Subject: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [0/10] introduction
Posted by KAMEZAWA Hiroyuki on Tue, 27 Nov 2007 02:54:41 GMT
View Forum Message <> Reply to Message

Hi, this is per-zone/reclaim support patch set for memory controller (cgroup).

Major changes from previous one is
 -- tested with 2.6.24-rc3-mm1 + ia64/NUMA
 -- applied comments.

I did small test on real NUMA machine.
My machine was ia64/8CPU/2Node NUMA. I tried to complile the kernel under 800M bytes limit with 32 parallel make. (make -j 32)

 - 2.6.24-rc3-mm1 (+ scsi fix) .... shows soft lock-up.
   before soft lock-up, %sys was almost 100% in several times.

 - 2.6.24-rc3-mm1 (+ scsi fix) + this set .... completed succesfully
   It seems %iowait dominates the total performance.
   (current memory controller has no background reclaim)

Seems this set give us some progress.

(*) I'd like to merge YAMAMOTO-san's background page reclaim for memory
    controller before discussing about the number of performance.

Andrew, could you pick these up to -mm ?

Patch series brief description:

[1/10] ... add scan_global_lru() macro (clean up)
[2/10] ... nid/zid helper function for cgroup
[3/10] ... introduce per-zone object for memory controller and add
          active/inactive counter.
[4/10] ... calculate mapper_ratio per cgroup (for memory reclaim)
[5/10] ... calculate active/inactive imbalance per cgroup (based on [3])
[6/10] ... remember reclaim priority in memory controller
[7/10] ... calculate the number of pages to be reclaimed per cgroup

[8/10] ... modifies vmscan.c to isolate global-lru-reclaim and
          memory-cgroup-reclaim in obvious manner.
          (this patch uses functions defined in [4 - 7])
[9/10] ... implement per-zone-lru for cgroup (based on [3])
[10/10] ... implement per-zone lru lock for cgroup (based on [3][9])

Any comments are welcome.

Thanks,
-Kame

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

---

Subject: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [1/10] add scan_glob
Posted by KAMEZAWA Hiroyuki on Tue, 27 Nov 2007 02:57:00 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. This may make the meaning obvious.

Acked-by: Balbir Singh <balbir@linux.vnet.ibm.com>
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-rc3-mm1/mm/vmscan.c
===================================================================
--- linux-2.6.24-rc3-mm1.orig/mm/vmscan.c 2007-11-26 15:31:19.000000000 +0900
+++ linux-2.6.24-rc3-mm1/mm/vmscan.c 2007-11-26 16:38:46.000000000 +0900
@@ -127,6 +127,12 @@
 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 @@
```

```
     * 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 @@
     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: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [2/10] nid/zid helpe
Posted by KAMEZAWA Hiroyuki on Tue, 27 Nov 2007 02:58:10 GMT
View Forum Message <> Reply to Message

Add macro to get node_id and zone_id of page_cgroup.
Will be used in per-zone-xxx patches and others.

Changelog:
 - returns zone_type instead of int.

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

```
 mm/memcontrol.c |   10 ++++++++++
 1 file changed, 10 insertions(+)
```

```
Index: linux-2.6.24-rc3-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-26 15:31:19.000000000 +0900
+++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-26 16:39:00.000000000 +0900
@@ -135,6 +135,16 @@
 #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 enum zone_type page_cgroup_zid(struct page_cgroup *pc)
+{
+ return page_zonenum(pc->page);
+}
+
 enum {
  MEM_CGROUP_TYPE_UNSPEC = 0,
  MEM_CGROUP_TYPE_MAPPED,
```

Subject: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [3/10] per-zone acti

Posted by KAMEZAWA Hiroyuki on Tue, 27 Nov 2007 03:00:48 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.
(So, atomic_ops is not necessary.)

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() in initialization.

Changelog V2 -> V3
- fixed comments.

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.

Acked-by: Balbir Singh <balbir@linux.vnet.ibm.com>
Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

```
 mm/memcontrol.c |  164 ++++++++++++++++++++++++++++++++++++++++++++++++++++---
 1 file changed, 157 insertions(+), 7 deletions(-)

Index: linux-2.6.24-rc3-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-26 16:39:00.000000000 +0900
+++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-26 16:39:02.000000000 +0900
@@ -78,6 +78,31 @@
 }

 /*
+ * per-zone information in memory controller.
+ */
+
+enum mem_cgroup_zstat_index {
+ MEM_CGROUP_ZSTAT_ACTIVE,
+ MEM_CGROUP_ZSTAT_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 +126,7 @@
   */
  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 +184,7 @@
  MEM_CGROUP_CHARGE_TYPE_MAPPED,
 };

+
 /*
  * Always modified under lru lock. Then, not necessary to preempt_disable()
  */
@@ -173,7 +200,39 @@
     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 +345,51 @@
  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 +609,7 @@

  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 +673,13 @@
    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 +711,26 @@
 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 +758,11 @@
   /* 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 +941,17 @@
   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 +1005,25 @@
  },
 };

+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 +1039,19 @@
  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 +1064,12 @@
 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 +1122,5 @@
  .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: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [4/10] calculate map

Posted by KAMEZAWA Hiroyuki on Tue, 27 Nov 2007 03:01:15 GMT

View Forum Message <> Reply to Message

Define function for calculating mapped_ratio in memory cgroup.

Changelog V1->V2
 - Fixed possible divide-by-zero bug.
 - Use "long" to avoid 64bit division on 32 bit system.
   and does necessary type casts.
 - Added comments.

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

 include/linux/memcontrol.h | 11 ++++++++++-
 mm/memcontrol.c | 17 +++++++++++++++++
 2 files changed, 27 insertions(+), 1 deletion(-)

Index: linux-2.6.24-rc3-mm1/mm/memcontrol.c
===
--- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-26 16:39:02.000000000 +0900
+++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-26 16:41:34.000000000 +0900
@@ -421,6 +421,23 @@
  spin_unlock(&mem->lru_lock);
 }

+/*
+ * 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) + 1L;
+ rss = (long)mem_cgroup_read_stat(&mem->stat, MEM_CGROUP_STAT_RSS);
+ return (int)((rss * 100L) / 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-rc3-mm1/include/linux/memcontrol.h
===
--- linux-2.6.24-rc3-mm1.orig/include/linux/memcontrol.h 2007-11-26 15:31:19.000000000 +0900
+++ linux-2.6.24-rc3-mm1/include/linux/memcontrol.h 2007-11-26 16:39:05.000000000 +0900
@@ -61,6 +61,12 @@
 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 @@
 {
 }

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

 #endif /* _LINUX_MEMCONTROL_H */
```

Subject: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [5/10] calculate act
Posted by KAMEZAWA Hiroyuki on Tue, 27 Nov 2007 03:02:55 GMT
View Forum Message <> Reply to Message

calculate active/inactive imbalance per memory cgroup.

Changelog V1 -> V2:
 - removed "total" (just count inactive and active)
 - fixed comment
 - fixed return type to be "long".

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

```
 include/linux/memcontrol.h |    8 ++++++++
 mm/memcontrol.c            |   14 ++++++++++++++
 2 files changed, 22 insertions(+)

Index: linux-2.6.24-rc3-mm1/mm/memcontrol.c
===================================================================
```

```
--- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-27 10:44:19.000000000 +0900
+++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-27 11:19:51.000000000 +0900
@@ -437,6 +437,20 @@
  rss = (long)mem_cgroup_read_stat(&mem->stat, MEM_CGROUP_STAT_RSS);
  return (int)((rss * 100L) / total);
 }
+/*
+ * This function is called from vmscan.c. In page reclaiming loop. balance
+ * between active and inactive list is calculated. For memory controller
+ * page reclaiming, we should use using mem_cgroup's imbalance rather than
+ * zone's global lru imbalance.
+ */
+long mem_cgroup_reclaim_imbalance(struct mem_cgroup *mem)
+{
+ unsigned long active, inactive;
+ /* active and inactive are the number of pages. 'long' is ok.*/
+ active = mem_cgroup_get_all_zonestat(mem, MEM_CGROUP_ZSTAT_ACTIVE);
+ inactive = mem_cgroup_get_all_zonestat(mem, MEM_CGROUP_ZSTAT_INACTIVE);
+ return (long) (active / (inactive + 1));
+}

 unsigned long mem_cgroup_isolate_pages(unsigned long nr_to_scan,
      struct list_head *dst,
Index: linux-2.6.24-rc3-mm1/include/linux/memcontrol.h
===================================================================
--- linux-2.6.24-rc3-mm1.orig/include/linux/memcontrol.h 2007-11-27 10:44:19.000000000 +0900
+++ linux-2.6.24-rc3-mm1/include/linux/memcontrol.h 2007-11-27 11:19:00.000000000 +0900
@@ -65,6 +65,8 @@
  * For memory reclaim.
  */
 extern int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem);
+extern long mem_cgroup_reclaim_imbalance(struct mem_cgroup *mem);
+


 #else /* CONFIG_CGROUP_MEM_CONT */
@@ -142,6 +144,12 @@
 {
  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: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [6/10] remember recl

Posted by KAMEZAWA Hiroyuki on Tue, 27 Nov 2007 03:03:52 GMT
View Forum Message <> Reply to Message

Functions to remember reclaim priority per cgroup (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-rc3-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-27 11:19:51.000000000 +0900
+++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-27 11:22:14.000000000 +0900
@@ -132,6 +132,7 @@
   */
  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 @@
  return (long) (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-rc3-mm1/include/linux/memcontrol.h
===================================================================
--- linux-2.6.24-rc3-mm1.orig/include/linux/memcontrol.h 2007-11-27 11:19:00.000000000 +0900
+++ linux-2.6.24-rc3-mm1/include/linux/memcontrol.h 2007-11-27 11:22:14.000000000 +0900
@@ -67,6 +67,11 @@
 extern int mem_cgroup_calc_mapped_ratio(struct mem_cgroup *mem);
 extern long 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 @@
  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: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [7/10] calculate the
Posted by KAMEZAWA Hiroyuki on Tue, 27 Nov 2007 03:06:25 GMT
View Forum Message <> Reply to Message

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

Changelog V1->V2.
 - fixed types of variable.

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-rc3-mm1/include/linux/memcontrol.h
===
--- linux-2.6.24-rc3-mm1.orig/include/linux/memcontrol.h 2007-11-27 11:22:14.000000000 +0900
+++ linux-2.6.24-rc3-mm1/include/linux/memcontrol.h 2007-11-27 11:22:51.000000000 +0900
@@ -73,6 +73,10 @@
 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 @@
  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-rc3-mm1/mm/memcontrol.c
===================================================================
--- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-27 11:22:14.000000000 +0900
+++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-27 11:24:04.000000000 +0900
@@ -472,6 +472,39 @@
  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)
+{
+ long 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 (nr_active >> priority);
+}
+
+long mem_cgroup_calc_reclaim_inactive(struct mem_cgroup *mem,
+      struct zone *zone, int priority)
+{
+ long 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 (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: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [8/10] modifies vmsc
Posted by KAMEZAWA Hiroyuki on Tue, 27 Nov 2007 03:08:18 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.
(scan_global_lru() macro)

This patch tries to make memory cgroup reclaim routine avoid affecting
system/zone memory reclaim. This patch inserts if (scan_global_lru()) 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 V1->V2:
 - 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-rc3-mm1/mm/vmscan.c
===================================================================
--- linux-2.6.24-rc3-mm1.orig/mm/vmscan.c 2007-11-26 16:38:46.000000000 +0900
+++ linux-2.6.24-rc3-mm1/mm/vmscan.c 2007-11-26 16:42:38.000000000 +0900
@@ -863,7 +863,8 @@
   __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 @@
 }

 /*
+ * 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 @@
  * 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 @@
  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 @@
  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 @@
  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 @@
  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 @@
   */
  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: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [9/10] per zone lru
Posted by KAMEZAWA Hiroyuki on Tue, 27 Nov 2007 03:09:19 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);
   &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-rc3-mm1/mm/memcontrol.c
===
--- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-27 11:24:04.000000000 +0900
+++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-27 11:24:16.000000000 +0900
@@ -89,6 +89,8 @@
 };

 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 */
@@ -122,10 +124,7 @@
   /*
    * 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
@@ -367,10 +366,10 @@

```
  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);
 }
@@ -388,11 +387,11 @@
  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 @@
  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 @@
    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 @@
  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 @@
 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 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 @@
   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));
```

Subject: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [10/10] per-zone-loc
Posted by KAMEZAWA Hiroyuki on Tue, 27 Nov 2007 03:10:49 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);
  &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-rc3-mm1/mm/memcontrol.c
===
--- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-27 11:24:16.000000000 +0900
+++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-27 11:24:22.000000000 +0900
@@ -89,6 +89,10 @@
 };

 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];
@@ -126,10 +130,7 @@
   * 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 */
  /*
@@ -410,15 +411,16 @@
  */
 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 @@
   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 @@
  }

  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 @@
  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 @@
   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 @@
 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 @@

  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 @@
    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 @@
  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 @@
  */
 #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 @@
   } 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 @@
     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 @@
   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 @@

  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));

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

---

Subject: Re: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [3/10] per-zone
Posted by Lee Schermerhorn on Wed, 28 Nov 2007 21:19:59 GMT
View Forum Message <> Reply to Message

Just a "heads up": This patch is the apparent cause of a boot time
panic--null pointer deref--on my numa platform. See below.

On Tue, 2007-11-27 at 12:00 +0900, 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.
> (So, atomic_ops is not necessary.)
>
> 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() in initialization.
>
> Changelog V2 -> V3
> - fixed comments.
>
> 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.
>
> Acked-by: Balbir Singh <balbir@linux.vnet.ibm.com>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>
>
> mm/memcontrol.c | 164
++---
> 1 file changed, 157 insertions(+), 7 deletions(-)
>
> Index: linux-2.6.24-rc3-mm1/mm/memcontrol.c
> ===
> --- linux-2.6.24-rc3-mm1.orig/mm/memcontrol.c 2007-11-26 16:39:00.000000000 +0900
> +++ linux-2.6.24-rc3-mm1/mm/memcontrol.c 2007-11-26 16:39:02.000000000 +0900
> @@ -78,6 +78,31 @@
<snip>
>
> +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 +1039,19 @@
>   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;
> +

As soon as this loop hits the first non-existent node on my platform, I
get a NULL pointer deref down in __alloc_pages. Stack trace below.

Perhaps N_POSSIBLE should be N_HIGH_MEMORY? That would require handling
of memory/node hotplug for each memory control group, right? But, I'm
going to try N_HIGH_MEMORY as a work around.

Lee
>   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 +1064,12 @@
> 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 +1122,5 @@
>   .destroy = mem_cgroup_destroy,
>   .populate = mem_cgroup_populate,
>   .attach = mem_cgroup_move_task,
> - .early_init = 1,
> + .early_init = 0,
> };

Initializing cgroup subsys memory
Unable to handle kernel NULL pointer dereference (address 0000000000003c80)
swapper[0]: Oops 11012296146944 [1]

Modules linked in:

Pid: 0, CPU 0, comm: swapper
psr : 00001210084a6010 ifs : 8000000000000b1a ip : [<a000000100132e11>] Not tainted
ip is at __alloc_pages+0x31/0x6e0
unat: 0000000000000000 pfs : 000000000000060f rsc : 0000000000000003
rnat: a0000001009db3b8 bsps: a0000001009e0490 pr : 656960155aa65659
ldrs: 0000000000000000 ccv : 0000000000000000 fpsr: 0009804c8a70433f
csd : 0000000000000000 ssd : 0000000000000000
b0 : a000000100187370 b6 : a000000100194440 b7 : a00000010086d560
f6 : 1003e0000000000000000 f7 : 1003e0000000000000055
f8 : 1003e00000000000000c0 f9 : 1003e0000000000003fc0
f10 : 1003e00000000000000c0 f11 : 1003e0000000000000055
r1 : a000000100bc0f10 r2 : fffffffffe00006 r3 : 0000000000020000
r8 : 0000000000071ef0 r9 : 0000000000000005 r10 : e00007002034d588
r11 : e00007002034d580 r12 : a0000001008e3df0 r13 : a0000001008dc000
r14 : 0000000000000001 r15 : e00007002034d5b0 r16 : 0000000000001e78
r17 : ffffffffffff04e0 r18 : 0000000000100002 r19 : 0000000000000000
r20 : 0000000000100002 r21 : 00000000000003cf r22 : 000000000000000f
r23 : 00000000000003c0 r24 : 0000000000000010 r25 : 0000000000000001
r26 : a0000001008e3e20 r27 : 0000000000000000 r28 : e0000701813dc088
r29 : e0000701813dc080 r30 : 0000000000000000 r31 : a000000100918ea8

Call Trace:
[<a000000100014de0>] show_stack+0x80/0xa0
sp=a0000001008e39c0 bsp=a0000001008dd1b0
[<a000000100015a70>] show_regs+0x870/0x8a0
sp=a0000001008e3b90 bsp=a0000001008dd158
[<a00000010003d130>] die+0x190/0x300
sp=a0000001008e3b90 bsp=a0000001008dd110
[<a000000100071b80>] ia64_do_page_fault+0x8e0/0xa20
sp=a0000001008e3b90 bsp=a0000001008dd0b8
[<a00000010000b5c0>] ia64_leave_kernel+0x0/0x270
sp=a0000001008e3c20 bsp=a0000001008dd0b8
[<a000000100132e10>] __alloc_pages+0x30/0x6e0
sp=a0000001008e3df0 bsp=a0000001008dcfe0
[<a000000100187370>] new_slab+0x610/0x6c0
sp=a0000001008e3e00 bsp=a0000001008dcf80
[<a000000100187470>] get_new_slab+0x50/0x200
sp=a0000001008e3e00 bsp=a0000001008dcf48
[<a000000100187900>] __slab_alloc+0x2e0/0x4e0
sp=a0000001008e3e00 bsp=a0000001008dcf00
[<a000000100187c80>] kmem_cache_alloc_node+0x180/0x200
sp=a0000001008e3e10 bsp=a0000001008dcec0
[<a0000001001945a0>] mem_cgroup_create+0x160/0x400
sp=a0000001008e3e10 bsp=a0000001008dce78
[<a0000001000f0940>] cgroup_init_subsys+0xa0/0x400
sp=a0000001008e3e20 bsp=a0000001008dce28

[<a0000001008521f0>] cgroup_init+0x90/0x160
sp=a0000001008e3e20 bsp=a0000001008dce00
[<a000000100831960>] start_kernel+0x700/0x820
sp=a0000001008e3e20 bsp=a0000001008dcd80

_____

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

---

Subject: Re: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [3/10] per-zone
Posted by KAMEZAWA Hiroyuki on Thu, 29 Nov 2007 01:37:02 GMT
View Forum Message <> Reply to Message

On Wed, 28 Nov 2007 16:19:59 -0500
Lee Schermerhorn <Lee.Schermerhorn@hp.com> wrote:

> As soon as this loop hits the first non-existent node on my platform, I
> get a NULL pointer deref down in __alloc_pages. Stack trace below.
>
> Perhaps N_POSSIBLE should be N_HIGH_MEMORY? That would require handling
> of memory/node hotplug for each memory control group, right? But, I'm
> going to try N_HIGH_MEMORY as a work around.
>
Hmm, ok. (>_<

> Call Trace:
> [<a000000100014de0>] show_stack+0x80/0xa0
> sp=a0000001008e39c0 bsp=a0000001008dd1b0
> [<a000000100015a70>] show_regs+0x870/0x8a0
> sp=a0000001008e3b90 bsp=a0000001008dd158
> [<a00000010003d130>] die+0x190/0x300
> sp=a0000001008e3b90 bsp=a0000001008dd110
> [<a000000100071b80>] ia64_do_page_fault+0x8e0/0xa20
> sp=a0000001008e3b90 bsp=a0000001008dd0b8
> [<a00000010000b5c0>] ia64_leave_kernel+0x0/0x270
> sp=a0000001008e3c20 bsp=a0000001008dd0b8
> [<a000000100132e10>] __alloc_pages+0x30/0x6e0
> sp=a0000001008e3df0 bsp=a0000001008dcfe0
> [<a000000100187370>] new_slab+0x610/0x6c0
> sp=a0000001008e3e00 bsp=a0000001008dcf80
> [<a000000100187470>] get_new_slab+0x50/0x200
> sp=a0000001008e3e00 bsp=a0000001008dcf48

> [<a000000100187900>] __slab_alloc+0x2e0/0x4e0
> sp=a0000001008e3e00 bsp=a0000001008dcf00
> [<a000000100187c80>] kmem_cache_alloc_node+0x180/0x200
> sp=a0000001008e3e10 bsp=a0000001008dcec0
> [<a0000001001945a0>] mem_cgroup_create+0x160/0x400
> sp=a0000001008e3e10 bsp=a0000001008dce78
> [<a0000001000f0940>] cgroup_init_subsys+0xa0/0x400
> sp=a0000001008e3e20 bsp=a0000001008dce28
> [<a0000001008521f0>] cgroup_init+0x90/0x160
> sp=a0000001008e3e20 bsp=a0000001008dce00
> [<a000000100831960>] start_kernel+0x700/0x820
> sp=a0000001008e3e20 bsp=a0000001008dcd80
>
Maybe zonelists of NODE_DATA() is not initialized. you are right.
I think N_HIGH_MEMORY will be suitable here...(I'll consider node-hotplug case later.)

Thank you for test!

Regards,
-Kame

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

---

Subject: Re: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [3/10] per-zone
Posted by KAMEZAWA Hiroyuki on Thu, 29 Nov 2007 02:24:06 GMT
View Forum Message <> Reply to Message

On Thu, 29 Nov 2007 10:37:02 +0900
KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:

> Maybe zonelists of NODE_DATA() is not initialized. you are right.
> I think N_HIGH_MEMORY will be suitable here...(I'll consider node-hotplug case later.)
>
> Thank you for test!
>
Could you try this ?

Thanks,
-Kame
==

Don't call kmalloc() against possible but offline node.

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

```
 mm/memcontrol.c |   10 ++++++++--
 1 file changed, 8 insertions(+), 2 deletions(-)

Index: test-2.6.24-rc3-mm1/mm/memcontrol.c
===================================================================
--- test-2.6.24-rc3-mm1.orig/mm/memcontrol.c
+++ test-2.6.24-rc3-mm1/mm/memcontrol.c
@@ -1117,8 +1117,14 @@ static int alloc_mem_cgroup_per_zone_inf
  struct mem_cgroup_per_node *pn;
  struct mem_cgroup_per_zone *mz;
  int zone;
-
- pn = kmalloc_node(sizeof(*pn), GFP_KERNEL, node);
+ /*
+  * This routine is called against possible nodes.
+  * But it's BUG to call kmalloc() against offline node.
+  */
+ if (node_state(N_ONLINE, node))
+  pn = kmalloc_node(sizeof(*pn), GFP_KERNEL, node);
+ else
+  pn = kmalloc(sizeof(*pn), GFP_KERNEL);
  if (!pn)
   return 1;
```

Subject: Re: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [3/10] per-zone

Posted by KAMEZAWA Hiroyuki on Thu, 29 Nov 2007 03:18:34 GMT

View Forum Message <> Reply to Message

On Thu, 29 Nov 2007 11:24:06 +0900
KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:

> On Thu, 29 Nov 2007 10:37:02 +0900
> KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
>
> > Maybe zonelists of NODE_DATA() is not initialized. you are right.
> > I think N_HIGH_MEMORY will be suitable here...(I'll consider node-hotplug case later.)

> >
> > Thank you for test!
> >
> Could you try this ?
>
Sorry..this can be a workaround but I noticed I miss something..

ok, just use N_HIGH_MEMORY here and add comment for hotplugging support is not yet.

Christoph-san, Lee-san, could you confirm following ?

- when SLAB is used, kmalloc_node() against offline node will success.
- when SLUB is used, kmalloc_node() against offline node will panic.

Then, the caller should take care that node is online before kmalloc().

Regards,
-Kame

Subject: Re: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [3/10] per-zone
Posted by yamamoto on Thu, 29 Nov 2007 03:19:37 GMT
View Forum Message <> Reply to Message

> @@ -651,10 +758,11 @@
> /* 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;

'active' seems unused.

YAMAMOTO Takashi

Subject: Re: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [3/10] per-zone
Posted by KAMEZAWA Hiroyuki on Thu, 29 Nov 2007 03:25:32 GMT
View Forum Message <> Reply to Message

On Thu, 29 Nov 2007 12:19:37 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> > @@ -651,10 +758,11 @@
> > /* 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;
>
> 'active' seems unused.
>
ok, I will post clean-up against -mm2.

Thanks,
-Kame

Subject: Re: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [3/10] per-zone
Posted by Christoph Lameter on Thu, 29 Nov 2007 03:26:29 GMT
View Forum Message <> Reply to Message

On Thu, 29 Nov 2007, KAMEZAWA Hiroyuki wrote:

> ok, just use N_HIGH_MEMORY here and add comment for hotplugging support is not yet.
>
> Christoph-san, Lee-san, could you confirm following ?
>
> - when SLAB is used, kmalloc_node() against offline node will success.
> - when SLUB is used, kmalloc_node() against offline node will panic.
>
> Then, the caller should take care that node is online before kmalloc().

Hmmmm... An offline node implies that the per node structure does not
exist. SLAB should fail too. If there is something wrong with the allocs
then its likely a difference in the way hotplug was put into SLAB and
SLUB.

Subject: Re: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [3/10] per-zone
Posted by yamamoto on Thu, 29 Nov 2007 03:33:28 GMT
View Forum Message <> Reply to Message

> +static inline struct mem_cgroup_per_zone *
> +mem_cgroup_zoneinfo(struct mem_cgroup *mem, int nid, int zid)
> +{
> + if (!mem->info.nodeinfo[nid])

can this be true?

YAMAMOTO Takashi

> + return NULL;
> + return &mem->info.nodeinfo[nid]->zoneinfo[zid];
> +}
> +

Subject: Re: [PATCH][for -mm] per-zone and reclaim enhancements for memory controller take 3 [3/10] per-zone
Posted by KAMEZAWA Hiroyuki on Thu, 29 Nov 2007 03:42:07 GMT
View Forum Message <> Reply to Message

On Thu, 29 Nov 2007 12:33:28 +0900 (JST)
yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:

> > +static inline struct mem_cgroup_per_zone *
> > +mem_cgroup_zoneinfo(struct mem_cgroup *mem, int nid, int zid)
> > +{
> > + if (!mem->info.nodeinfo[nid])
>
> can this be true?
>
> YAMAMOTO Takashi

When I set early_init=1, I added that check.
BUG_ON() is better ?

Thanks,
-Kame