zone->nr_scan_inactive;
+  if (nr_inactive >= sc->swap_cluster_max)
+   zone->nr_scan_inactive = 0;
+  else
+   nr_inactive = 0;
+
+  if (nr_active >= sc->swap_cluster_max)
+   zone->nr_scan_active = 0;
+  else
+   nr_active = 0;
+ } else {
+  /*
+   * This reclaim occurs not because zone memory shortage but
+   * because memory controller hits its limit.
+   * Then, don't modify zone reclaim related data.
+   */
+  nr_active = mem_cgroup_calc_reclaim_active(sc->mem_cgroup,
+     zone, priority);
+
+  nr_inactive = mem_cgroup_calc_reclaim_inactive(sc->mem_cgroup,
+     zone, priority);
+ }
```

```
- zone->nr_scan_inactive +=
-  (zone_page_state(zone, NR_INACTIVE) >> priority) + 1;
- nr_inactive = zone->nr_scan_inactive;
- if (nr_inactive >= sc->swap_cluster_max)
-  zone->nr_scan_inactive = 0;
- else
-  nr_inactive = 0;

  while (nr_active || nr_inactive) {
   if (nr_active) {
@@ -1228,25 +1273,39 @@ static unsigned long shrink_zones(int pr
  unsigned long nr_reclaimed = 0;
  int i;

+
  sc->all_unreclaimable = 1;
  for (i = 0; zones[i] != NULL; i++) {
   struct zone *zone = zones[i];

   if (!populated_zone(zone))
    continue;
+  /*
+   * Take care memory controller reclaiming has small influence
+   * to global LRU.
+   */
+  if (scan_global_lru(sc)) {
+   if (!cpuset_zone_allowed_hardwall(zone, GFP_KERNEL))
+    continue;
+   note_zone_scanning_priority(zone, priority);

-  if (!cpuset_zone_allowed_hardwall(zone, GFP_KERNEL))
-   continue;
-
-  note_zone_scanning_priority(zone, priority);
-
-  if (zone_is_all_unreclaimable(zone) && priority != DEF_PRIORITY)
-   continue; /* Let kswapd poll it */
-
-  sc->all_unreclaimable = 0;
+   if (zone_is_all_unreclaimable(zone) &&
+       priority != DEF_PRIORITY)
+    continue; /* Let kswapd poll it */
+   sc->all_unreclaimable = 0;
+  } else {
+   /*
+    * Ignore cpuset limitation here. We just want to reduce
+    * # of used pages by us regardless of memory shortage.
+    */
```

```
+   sc->all_unreclaimable = 0;
+   mem_cgroup_note_reclaim_priority(sc->mem_cgroup,
+       priority);
+  }

   nr_reclaimed += shrink_zone(priority, zone, sc);
  }
+
  return nr_reclaimed;
 }

@@ -1275,15 +1334,19 @@ static unsigned long do_try_to_free_page
  int i;

  count_vm_event(ALLOCSTALL);
+ /*
+  * mem_cgroup will not do shrink_slab.
+  */
+ if (scan_global_lru(sc)) {
+  for (i = 0; zones[i] != NULL; i++) {
+   struct zone *zone = zones[i];

- for (i = 0; zones[i] != NULL; i++) {
-  struct zone *zone = zones[i];
-
-  if (!cpuset_zone_allowed_hardwall(zone, GFP_KERNEL))
-   continue;
+   if (!cpuset_zone_allowed_hardwall(zone, GFP_KERNEL))
+    continue;

-  lru_pages += zone_page_state(zone, NR_ACTIVE)
-    + zone_page_state(zone, NR_INACTIVE);
+   lru_pages += zone_page_state(zone, NR_ACTIVE)
+     + zone_page_state(zone, NR_INACTIVE);
+  }
  }

  for (priority = DEF_PRIORITY; priority >= 0; priority--) {
@@ -1340,14 +1403,19 @@ out:
   */
  if (priority < 0)
   priority = 0;
- for (i = 0; zones[i] != NULL; i++) {
-  struct zone *zone = zones[i];

-  if (!cpuset_zone_allowed_hardwall(zone, GFP_KERNEL))
-   continue;
+ if (scan_global_lru(sc)) {
```

```
+  for (i = 0; zones[i] != NULL; i++) {
+   struct zone *zone = zones[i];
+
+   if (!cpuset_zone_allowed_hardwall(zone, GFP_KERNEL))
+    continue;
+
+   zone->prev_priority = priority;
+  }
+ } else
+  mem_cgroup_record_reclaim_priority(sc->mem_cgroup, priority);

-  zone->prev_priority = priority;
- }
  return ret;
 }
```



---

Subject: [RFC][PATCH] memory controller per zone patches take 2 [9/10] per-zone-lru for memory cgroup
Posted by KAMEZAWA Hiroyuki on Fri, 16 Nov 2007 10:25:51 GMT
View Forum Message <> Reply to Message

This patch implements per-zone lru for memory cgroup.
This patch makes use of mem_cgroup_per_zone struct for per zone lru.

LRU can be accessed by

```
  mz = mem_cgroup_zoneinfo(mem_cgroup, node, zone);
  &mz->active_list
  &mz->inactive_list

  or
  mz = page_cgroup_zoneinfo(page_cgroup, node, zone);
  &mz->active_list
  &mz->inactive_list
```

Changelog v1->v2
 - merged to mem_cgroup_per_zone struct.
 - handle page migraiton.

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

```
 mm/memcontrol.c |   63 ++++++++++++++++++++++++++++++++++++----------------
 1 file changed, 39 insertions(+), 24 deletions(-)

Index: linux-2.6.24-rc2-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc2-mm1.orig/mm/memcontrol.c
+++ linux-2.6.24-rc2-mm1/mm/memcontrol.c
@@ -89,6 +89,8 @@ enum mem_cgroup_zstat_index {
 };

 struct mem_cgroup_per_zone {
+ struct list_head active_list;
+ struct list_head inactive_list;
   unsigned long count[NR_MEM_CGROUP_ZSTAT];
 };
 /* Macro for accessing counter */
@@ -123,10 +125,7 @@ struct mem_cgroup {
   /*
    * Per cgroup active and inactive list, similar to the
    * per zone LRU lists.
-  * TODO: Consider making these lists per zone
    */
- struct list_head active_list;
- struct list_head inactive_list;
   struct mem_cgroup_lru_info info;
   /*
    * spin_lock to protect the per cgroup LRU
@@ -369,10 +368,10 @@ static void __mem_cgroup_add_list(struct

   if (!to) {
    MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_INACTIVE) += 1;
-  list_add(&pc->lru, &pc->mem_cgroup->inactive_list);
+  list_add(&pc->lru, &mz->inactive_list);
   } else {
    MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_ACTIVE) += 1;
-  list_add(&pc->lru, &pc->mem_cgroup->active_list);
+  list_add(&pc->lru, &mz->active_list);
   }
   mem_cgroup_charge_statistics(pc->mem_cgroup, pc->flags, true);
 }
@@ -390,11 +389,11 @@ static void __mem_cgroup_move_lists(stru
   if (active) {
    MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_ACTIVE) += 1;
    pc->flags |= PAGE_CGROUP_FLAG_ACTIVE;
-  list_move(&pc->lru, &pc->mem_cgroup->active_list);
+  list_move(&pc->lru, &mz->active_list);
```

```
  } else {
   MEM_CGROUP_ZSTAT(mz, MEM_CGROUP_ZSTAT_INACTIVE) += 1;
   pc->flags &= ~PAGE_CGROUP_FLAG_ACTIVE;
-  list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
+  list_move(&pc->lru, &mz->inactive_list);
  }
 }

@@ -518,11 +517,16 @@ unsigned long mem_cgroup_isolate_pages(u
  LIST_HEAD(pc_list);
  struct list_head *src;
  struct page_cgroup *pc, *tmp;
+ int nid = z->zone_pgdat->node_id;
+ int zid = zone_idx(z);
+ struct mem_cgroup_per_zone *mz;

+ mz = mem_cgroup_zoneinfo(mem_cont, nid, zid);
  if (active)
-  src = &mem_cont->active_list;
+  src = &mz->active_list;
  else
-  src = &mem_cont->inactive_list;
+  src = &mz->inactive_list;
+

  spin_lock(&mem_cont->lru_lock);
  scan = 0;
@@ -544,13 +548,6 @@ unsigned long mem_cgroup_isolate_pages(u
    continue;
   }

-  /*
-   * Reclaim, per zone
-   * TODO: make the active/inactive lists per zone
-   */
-  if (page_zone(page) != z)
-   continue;
-
   scan++;
   list_move(&pc->lru, &pc_list);

@@ -832,6 +829,8 @@ mem_cgroup_force_empty_list(struct mem_c
  int count;
  unsigned long flags;

+ if (list_empty(list))
+  return;
 retry:
```

```
  count = FORCE_UNCHARGE_BATCH;
  spin_lock_irqsave(&mem->lru_lock, flags);
@@ -867,20 +866,27 @@ retry:
 int mem_cgroup_force_empty(struct mem_cgroup *mem)
 {
  int ret = -EBUSY;
+ int node, zid;
  css_get(&mem->css);
  /*
   * page reclaim code (kswapd etc..) will move pages between
`  * active_list <-> inactive_list while we don't take a lock.
   * So, we have to do loop here until all lists are empty.
   */
- while (!(list_empty(&mem->active_list) &&
-   list_empty(&mem->inactive_list))) {
+ while (mem->res.usage > 0) {
   if (atomic_read(&mem->css.cgroup->count) > 0)
    goto out;
-  /* drop all page_cgroup in active_list */
-  mem_cgroup_force_empty_list(mem, &mem->active_list);
-  /* drop all page_cgroup in inactive_list */
-  mem_cgroup_force_empty_list(mem, &mem->inactive_list);
+  for_each_node_state(node, N_POSSIBLE)
+   for (zid = 0; zid < MAX_NR_ZONES; zid++) {
+    struct mem_cgroup_per_zone *mz;
+    mz = mem_cgroup_zoneinfo(mem, node, zid);
+    /* drop all page_cgroup in active_list */
+    mem_cgroup_force_empty_list(mem,
+       &mz->active_list);
+    /* drop all page_cgroup in inactive_list */
+    mem_cgroup_force_empty_list(mem,
+       &mz->inactive_list);
+   }
  }
  ret = 0;
 out:
@@ -1092,15 +1098,25 @@ static struct cftype mem_cgroup_files[]
 static int alloc_mem_cgroup_per_zone_info(struct mem_cgroup *mem, int node)
 {
  struct mem_cgroup_per_node *pn;
+ struct mem_cgroup_per_zone *mz;
+ int zone;

  pn = kmalloc_node(sizeof(*pn), GFP_KERNEL, node);
  if (!pn)
   return 1;
+
  mem->info.nodeinfo[node] = pn;
```

```
  memset(pn, 0, sizeof(*pn));
+
+ for (zone = 0; zone < MAX_NR_ZONES; zone++) {
+   mz = &pn->zoneinfo[zone];
+   INIT_LIST_HEAD(&mz->active_list);
+   INIT_LIST_HEAD(&mz->inactive_list);
+ }
  return 0;
 }

+
 static struct mem_cgroup init_mem_cgroup;

 static struct cgroup_subsys_state *
@@ -1119,8 +1135,7 @@ mem_cgroup_create(struct cgroup_subsys *
   return NULL;

  res_counter_init(&mem->res);
- INIT_LIST_HEAD(&mem->active_list);
- INIT_LIST_HEAD(&mem->inactive_list);
+
  spin_lock_init(&mem->lru_lock);
  mem->control_type = MEM_CGROUP_TYPE_ALL;
  memset(&mem->info, 0, sizeof(mem->info));
```

---

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

---

## Subject: [RFC][PATCH] memory controller per zone patches take 2 [10/10] per-zone-lock for memory cgroup

Posted by KAMEZAWA Hiroyuki on Fri, 16 Nov 2007 10:27:33 GMT
View Forum Message <> Reply to Message

Now, lru is per-zone.

Then, lru_lock can be (should be) per-zone, too.
This patch implementes per-zone lru lock.

lru_lock is placed into mem_cgroup_per_zone struct.

lock can be accessed by
  mz = mem_cgroup_zoneinfo(mem_cgroup, node, zone);
  &mz->lru_lock

  or

```
   mz = page_cgroup_zoneinfo(page_cgroup, node, zone);
   &mz->lru_lock


Signed-off-by: KAMEZAWA hiroyuki <kmaezawa.hiroyu@jp.fujitsu.com>

 mm/memcontrol.c |   71 ++++++++++++++++++++++++++++++++++++----------------
 1 file changed, 44 insertions(+), 27 deletions(-)

Index: linux-2.6.24-rc2-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc2-mm1.orig/mm/memcontrol.c
+++ linux-2.6.24-rc2-mm1/mm/memcontrol.c
@@ -89,6 +89,10 @@ enum mem_cgroup_zstat_index {
 };

 struct mem_cgroup_per_zone {
+ /*
+  * spin_lock to protect the per cgroup LRU
+  */
+ spinlock_t  lru_lock;
   struct list_head active_list;
   struct list_head inactive_list;
   unsigned long count[NR_MEM_CGROUP_ZSTAT];
@@ -127,10 +131,7 @@ struct mem_cgroup {
    * per zone LRU lists.
    */
   struct mem_cgroup_lru_info info;
- /*
-  * spin_lock to protect the per cgroup LRU
-  */
- spinlock_t lru_lock;
+
   unsigned long control_type; /* control RSS or RSS+Pagecache */
   int prev_priority; /* for recording reclaim priority */
   /*
@@ -412,15 +413,16 @@ int task_in_mem_cgroup(struct task_struc
   */
 void mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
- struct mem_cgroup *mem;
+ struct mem_cgroup_per_zone *mz;
+ unsigned long flags;
+
   if (!pc)
    return;

- mem = pc->mem_cgroup;
```

```
-
- spin_lock(&mem->lru_lock);
+ mz = page_cgroup_zoneinfo(pc);
+ spin_lock_irqsave(&mz->lru_lock, flags);
  __mem_cgroup_move_lists(pc, active);
- spin_unlock(&mem->lru_lock);
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
 }

 /*
@@ -528,7 +530,7 @@ unsigned long mem_cgroup_isolate_pages(u
   src = &mz->inactive_list;


- spin_lock(&mem_cont->lru_lock);
+ spin_lock(&mz->lru_lock);
  scan = 0;
  list_for_each_entry_safe_reverse(pc, tmp, src, lru) {
   if (scan >= nr_to_scan)
@@ -558,7 +560,7 @@ unsigned long mem_cgroup_isolate_pages(u
  }

  list_splice(&pc_list, src);
- spin_unlock(&mem_cont->lru_lock);
+ spin_unlock(&mz->lru_lock);

  *scanned = scan;
  return nr_taken;
@@ -577,6 +579,7 @@ static int mem_cgroup_charge_common(stru
  struct page_cgroup *pc;
  unsigned long flags;
  unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
+ struct mem_cgroup_per_zone *mz;

  /*
   * Should page_cgroup's go to their own slab?
@@ -688,10 +691,11 @@ noreclaim:
   goto retry;
  }

- spin_lock_irqsave(&mem->lru_lock, flags);
+ mz = page_cgroup_zoneinfo(pc);
+ spin_lock_irqsave(&mz->lru_lock, flags);
  /* Update statistics vector */
  __mem_cgroup_add_list(pc);
- spin_unlock_irqrestore(&mem->lru_lock, flags);
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
```

```
 done:
  return 0;
@@ -733,6 +737,7 @@ int mem_cgroup_cache_charge(struct page
 void mem_cgroup_uncharge(struct page_cgroup *pc)
 {
  struct mem_cgroup *mem;
+ struct mem_cgroup_per_zone *mz;
  struct page *page;
  unsigned long flags;

@@ -745,6 +750,7 @@ void mem_cgroup_uncharge(struct page_cgr

  if (atomic_dec_and_test(&pc->ref_cnt)) {
   page = pc->page;
+  mz = page_cgroup_zoneinfo(pc);
   /*
    * get page->cgroup and clear it under lock.
    * force_empty can drop page->cgroup without checking refcnt.
@@ -753,9 +759,9 @@ void mem_cgroup_uncharge(struct page_cgr
    mem = pc->mem_cgroup;
    css_put(&mem->css);
    res_counter_uncharge(&mem->res, PAGE_SIZE);
-   spin_lock_irqsave(&mem->lru_lock, flags);
+   spin_lock_irqsave(&mz->lru_lock, flags);
    __mem_cgroup_remove_list(pc);
-   spin_unlock_irqrestore(&mem->lru_lock, flags);
+   spin_unlock_irqrestore(&mz->lru_lock, flags);
    kfree(pc);
   }
  }
@@ -794,24 +800,29 @@ void mem_cgroup_page_migration(struct pa
  struct page_cgroup *pc;
  struct mem_cgroup *mem;
  unsigned long flags;
+ struct mem_cgroup_per_zone *mz;
 retry:
  pc = page_get_page_cgroup(page);
  if (!pc)
   return;
  mem = pc->mem_cgroup;
+ mz = page_cgroup_zoneinfo(pc);
  if (clear_page_cgroup(page, pc) != pc)
   goto retry;
-
- spin_lock_irqsave(&mem->lru_lock, flags);
+ spin_lock_irqsave(&mz->lru_lock, flags);

  __mem_cgroup_remove_list(pc);
```

```
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
+
  pc->page = newpage;
  lock_page_cgroup(newpage);
  page_assign_page_cgroup(newpage, pc);
  unlock_page_cgroup(newpage);
- __mem_cgroup_add_list(pc);

- spin_unlock_irqrestore(&mem->lru_lock, flags);
+ mz = page_cgroup_zoneinfo(pc);
+ spin_lock_irqsave(&mz->lru_lock, flags);
+ __mem_cgroup_add_list(pc);
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
  return;
 }

@@ -822,18 +833,26 @@ retry:
  */
 #define FORCE_UNCHARGE_BATCH (128)
 static void
-mem_cgroup_force_empty_list(struct mem_cgroup *mem, struct list_head *list)
+mem_cgroup_force_empty_list(struct mem_cgroup *mem,
+       struct mem_cgroup_per_zone *mz,
+       int active)
 {
  struct page_cgroup *pc;
  struct page *page;
  int count;
  unsigned long flags;
+ struct list_head *list;
+
+ if (active)
+  list = &mz->active_list;
+ else
+  list = &mz->inactive_list;

  if (list_empty(list))
   return;
 retry:
  count = FORCE_UNCHARGE_BATCH;
- spin_lock_irqsave(&mem->lru_lock, flags);
+ spin_lock_irqsave(&mz->lru_lock, flags);

  while (--count && !list_empty(list)) {
   pc = list_entry(list->prev, struct page_cgroup, lru);
@@ -850,7 +869,7 @@ retry:
   } else  /* being uncharged ? ...do relax */
    break;
```

```
  }
- spin_unlock_irqrestore(&mem->lru_lock, flags);
+ spin_unlock_irqrestore(&mz->lru_lock, flags);
  if (!list_empty(list)) {
   cond_resched();
   goto retry;
@@ -881,11 +900,9 @@ int mem_cgroup_force_empty(struct mem_cg
     struct mem_cgroup_per_zone *mz;
     mz = mem_cgroup_zoneinfo(mem, node, zid);
     /* drop all page_cgroup in active_list */
-    mem_cgroup_force_empty_list(mem,
-       &mz->active_list);
+    mem_cgroup_force_empty_list(mem, mz, 1);
     /* drop all page_cgroup in inactive_list */
-    mem_cgroup_force_empty_list(mem,
-       &mz->inactive_list);
+    mem_cgroup_force_empty_list(mem, mz, 0);
    }
  }
  ret = 0;
@@ -1112,6 +1129,7 @@ static int alloc_mem_cgroup_per_zone_inf
   mz = &pn->zoneinfo[zone];
   INIT_LIST_HEAD(&mz->active_list);
   INIT_LIST_HEAD(&mz->inactive_list);
+  spin_lock_init(&mz->lru_lock);
  }
  return 0;
 }
@@ -1136,7 +1154,6 @@ mem_cgroup_create(struct cgroup_subsys *

  res_counter_init(&mem->res);

- spin_lock_init(&mem->lru_lock);
  mem->control_type = MEM_CGROUP_TYPE_ALL;
  memset(&mem->info, 0, sizeof(mem->info));
```



Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [1/10] add scan_global_lru() macro
Posted by Balbir Singh on Sat, 17 Nov 2007 09:03:46 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> add macro scan_global_lru().
>
> This is used to detect which scan_control scans global lru or
> mem_cgroup lru. And compiled to be static value (1) when
> memory controller is not configured. (maybe good for compiler)
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

Looks good to me

Acked-by: Balbir Singh <balbir@linux.vnet.ibm.com>

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL



---

Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [2/10] add nid/zid function for page_cgro
Posted by Balbir Singh on Sat, 17 Nov 2007 09:15:45 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> mm/memcontrol.c | 10 ++++++++++
> 1 file changed, 10 insertions(+)
>
> Index: linux-2.6.24-rc2-mm1/mm/memcontrol.c
> ===================================================================
> --- linux-2.6.24-rc2-mm1.orig/mm/memcontrol.c
> +++ linux-2.6.24-rc2-mm1/mm/memcontrol.c
> @@ -135,6 +135,16 @@ struct page_cgroup {
> #define PAGE_CGROUP_FLAG_CACHE (0x1) /* charged as cache */
> #define PAGE_CGROUP_FLAG_ACTIVE (0x2) /* page is active in this cgroup */
>
> +static inline int page_cgroup_nid(struct page_cgroup *pc)
> +{
> + return page_to_nid(pc->page);
> +}
> +
> +static inline int page_cgroup_zid(struct page_cgroup *pc)

> +{
> + return page_zonenum(pc->page);

page_zonenum returns zone_type, isn't it better we carry the
type through to the caller?

> +}
> +
>  enum {
>   MEM_CGROUP_TYPE_UNSPEC = 0,
>   MEM_CGROUP_TYPE_MAPPED,
>

--
 Warm Regards,
 Balbir Singh
 Linux Technology Center
 IBM, ISTL

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

---

Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [3/10] add per zone active/inactive count
Posted by Balbir Singh on Sat, 17 Nov 2007 16:07:17 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> Counting active/inactive per-zone in memory controller.
>
> This patch adds per-zone status in memory cgroup.
> These values are often read (as per-zone value) by page reclaiming.
>
> In current design, per-zone stat is just a unsigned long value and
> not an atomic value because they are modified only under lru_lock.
> (for avoiding atomic_t ops.)
>
> This patch adds ACTIVE and INACTIVE per-zone status values.
>
> For handling per-zone status, this patch adds
>   struct mem_cgroup_per_zone {
>   ...
>   }
> and some helper functions. This will be useful to add per-zone objects
> in mem_cgroup.

>
> This patch turns memory controller's early_init to be 0 for calling
> kmalloc().
>
> Changelog V1 -> V2
>  - added mem_cgroup_per_zone struct.
>  This will help following patches to implement per-zone objects and
>  pack them into a struct.
>  - added __mem_cgroup_add_list() and __mem_cgroup_remove_list()
>  - fixed page migration handling.
>  - renamed zstat to info (per-zone-info)
>  This will be place for per-zone information(lru, lock, ..)
>  - use page_cgroup_nid()/zid() funcs.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

The code looks OK to me, pending test on a real NUMA box

Acked-by: Balbir Singh <balbir@linux.vnet.ibm.com>

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

---

## Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [4/10] calculate mapped ratio for memory
Posted by Balbir Singh on Sat, 17 Nov 2007 16:12:06 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> Define function for calculating mapped_ratio in memory cgroup.
>

Could you explain what the ratio is used for? Is it for reclaim
later?

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

>  include/linux/memcontrol.h |   11 ++++++++++-
>  mm/memcontrol.c            |   13 +++++++++++++
>  2 files changed, 23 insertions(+), 1 deletion(-)
>
> Index: linux-2.6.24-rc2-mm1/mm/memcontrol.c
> ===================================================================
> --- linux-2.6.24-rc2-mm1.orig/mm/memcontrol.c
> +++ linux-2.6.24-rc2-mm1/mm/memcontrol.c
> @@ -423,6 +423,19 @@ void mem_cgroup_move_lists(struct page_c
>   spin_unlock(&mem->lru_lock);
>  }
>
> +/*
> + * Calculate mapped_ratio under memory controller.
> + */
> +int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem)
> +{
> + s64 total, rss;
> +
> + /* usage is recorded in bytes */
> + total = mem->res.usage >> PAGE_SHIFT;
> + rss = mem_cgroup_read_stat(&mem->stat, MEM_CGROUP_STAT_RSS);
> + return (rss * 100) / total;

Never tried 64 bit division on a 32 bit system. I hope we don't
have to resort to do_div() sort of functionality.

> +}
> +
>  unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
>       struct list_head *dst,
>       unsigned long *scanned, int order,
> Index: linux-2.6.24-rc2-mm1/include/linux/memcontrol.h
> ===================================================================
> --- linux-2.6.24-rc2-mm1.orig/include/linux/memcontrol.h
> +++ linux-2.6.24-rc2-mm1/include/linux/memcontrol.h
> @@ -61,6 +61,12 @@ extern int mem_cgroup_prepare_migration(
>  extern void mem_cgroup_end_migration(struct page *page);
>  extern void mem_cgroup_page_migration(struct page *page, struct page *newpage);
>
> +/*
> + * For memory reclaim.
> + */
> +extern int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem);
> +
> +
>  #else /* CONFIG_CGROUP_MEM_CONT */
>  static inline void mm_init_cgroup(struct mm_struct *mm,

>       struct task_struct *p)
> @@ -132,7 +138,10 @@ mem_cgroup_page_migration(struct page *p
>  {
>  }
>
> -
> +static inline int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem)
> +{
> + return 0;
> +}
>  #endif /* CONFIG_CGROUP_MEM_CONT */
>
>  #endif /* _LINUX_MEMCONTROL_H */
>

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

---

Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [8/10] changes in vmscan.c
Posted by Balbir Singh on Sat, 17 Nov 2007 17:37:46 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> When using memory controller, there are 2 levels of memory reclaim.
>  1. zone memory reclaim because of system/zone memory shortage.
>  2. memory cgroup memory reclaim because of hitting limit.
>
> These two can be distinguished by sc->mem_cgroup parameter.
>
> This patch tries to make memory cgroup reclaim routine avoid affecting
> system/zone memory reclaim. This patch inserts if (!sc->mem_cgroup) and
> hook to memory_cgroup reclaim support functions.
>
> This patch can be a help for isolating system lru activity and group lru
> activity and shows what additional functions are necessary.
>
>  * mem_cgroup_calc_mapped_ratio() ... calculate mapped ratio for cgroup.
>  * mem_cgroup_reclaim_imbalance() ... calculate active/inactive balance in

> cgroup.
> * mem_cgroup_calc_reclaim_active() ... calculate the number of active pages to
> be scanned in this priority in mem_cgroup.
>
> * mem_cgroup_calc_reclaim_inactive() ... calculate the number of inactive pages
> to be scanned in this priority in mem_cgroup.
>
> * mem_cgroup_all_unreclaimable() .. checks cgroup's page is all unreclaimable
> or not.
> * mem_cgroup_get_reclaim_priority() ...
> * mem_cgroup_note_reclaim_priority() ... record reclaim priority (temporal)
> * mem_cgroup_remember_reclaim_priority()
> .... record reclaim priority as
> zone->prev_priority.
> This value is used for calc reclaim_mapped.
> Changelog:
> - merged calc_reclaim_mapped patch in previous version.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

The overall idea looks good, it brings the two reclaims closer. The one
pending to do for memory controllers is to make the reclaim lumpy
reclaim aware. But at this point, I don't see a need for it, since
we track only order 1 allocations in the memory controller.

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

---

Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [9/10] per-zone-lru for memory cgroup
Posted by Balbir Singh on Sat, 17 Nov 2007 17:51:22 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> This patch implements per-zone lru for memory cgroup.
> This patch makes use of mem_cgroup_per_zone struct for per zone lru.
>
> LRU can be accessed by
>
> mz = mem_cgroup_zoneinfo(mem_cgroup, node, zone);

>  &mz->active_list
>  &mz->inactive_list
>
>  or
>  mz = page_cgroup_zoneinfo(page_cgroup, node, zone);
>  &mz->active_list
>  &mz->inactive_list
>
>
> Changelog v1->v2
>  - merged to mem_cgroup_per_zone struct.
>  - handle page migraiton.
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>

Thanks, this has been a long pending TODO. What is pending now on my
plate is re-organizing res_counter to become aware of the filesystem
hierarchy. I want to split out the LRU lists from the memory controller
and resource counters.

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

---

Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [2/10] add nid/zid function for page_cgro
Posted by KAMEZAWA Hiroyuki on Mon, 19 Nov 2007 01:37:20 GMT
View Forum Message <> Reply to Message

On Sat, 17 Nov 2007 14:45:45 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> KAMEZAWA Hiroyuki wrote:
> > Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> >  mm/memcontrol.c |  10 ++++++++++
> >  1 file changed, 10 insertions(+)
> >
> > Index: linux-2.6.24-rc2-mm1/mm/memcontrol.c
> > ===================================================================
> > --- linux-2.6.24-rc2-mm1.orig/mm/memcontrol.c

> > +++ linux-2.6.24-rc2-mm1/mm/memcontrol.c
> > @@ -135,6 +135,16 @@ struct page_cgroup {
> >  #define PAGE_CGROUP_FLAG_CACHE (0x1) /* charged as cache */
> >  #define PAGE_CGROUP_FLAG_ACTIVE (0x2) /* page is active in this cgroup */
> >
> > +static inline int page_cgroup_nid(struct page_cgroup *pc)
> > +{
> > + return page_to_nid(pc->page);
> > +}
> > +
> > +static inline int page_cgroup_zid(struct page_cgroup *pc)
> > +{
> > + return page_zonenum(pc->page);
>
> page_zonenum returns zone_type, isn't it better we carry the
> type through to the caller?
>
seems resonable. ok. I will fix.

-Kame

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

---

Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [4/10] calculate mapped ratio for memory
Posted by KAMEZAWA Hiroyuki on Mon, 19 Nov 2007 01:41:44 GMT
View Forum Message <> Reply to Message

On Sat, 17 Nov 2007 21:42:06 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> KAMEZAWA Hiroyuki wrote:
> > Define function for calculating mapped_ratio in memory cgroup.
> >
>
> Could you explain what the ratio is used for? Is it for reclaim
> later?
>
Yes, for later.

> > + /* usage is recorded in bytes */
> > + total = mem->res.usage >> PAGE_SHIFT;
> > + rss = mem_cgroup_read_stat(&mem->stat, MEM_CGROUP_STAT_RSS);

> > + return (rss * 100) / total;
>
> Never tried 64 bit division on a 32 bit system. I hope we don't
> have to resort to do_div() sort of functionality.
>
Hmm, maybe it's better to make these numebrs be just "long".
I'll try to change per-cpu-counter implementation.

Thanks,
-Kame

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

---

Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [9/10] per-zone-lru for memory cgroup
Posted by KAMEZAWA Hiroyuki on Mon, 19 Nov 2007 01:47:09 GMT
View Forum Message <> Reply to Message

On Sat, 17 Nov 2007 23:21:22 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> Thanks, this has been a long pending TODO. What is pending now on my
> plate is re-organizing res_counter to become aware of the filesystem
> hierarchy. I want to split out the LRU lists from the memory controller
> and resource counters.
>
Does "file system hierarchy" here means "control group hierarchy" ?
like
=
/cgroup/group_A/group_A_1
. /group_A_2
/group_A_3
(LRU(s) will be used for maintaining parent/child groups.)

Thanks,
-Kame

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

---

Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [9/10] per-zone-lru for memory cgroup
Posted by Balbir Singh on Mon, 19 Nov 2007 06:21:15 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> On Sat, 17 Nov 2007 23:21:22 +0530
> Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>
>> Thanks, this has been a long pending TODO. What is pending now on my
>> plate is re-organizing res_counter to become aware of the filesystem
>> hierarchy. I want to split out the LRU lists from the memory controller
>> and resource counters.
>>
> Does "file system hierarchy" here means "control group hierarchy" ?
> like

Yes, you are right

> =
> /cgroup/group_A/group_A_1
>  . /group_A_2
>  /group_A_3
> (LRU(s) will be used for maintaining parent/child groups.)
>

The LRU's will be shared, my vision is

```
 LRU
 ^ ^
 | |
Mem-----+ +----Mem
```

That two or more mem_cgroup's can refer to the same LRU list and have
their own resource counters. This setup will be used in the case
of a hierarchy, so that a child can share memory with its parent
and have it's own limit.

The mem_cgroup will basically then only contain a reference
to the LRU list.

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

_____
Containers mailing list

Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [9/10] per-zone-lru for memory cgroup
Posted by KAMEZAWA Hiroyuki on Mon, 19 Nov 2007 06:35:49 GMT
View Forum Message <> Reply to Message

On Mon, 19 Nov 2007 11:51:15 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> > =
> > /cgroup/group_A/group_A_1
> > . /group_A_2
> > /group_A_3
> > (LRU(s) will be used for maintaining parent/child groups.)
> >
>
> The LRU's will be shared, my vision is
>
> LRU
> ^ ^
> | |
> Mem-----+ +----Mem
>
>
> That two or more mem_cgroup's can refer to the same LRU list and have
> their own resource counters. This setup will be used in the case
> of a hierarchy, so that a child can share memory with its parent
> and have it's own limit.
>
> The mem_cgroup will basically then only contain a reference
> to the LRU list.
>
Hmm, interesting.

Then,
  group_A_1's usage + group_A_2's usage + group_A_3's usgae < group_A's limit.
  group_A_1, group_A_2, group_A_3 has its own limit.
In plan.

I wonder if we want rich control functions, we need "share" or "priority" among
childs. (but maybe this will be complicated one.)

Thank you for explanation.

Regards,
-Kame

Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [9/10] per-zone-lru for memory cgroup
Posted by Balbir Singh on Mon, 19 Nov 2007 08:34:23 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> On Mon, 19 Nov 2007 11:51:15 +0530
> Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>>> =
>>> /cgroup/group_A/group_A_1
>>> . /group_A_2
>>> /group_A_3
>>> (LRU(s) will be used for maintaining parent/child groups.)
>>>
>> The LRU's will be shared, my vision is
>>
>> LRU
>> ^ ^
>> | |
>> Mem-----+ +----Mem
>>
>>
>> That two or more mem_cgroup's can refer to the same LRU list and have
>> their own resource counters. This setup will be used in the case
>> of a hierarchy, so that a child can share memory with its parent
>> and have it's own limit.
>>
>> The mem_cgroup will basically then only contain a reference
>> to the LRU list.
>>
> Hmm, interesting.
>
> Then,
> group_A_1's usage + group_A_2's usage + group_A_3's usgae < group_A's limit.
> group_A_1, group_A_2, group_A_3 has its own limit.

Yes that is correct

> In plan.
>
> I wonder if we want rich control functions, we need "share" or "priority" among

> childs. (but maybe this will be complicated one.)
>

That would nice and the end goal of providing this feature. We also need
to provide soft-limits (more complex) and guarantees (with the kernel
memory controller coming in - nice to have, but not necessary for now)

> Thank you for explanation.
>
> Regards,
> -Kame
>

Thanks for helping out the memory controller.

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

---

Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [4/10] calculate mapped ratio for memory
Posted by yamamoto on Thu, 22 Nov 2007 08:34:20 GMT
View Forum Message <> Reply to Message

> > > + /* usage is recorded in bytes */
> > > + total = mem->res.usage >> PAGE_SHIFT;
> > > + rss = mem_cgroup_read_stat(&mem->stat, MEM_CGROUP_STAT_RSS);
> > > + return (rss * 100) / total;
> >
> > Never tried 64 bit division on a 32 bit system. I hope we don't
> > have to resort to do_div() sort of functionality.
> >
> Hmm, maybe it's better to make these numebrs be just "long".
> I'll try to change per-cpu-counter implementation.
>
> Thanks,
> -Kame

besides that, i think 'total' can be zero here.

YAMAMOTO Takashi

Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [4/10] calculate mapped ratio for memory
Posted by KAMEZAWA Hiroyuki on Thu, 22 Nov 2007 08:40:15 GMT
View Forum Message <> Reply to Message

On Thu, 22 Nov 2007 17:34:20 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> > > > + /* usage is recorded in bytes */
> > > > + total = mem->res.usage >> PAGE_SHIFT;
> > > > + rss = mem_cgroup_read_stat(&mem->stat, MEM_CGROUP_STAT_RSS);
> > > > + return (rss * 100) / total;
> > >
> > > Never tried 64 bit division on a 32 bit system. I hope we don't
> > > have to resort to do_div() sort of functionality.
> > >
> > Hmm, maybe it's better to make these numebrs be just "long".
> > I'll try to change per-cpu-counter implementation.
> >
> > Thanks,
> > -Kame
>
> besides that, i think 'total' can be zero here.
>
Ah, This is what I do now.
==
+/*
+ * Calculate mapped_ratio under memory controller. This will be used in
+ * vmscan.c for deteremining we have to reclaim mapped pages.
+ */
+int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem)
+{
+	long total, rss;
+
+	/*
+	 * usage is recorded in bytes. But, here, we assume the number of
+	 * physical pages can be represented by "long" on any arch.
+	 */
+	total = (long) (mem->res.usage >> PAGE_SHIFT);
+	rss = (long)mem_cgroup_read_stat(&mem->stat, MEM_CGROUP_STAT_RSS);
+	return (int)((rss * 100L) / total);
+}

==

maybe works well.

-Kame



---

Subject: Re: [RFC][PATCH] memory controller per zone patches take 2 [4/10] calculate mapped ratio for memory
Posted by yamamoto on Thu, 22 Nov 2007 08:46:47 GMT
View Forum Message <> Reply to Message

> On Thu, 22 Nov 2007 17:34:20 +0900 (JST)
> yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
>
> > > > > > + /* usage is recorded in bytes */
> > > > > > + total = mem->res.usage >> PAGE_SHIFT;
> > > > > > + rss = mem_cgroup_read_stat(&mem->stat, MEM_CGROUP_STAT_RSS);
> > > > > > + return (rss * 100) / total;
> > > > >
> > > > Never tried 64 bit division on a 32 bit system. I hope we don't
> > > > have to resort to do_div() sort of functionality.
> > > >
> > > Hmm, maybe it's better to make these numebrs be just "long".
> > > I'll try to change per-cpu-counter implementation.
> > >
> > > Thanks,
> > > -Kame
> >
> > besides that, i think 'total' can be zero here.
> >
> Ah, This is what I do now.
> ==
> +/*
> + * Calculate mapped_ratio under memory controller. This will be used in
> + * vmscan.c for deteremining we have to reclaim mapped pages.
> + */
> +int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem)
> +{
> +	long total, rss;
> +
> +	/*
> +	 * usage is recorded in bytes. But, here, we assume the number of

> +  * physical pages can be represented by "long" on any arch.
> +  */
> +  total = (long) (mem->res.usage >> PAGE_SHIFT);
> +  rss = (long)mem_cgroup_read_stat(&mem->stat, MEM_CGROUP_STAT_RSS);
> +  return (int)((rss * 100L) / total);
> +}
> ==
>
> maybe works well.
>
> -Kame

i meant that "/ total" can cause a division-by-zero exception.

YAMAMOTO Takashi

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

---

Subject: Re: Re: [RFC][PATCH] memory controller per zone patches take 2 [4/10] calculate mapped ratio for mem
Posted by KAMEZAWA Hiroyuki on Thu, 22 Nov 2007 13:31:10 GMT
View Forum Message <> Reply to Message

>> Ah, This is what I do now.
>> ==
>> +/*
>> + * Calculate mapped_ratio under memory controller. This will be used in
>> + * vmscan.c for deteremining we have to reclaim mapped pages.
>> + */
>> +int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem)
>> +{
>> +  long total, rss;
>> +
>> +  /*
>> +  * usage is recorded in bytes. But, here, we assume the number of
>> +  * physical pages can be represented by "long" on any arch.
>> +  */
>> +  total = (long) (mem->res.usage >> PAGE_SHIFT);
>> +  rss = (long)mem_cgroup_read_stat(&mem->stat, MEM_CGROUP_STAT_RSS);
>> +  return (int)((rss * 100L) / total);
>> +}
>> ==
>>
>> maybe works well.
>>

>> -Kame
>
>i meant that "/ total" can cause a division-by-zero exception.
>
ouch, ok, will fix.

-Kame

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