Subject: [RFC][for -mm] memory controller enhancements for NUMA [0/10] introduction
Posted by KAMEZAWA Hiroyuki on Wed, 14 Nov 2007 08:39:50 GMT
View Forum Message <> Reply to Message

Hi,

This is a patch-set for memory controlelr on NUMA.
patches are

1. record nid/zid on page_cgroup struct
2. record per-zone active/inactive
3-9 Patches for isolate global-lru reclaiming and memory controller reclaming
10. implements per-zone LRU on memory controller.

now this is just RFC.

Tested on
  2.6.24-rc2-mm1 + x86_64/fake-NUMA(# of nodes = 3)

I did test with numactl under memory limitation.
 % numactl -i 0,1,2 dd if=.....

It seems per-zone-lru works well.

I'd like to do test on ia64/real-NUMA when I have a chance.

Any comments are welcome.

Thanks,
 -kame

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

Subject: [RFC][for -mm] memory controller enhancements for NUMA [2/10] account active inactive
Posted by KAMEZAWA Hiroyuki on Wed, 14 Nov 2007 08:42:33 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.

If we need to modify value withou taking lock, we'll have to use
atomic64_t.

This patch adds TOTAL and INACTIVE values.

 TOTAL .... the number of pages of zone in memory cgroup.
 INACTIVE .... the number of inactive pages of zone in memory cgroup.

the number of acitve pages is TOTAL - INACTIVE.

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

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

 mm/memcontrol.c | 131
++-
 1 file changed, 130 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
@@ -78,6 +78,30 @@ static s64 mem_cgroup_read_stat(struct m
 }

 /*
+ * some of parameters used for page reclaiming should be counted per zone.
+ * Its array size will be MAX_NUMNODES * MAX_NR_ZONES * # of params.
+ * This seems a bit big for per_cpu....
+ *
+ * Values are modified under mem_cgroup->lru_lock. (now)
+ */
+
+enum mem_cgroup_zstat_index {
+ MEM_CGROUP_ZSTAT_TOTAL,
+ MEM_CGROUP_ZSTAT_INACTIVE, /* Active = Total - Inactive */
+
+ NR_MEM_CGROUP_ZSTAT,
+};
+
+struct mem_cgroup_stat_zone {

```
+ unsigned long count[MAX_NR_ZONES][NR_MEM_CGROUP_ZSTAT];
+};
+
+struct mem_cgroup_zonestat {
+ struct mem_cgroup_stat_zone *nodestat[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,
@@ -110,6 +134,10 @@ struct mem_cgroup {
   * statistics.
   */
  struct mem_cgroup_stat stat;
+ /*
+  * Per zone statistics (used for memory reclaim)
+  */
+ struct mem_cgroup_zonestat zstat;
 };

 /*
@@ -168,6 +196,52 @@ static void mem_cgroup_charge_statistics

 }

+/*
+ * Always used with page_cgroup pointer. (now)
+ */
+static void mem_cgroup_add_zonestat(struct page_cgroup *pc,
+   enum mem_cgroup_zstat_index idx, int val)
+{
+ struct mem_cgroup *mem = pc->mem_cgroup;
+ struct mem_cgroup_stat_zone *zs = mem->zstat.nodestat[pc->nid];
+
+ BUG_ON(!zs);
+
+ zs->count[pc->zid][idx] += val;
+}
+
+static void mem_cgroup_sub_zonestat(struct page_cgroup *pc,
+   enum mem_cgroup_zstat_index idx, int val)
+{
+ struct mem_cgroup *mem = pc->mem_cgroup;
+ struct mem_cgroup_stat_zone *zs = mem->zstat.nodestat[pc->nid];
+
+ BUG_ON(!zs);
```

```
+
+ zs->count[pc->zid][idx] -= val;
+}
+
+static u64 mem_cgroup_get_zonestat(struct mem_cgroup *mem,
+     int nid, int zid,
+     enum mem_cgroup_zstat_index idx)
+{
+ struct mem_cgroup_stat_zone *zs = mem->zstat.nodestat[nid];
+ if (!zs)
+  return 0;
+ return zs->count[zid][idx];
+}
+
+static unsigned long mem_cgroup_get_all_zonestat(struct mem_cgroup *mem,
+     enum mem_cgroup_zstat_index idx)
+{
+ int nid, zid;
+ u64 total = 0;
+ for_each_online_node(nid)
+  for (zid = 0; zid < MAX_NR_ZONES; zid++)
+   total += mem_cgroup_get_zonestat(mem, nid, zid, idx);
+ return total;
+}
+
 static struct mem_cgroup init_mem_cgroup;

 static inline
@@ -284,6 +358,13 @@ static struct page_cgroup *clear_page_cg

 static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
+ int from = pc->flags & PAGE_CGROUP_FLAG_ACTIVE;
+
+ if (from && !active) /* active -> inactive */
+  mem_cgroup_add_zonestat(pc, MEM_CGROUP_ZSTAT_INACTIVE, 1);
+ else if (!from & active) /* inactive to active */
+  mem_cgroup_sub_zonestat(pc, MEM_CGROUP_ZSTAT_INACTIVE, 1);
+
  if (active) {
   pc->flags |= PAGE_CGROUP_FLAG_ACTIVE;
   list_move(&pc->lru, &pc->mem_cgroup->active_list);
@@ -493,6 +574,7 @@ noreclaim:
  pc->flags = PAGE_CGROUP_FLAG_ACTIVE;
  if (ctype == MEM_CGROUP_CHARGE_TYPE_CACHE)
   pc->flags |= PAGE_CGROUP_FLAG_CACHE;
+
  if (page_cgroup_assign_new_page_cgroup(page, pc)) {
```

```
   /*
    * an another charge is added to this page already.
@@ -506,6 +588,7 @@ noreclaim:
  }

  spin_lock_irqsave(&mem->lru_lock, flags);
+ mem_cgroup_add_zonestat(pc, MEM_CGROUP_ZSTAT_TOTAL, 1);
  /* Update statistics vector */
  mem_cgroup_charge_statistics(mem, pc->flags, true);
  list_add(&pc->lru, &mem->active_list);
@@ -574,11 +657,16 @@ void mem_cgroup_uncharge(struct page_cgr
    spin_lock_irqsave(&mem->lru_lock, flags);
    list_del_init(&pc->lru);
    mem_cgroup_charge_statistics(mem, pc->flags, false);
+   if (!(pc->flags & PAGE_CGROUP_FLAG_ACTIVE))
+    mem_cgroup_sub_zonestat(pc,
+      MEM_CGROUP_ZSTAT_INACTIVE, 1);
+   mem_cgroup_sub_zonestat(pc, MEM_CGROUP_ZSTAT_TOTAL, 1);
    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.
@@ -647,10 +735,16 @@ 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);
+   if (!(pc->flags & PAGE_CGROUP_FLAG_ACTIVE))
+    mem_cgroup_sub_zonestat(pc,
+      MEM_CGROUP_ZSTAT_INACTIVE, 1);
+   mem_cgroup_sub_zonestat(pc, MEM_CGROUP_ZSTAT_TOTAL, 1);
    kfree(pc);
   } else  /* being uncharged ? ...do relax */
    break;
@@ -829,6 +923,17 @@ 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 total, inactive;
+
+  inactive = mem_cgroup_get_all_zonestat(mem_cont,
+      MEM_CGROUP_ZSTAT_INACTIVE);
+  total = mem_cgroup_get_all_zonestat(mem_cont,
+      MEM_CGROUP_ZSTAT_TOTAL);
+  seq_printf(m, "active %ld\n", (total - inactive) * PAGE_SIZE);
+  seq_printf(m, "inactive %ld\n", (inactive) * PAGE_SIZE);
+ }
  return 0;
 }

@@ -888,6 +993,7 @@ 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;
@@ -903,7 +1009,24 @@ 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->zstat, 0, sizeof(mem->zstat));
+
+ for_each_node_state(node, N_POSSIBLE) {
+  int size = sizeof(struct mem_cgroup_stat_zone);
+
+  mem->zstat.nodestat[node] =
+    kmalloc_node(size, GFP_KERNEL, node);
+  if (mem->zstat.nodestat[node] == NULL)
+   goto free_out;
+  memset(mem->zstat.nodestat[node], 0, size);
+ }
  return &mem->css;
+free_out:
+ for_each_node_state(node, N_POSSIBLE)
+  kfree(mem->zstat.nodestat[node]);
+ if (cont->parent != NULL)
+  kfree(mem);
+ return NULL;
 }

 static void mem_cgroup_pre_destroy(struct cgroup_subsys *ss,
@@ -916,6 +1039,12 @@ 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->zstat.nodestat[node]);
+
  kfree(mem_cgroup_from_cont(cont));
 }

@@ -968,5 +1097,5 @@ 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

---

Subject: [RFC][for -mm] memory controller enhancements for NUMA [3/10] define macro for global lru scan

Posted by KAMEZAWA Hiroyuki on Wed, 14 Nov 2007 08:43:52 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.

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:
  /*
```

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

---

Subject: [RFC][for -mm] memory controller enhancements for NUMA [4/10] calc mapped_ratio in cgroup
Posted by KAMEZAWA Hiroyuki on Wed, 14 Nov 2007 08:45:34 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
@@ -400,6 +400,19 @@ void mem_cgroup_move_lists(struct page_c
  spin_unlock(&mem->lru_lock);
 }

+/*
+ * Calclate mapped_ratio under memory controller.
+ */
+int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem)
+{
+ s64 total, rss;
+
+ /* usage is recoreded 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 */
```

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

---

Subject: [RFC][for -mm] memory controller enhancements for NUMA [5/10] calc active/inactive impabance

Posted by KAMEZAWA Hiroyuki on Wed, 14 Nov 2007 08:48:28 GMT

View Forum Message <> Reply to Message

Define function for determining active/inactive balance in memory
cgroup. This imbalance value is calculated against the whole zone..

maybe should be per-zone...?

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
@@ -412,6 +412,22 @@ 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 detemining 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 recoreded 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 */
```

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

Subject: [RFC][for -mm] memory controller enhancements for NUMA [6/10] record reclaim priority
Posted by KAMEZAWA Hiroyuki on Wed, 14 Nov 2007 08:50:19 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
@@ -125,6 +125,7 @@ struct mem_cgroup {
   */
  struct list_head active_list;
  struct list_head inactive_list;
+ int    prev_priority; /* used in memory reclaim (see zone's one)*/
  /*
   * spin_lock to protect the per cgroup LRU
   */
@@ -429,6 +430,25 @@ 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 */
```

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

Subject: [RFC][for -mm] memory controller enhancements for NUMA [7/10] calc scan numbers per zone
Posted by KAMEZAWA Hiroyuki on Wed, 14 Nov 2007 08:51:29 GMT
View Forum Message <> Reply to Message

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

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

```
 include/linux/memcontrol.h |   15 +++++++++++++++
 mm/memcontrol.c            |   40 ++++++++++++++++++++++++++++++++++++++++
 2 files changed, 55 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
@@ -449,6 +449,46 @@ void mem_cgroup_record_reclaim_priority(
  mem->prev_priority = priority;
 }

+/*
+ * Calculate # of pages to be scanned in this proirity/zone.
+ * See also vmscan.c
+ *
+ * priority statrs 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, total, inactive;
+ int nid = zone->zone_pgdat->node_id;
+ int zid = zone_idx(zone);
+
+ total = mem_cgroup_get_zonestat(mem, nid, zid, MEM_CGROUP_ZSTAT_TOTAL);
+ inactive = mem_cgroup_get_zonestat(mem, nid, zid,
+      MEM_CGROUP_ZSTAT_INACTIVE);
+ nr_active = total - inactive;
+ /*
+  * FIXME: on NUMA, returning 0 here is not good ?
+  */
+ return (long)(nr_active >> priority);
+}
+
+long mem_cgroup_calc_reclaim_inactive(struct mem_cgroup *mem,
+     struct zone *zone, int priority)
+{
+ s64 nr_inactive;
+ int nid = zone->zone_pgdat->node_id;
+ int zid = zone_idx(zone);
+
+ nr_inactive = mem_cgroup_get_zonestat(mem, nid, zid,
+      MEM_CGROUP_ZSTAT_INACTIVE);
+ /*
+  * FIXME: on NUMA, returning 0 here is not good ?
+  */
+ 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,
```

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

Subject: [RFC][for -mm] memory controller enhancements for NUMA [8/10] move reclaim_mapped calc routine (cle
Posted by KAMEZAWA Hiroyuki on Wed, 14 Nov 2007 08:53:06 GMT
View Forum Message <> Reply to Message

Just for clean up for later patch for avoiding dirty nesting....

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

```
 mm/vmscan.c |  184 ++++++++++++++++++++++++++++++-----------------------------
 1 file changed, 97 insertions(+), 87 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
@@ -950,6 +950,98 @@ static inline int zone_is_near_oom(struc
 }

 /*
+ * Determine we should try to reclaim mapped pages.
+ */
+static int calc_reclaim_mapped(struct zone *zone, int priority, int swappiness)
+{
+ long mapped_ratio;
+ long distress;
+ long swap_tendency;
+ long imbalance;
+ int reclaim_mapped;
+
+ if (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.
+  */
+ 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 + 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)
+  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 +1058,8 @@ 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)
 {
@@ -979,93 +1073,9 @@ 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(zone, priority,
+          sc->swappiness);

  lru_add_drain();
  spin_lock_irq(&zone->lru_lock);
```

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

Subject: [RFC][for -mm] memory controller enhancements for NUMA [9/10] zone reclaim vs. cgroup reclaim isola
Posted by KAMEZAWA Hiroyuki on Wed, 14 Nov 2007 08:55:28 GMT
View Forum Message <> Reply to Message

When using memory controller, there are 2 levels of memory reclaim.
 1. zone memory relcaim 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.
(we can use macro "scan_global_lru")

This patch tries to make memory cgroup reclaim routine avoid affecting
system/zone memory reclaim. This patch inserts if (scan_global_lru(sc)) 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() ... calclate the number of active pages to
   be scanned in this priority in mem_cgroup.

 * mem_cgroup_calc_reclaim_inactive() ... calclate 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.

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

 mm/vmscan.c | 155 ++----------------
 1 file changed, 105 insertions(+), 50 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;
@@ -951,22 +952,30 @@ 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 zone *zone, int priority, int swappiness)
+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 (zone_is_near_oom(zone))
+ 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.
   */
- distress = 100 >> min(zone->prev_priority, priority);
+ 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
@@ -974,9 +983,13 @@ static int calc_reclaim_mapped(struct zo
   * how much memory
   * is mapped.
   */
- mapped_ratio = ((global_page_state(NR_FILE_MAPPED) +
-   global_page_state(NR_ANON_PAGES)) * 100) /
-    vm_total_pages;
+ 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
@@ -989,7 +1002,7 @@ static int calc_reclaim_mapped(struct zo
   * A 100% value of vm_swappiness overrides this algorithm
   * altogether.
   */
- swap_tendency = mapped_ratio / 2 + distress + swappiness;
+ swap_tendency = mapped_ratio / 2 + distress + sc->swappiness;

  /*
   * If there's huge imbalance between active and inactive
@@ -1003,8 +1016,11 @@ static int calc_reclaim_mapped(struct zo
   * 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;
+ 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,
@@ -1074,15 +1090,20 @@ static void shrink_active_list(unsigned
  int reclaim_mapped = 0;

  if (sc->may_swap)
-  reclaim_mapped = calc_reclaim_mapped(zone, priority,
-          sc->swappiness);
+  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);

@@ -1175,25 +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 relcaim 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) {
@@ -1238,6 +1273,7 @@ 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];
@@ -1247,16 +1283,26 @@ static unsigned long shrink_zones(int pr

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

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

```
+  }

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

@@ -1285,15 +1331,19 @@ 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--) {
@@ -1350,14 +1400,19 @@ 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;
 }
```

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

---

Subject: [RFC][for -mm] memory controller enhancements for NUMA [10/10] per-zone-lru
Posted by KAMEZAWA Hiroyuki on Wed, 14 Nov 2007 08:57:37 GMT
View Forum Message <> Reply to Message

I think there is a consensus that memory controller needs per-zone lru.
But it was postphoned that it seems some people tries to modify lru of zones.

Now, "scan" value, in mem_cgroup_isolate_pages(), handling is fixed. So,
demand of implementing per-zone-lru is raised.

This patch implements per-zone lru for memory cgroup.
I think this patch's style implementation can be adjusted to zone's lru
implementation changes if happens.

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

```
 mm/memcontrol.c |  117 ++++++++++++++++++++++++++++++++++++++++++++++---------
 1 file changed, 100 insertions(+), 17 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
@@ -101,6 +101,11 @@ struct mem_cgroup_zonestat {
 };
```

```
+struct mc_lru_head {
+ struct list_head active_list[MAX_NR_ZONES];
+ struct list_head inactive_list[MAX_NR_ZONES];
+};
+
 /*
  * The memory controller data structure. The memory controller controls both
  * page cache and RSS per cgroup. We would eventually like to provide
@@ -123,8 +128,8 @@ struct mem_cgroup {
   * per zone LRU lists.
   * TODO: Consider making these lists per zone
   */
- struct list_head active_list;
- struct list_head inactive_list;
+ struct mc_lru_head *lrus[MAX_NUMNODES];
+
  int    prev_priority; /* used in memory reclaim (see zone's one)*/
  /*
   * spin_lock to protect the per cgroup LRU
@@ -139,8 +144,20 @@ struct mem_cgroup {
   * Per zone statistics (used for memory reclaim)
   */
  struct mem_cgroup_zonestat zstat;
+#ifndef CONFIG_NUMA
+ struct lru_head local_head;
+#endif
 };

+struct list_head *mem_cgroup_lru_head(struct mem_cgroup *mem, int nid, int zid,
+     int active)
+{
+ if (active)
+  return &mem->lrus[nid]->active_list[zid];
+ else
+  return &mem->lrus[nid]->inactive_list[zid];
+}
+
 /*
  * We use the lower bit of the page->page_cgroup pointer as a bit spin
  * lock. We need to ensure that page->page_cgroup is atleast two
@@ -360,6 +377,9 @@ static struct page_cgroup *clear_page_cg
 static void __mem_cgroup_move_lists(struct page_cgroup *pc, bool active)
 {
  int from = pc->flags & PAGE_CGROUP_FLAG_ACTIVE;
+ struct list_head *head;
+
+ head = mem_cgroup_lru_head(pc->mem_cgroup, pc->nid, pc->zid, active);
```

```
  if (from && !active) /* active -> inactive */
   mem_cgroup_add_zonestat(pc, MEM_CGROUP_ZSTAT_INACTIVE, 1);
@@ -368,10 +388,10 @@ static void __mem_cgroup_move_lists(stru

  if (active) {
   pc->flags |= PAGE_CGROUP_FLAG_ACTIVE;
-  list_move(&pc->lru, &pc->mem_cgroup->active_list);
+  list_move(&pc->lru, head);
  } else {
   pc->flags &= ~PAGE_CGROUP_FLAG_ACTIVE;
-  list_move(&pc->lru, &pc->mem_cgroup->inactive_list);
+  list_move(&pc->lru, head);
  }
 }

@@ -502,11 +522,10 @@ 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);

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

  spin_lock(&mem_cont->lru_lock);
  scan = 0;
@@ -564,6 +583,7 @@ static int mem_cgroup_charge_common(stru
  struct page_cgroup *pc;
  unsigned long flags;
  unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
+ struct list_head *head;

  /*
   * Should page_cgroup's go to their own slab?
@@ -676,11 +696,12 @@ noreclaim:
   goto retry;
  }

+ head = mem_cgroup_lru_head(mem, pc->nid, pc->zid, 1);
  spin_lock_irqsave(&mem->lru_lock, flags);
  mem_cgroup_add_zonestat(pc, MEM_CGROUP_ZSTAT_TOTAL, 1);
  /* Update statistics vector */
  mem_cgroup_charge_statistics(mem, pc->flags, true);
```

```
- list_add(&pc->lru, &mem->active_list);
+ list_add(&pc->lru, head);
  spin_unlock_irqrestore(&mem->lru_lock, flags);

 done:
@@ -814,6 +835,8 @@ 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);
@@ -854,20 +877,26 @@ 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 list_head *head;
+    /* drop all page_cgroup in active_list */
+    head = mem_cgroup_lru_head(mem, node, zid, 1);
+    mem_cgroup_force_empty_list(mem, head);
+    head = mem_cgroup_lru_head(mem, node, zid, 0);
+    /* drop all page_cgroup in inactive_list */
+    mem_cgroup_force_empty_list(mem, head);
+   }
  }
  ret = 0;
 out:
@@ -1076,6 +1105,56 @@ static struct cftype mem_cgroup_files[]
```

```
 },
};

+
+static inline void __memory_cgroup_init_lru_head(struct mc_lru_head *head)
+{
+ int zone;
+ for (zone = 0; zone < MAX_NR_ZONES; zone++) {
+  INIT_LIST_HEAD(&head->active_list[zone]);
+  INIT_LIST_HEAD(&head->inactive_list[zone]);
+ }
+}
+
+#ifdef CONFIG_NUMA
+/*
+ * Returns 0 if success.
+ */
+static int mem_cgroup_init_lru(struct mem_cgroup *mem)
+{
+ int node;
+ struct mc_lru_head *head;
+
+ for_each_node_state(node, N_POSSIBLE) {
+  head = kmalloc_node(sizeof(*head), GFP_KERNEL, node);
+  if (!head)
+   return 1;
+  mem->lrus[node] = head;
+  __memory_cgroup_init_lru_head(head);
+ }
+ return 0;
+}
+
+static void mem_cgroup_free_lru(struct mem_cgroup *mem)
+{
+ int node;
+
+ for_each_node_state(node, N_POSSIBLE)
+  kfree(mem->lrus[node]);
+}
+#else
+
+static int mem_cgroup_init_lru(struct mem_cgroup *mem)
+{
+ int zone;
+ mem->lrus[0] = &mem->local_lru;
+ __memory_cgroup_init_lru_head(mem->lrus[0]);
+}
+
```

```
+static void mem_cgroup_free_lru(struct mem_cgroup *mem)
+{
+}
+#endif
+
 static struct mem_cgroup init_mem_cgroup;

 static struct cgroup_subsys_state *
@@ -1094,8 +1173,9 @@ 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);
+ if (mem_cgroup_init_lru(mem))
+  goto free_out;
+
  spin_lock_init(&mem->lru_lock);
  mem->control_type = MEM_CGROUP_TYPE_ALL;
  memset(&mem->zstat, 0, sizeof(mem->zstat));
@@ -1111,6 +1191,8 @@ mem_cgroup_create(struct cgroup_subsys *
  }
  return &mem->css;
 free_out:
+ mem_cgroup_free_lru(mem);
+
  for_each_node_state(node, N_POSSIBLE)
   kfree(mem->zstat.nodestat[node]);
  if (cont->parent != NULL)
@@ -1131,6 +1213,7 @@ static void mem_cgroup_destroy(struct cg
  int node;
  struct mem_cgroup *mem = mem_cgroup_from_cont(cont);

+ mem_cgroup_free_lru(mem);
  for_each_node_state(node, N_POSSIBLE)
   kfree(mem->zstat.nodestat[node]);
```

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

Subject: Re: [RFC][for -mm] memory controller enhancements for NUMA [10/10] per-zone-lru
Posted by yamamoto on Wed, 14 Nov 2007 09:11:37 GMT

> +struct mc_lru_head {
> + struct list_head active_list[MAX_NR_ZONES];
> + struct list_head inactive_list[MAX_NR_ZONES];
> +};
> +

i guess
 struct foo {
  struct list_head active_list;
  struct list_head inactive_list;
 } lists[MAX_NR_ZONES];
is better.

> @@ -139,8 +144,20 @@ struct mem_cgroup {
>   * Per zone statistics (used for memory reclaim)
>   */
>   struct mem_cgroup_zonestat zstat;
> +#ifndef CONFIG_NUMA
> + struct lru_head local_head;
> +#endif

 struct mc_lru_head local_lru;

> +static int mem_cgroup_init_lru(struct mem_cgroup *mem)
> +{
> + int zone;
> + mem->lrus[0] = &mem->local_lru;

'zone' seems unused.

YAMAMOTO Takashi

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

Subject: Re: Re: [RFC][for -mm] memory controller enhancements for NUMA [10/10] per-zone-lru
Posted by KAMEZAWA Hiroyuki on Wed, 14 Nov 2007 10:29:02 GMT

>> +struct mc_lru_head {
>> + struct list_head active_list[MAX_NR_ZONES];
>> + struct list_head inactive_list[MAX_NR_ZONES];
>> +};

>> +
>
>i guess
> struct foo {
> struct list_head active_list;
> struct list_head inactive_list;
> } lists[MAX_NR_ZONES];
>is better.
Ah, yes. I'll change this.

>> @@ -139,8 +144,20 @@ struct mem_cgroup {
>>   * Per zone statistics (used for memory reclaim)
>>   */
>>   struct mem_cgroup_zonestat zstat;
>> +#ifndef CONFIG_NUMA
>> + struct lru_head local_head;
>> +#endif
>
> struct mc_lru_head local_lru;
>
thanks, I'll do test with !CONFIG_NUMA in the next post.

>> +static int mem_cgroup_init_lru(struct mem_cgroup *mem)
>> +{
>> + int zone;
>> + mem->lrus[0] = &mem->local_lru;
>
>'zone' seems unused.

ok.

Thanks,
-Kame

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

---

Subject: Re: [RFC][for -mm] memory controller enhancements for NUMA [0/10] introduction
Posted by Balbir Singh on Thu, 15 Nov 2007 06:25:50 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:

> Hi,
>
> This is a patch-set for memory controlelr on NUMA.
> patches are
>
> 1. record nid/zid on page_cgroup struct
> 2. record per-zone active/inactive
> 3-9 Patches for isolate global-lru reclaiming and memory controller reclaming
> 10. implements per-zone LRU on memory controller.
>
> now this is just RFC.
>
> Tested on
> 2.6.24-rc2-mm1 + x86_64/fake-NUMA(# of nodes = 3)
>
> I did test with numactl under memory limitation.
> % numactl -i 0,1,2 dd if=.....
>
> It seems per-zone-lru works well.
>
> I'd like to do test on ia64/real-NUMA when I have a chance.
>
> Any comments are welcome.
>

Hi, KAMEZAWA-San,

Thanks for the patchset, I'll review it and get back. I'd
also try and get some testing done on it.

> Thanks,
> -kame
>

--
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][for -mm] memory controller enhancements for NUMA [0/10] introduction

Posted by KAMEZAWA Hiroyuki on Thu, 15 Nov 2007 06:39:36 GMT
View Forum Message <> Reply to Message

On Thu, 15 Nov 2007 11:55:50 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> Hi, KAMEZAWA-San,
>
> Thanks for the patchset, I'll review it and get back. I'd
> also try and get some testing done on it.
>
Hi, thank you.

I'm now writing per-zone-lru-lock patches and reflects comments from YAMAMOTO-san.

Thanks,
-Kame

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

Subject: Re: [RFC][for -mm] memory controller enhancements for NUMA [10/10] per-zone-lru
Posted by KAMEZAWA Hiroyuki on Thu, 15 Nov 2007 11:36:45 GMT
View Forum Message <> Reply to Message

On Wed, 14 Nov 2007 17:57:37 +0900
KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:

> I think there is a consensus that memory controller needs per-zone lru.
> But it was postphoned that it seems some people tries to modify lru of zones.
>
> Now, "scan" value, in mem_cgroup_isolate_pages(), handling is fixed. So,
> demand of implementing per-zone-lru is raised.
>
> This patch implements per-zone lru for memory cgroup.
> I think this patch's style implementation can be adjusted to zone's lru
> implementation changes if happens.
>

I noticed this patch doesn't handle memory migration in sane way.
I will fix in the next version.

-Kame

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