Subject: [RFC][-mm] Memory controller hierarchy support (v1)
Posted by Balbir Singh on Sat, 19 Apr 2008 05:35:51 GMT
View Forum Message <> Reply to Message

This applies on top of 2.6.25-rc8-mm2. The next version will be applied
on top of 2.5.25-mm1.

This code is built on top of Pavel's hierarchy patches.

1. It propagates the charges upwards. A charge incurred on a cgroup
   is propagated to root. If any of the counters along the hierarchy
   is over limit, reclaim is initiated from the parent. We reclaim
   pages from the parent and the children below it. We also keep track
   of the last child from whom reclaim was done and start from there in
   the next reclaim.

TODO's/Open Questions

1. We need to hold cgroup_mutex while walking through the children
   in reclaim. We need to figure out the best way to do so. Should
   cgroups provide a helper function/macro for it?
2. Do not allow children to have a limit greater than their parents.
3. Allow the user to select if hierarchial support is required
4. Fine tune reclaim from children logic

Testing

This code was tested on a UML instance, where it compiled and worked well.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
---

 include/linux/res_counter.h | 14 ++++
 kernel/res_counter.c | 42 ++++++++++---
 mm/memcontrol.c | 128 ++++++++++++++++++++++++++++++++++++++-----
 3 files changed, 148 insertions(+), 36 deletions(-)

diff -puN include/linux/res_counter.h~memory-controller-hierarchy-support
include/linux/res_counter.h
--- linux-2.6.25-rc8/include/linux/res_counter.h~memory-controller-hierarchy-support 2008-04-19
11:00:28.000000000 +0530
+++ linux-2.6.25-rc8-balbir/include/linux/res_counter.h 2008-04-19 11:00:28.000000000 +0530
@@ -43,6 +43,10 @@ struct res_counter {
   * the routines below consider this to be IRQ-safe
   */
  spinlock_t lock;
+ /*

```
+  * the parent counter. used for hierarchical resource accounting
+  */
+ struct res_counter *parent;
 };

 /**
@@ -82,7 +86,12 @@ enum {
  * helpers for accounting
  */

-void res_counter_init(struct res_counter *counter);
+/*
+ * the parent pointer is set only once - during the counter
+ * initialization. caller then must itself provide that this
+ * pointer is valid during the new counter lifetime
+ */
+void res_counter_init(struct res_counter *counter, struct res_counter *parent);

 /*
  * charge - try to consume more resource.
@@ -96,7 +105,8 @@ void res_counter_init(struct res_counter
  */

 int res_counter_charge_locked(struct res_counter *counter, unsigned long val);
-int res_counter_charge(struct res_counter *counter, unsigned long val);
+int res_counter_charge(struct res_counter *counter, unsigned long val,
+    struct res_counter **limit_exceeded_at);

 /*
  * uncharge - tell that some portion of the resource is released
diff -puN kernel/res_counter.c~memory-controller-hierarchy-support kernel/res_counter.c
--- linux-2.6.25-rc8/kernel/res_counter.c~memory-controller-hierarchy-support 2008-04-19
11:00:28.000000000 +0530
+++ linux-2.6.25-rc8-balbir/kernel/res_counter.c 2008-04-19 11:00:28.000000000 +0530
@@ -14,10 +14,11 @@
 #include <linux/res_counter.h>
 #include <linux/uaccess.h>

-void res_counter_init(struct res_counter *counter)
+void res_counter_init(struct res_counter *counter, struct res_counter *parent)
 {
  spin_lock_init(&counter->lock);
  counter->limit = (unsigned long long)LLONG_MAX;
+ counter->parent = parent;
 }

 int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
@@ -33,14 +34,34 @@ int res_counter_charge_locked(struct res
```

```
  return 0;
 }

-int res_counter_charge(struct res_counter *counter, unsigned long val)
+int res_counter_charge(struct res_counter *counter, unsigned long val,
+    struct res_counter **limit_exceeded_at)
 {
  int ret;
  unsigned long flags;
+ struct res_counter *c, *unroll_c;

- spin_lock_irqsave(&counter->lock, flags);
- ret = res_counter_charge_locked(counter, val);
- spin_unlock_irqrestore(&counter->lock, flags);
+ *limit_exceeded_at = NULL;
+ local_irq_save(flags);
+ for (c = counter; c != NULL; c = c->parent) {
+  spin_lock(&c->lock);
+  ret = res_counter_charge_locked(c, val);
+  spin_unlock(&c->lock);
+  if (ret < 0) {
+   *limit_exceeded_at = c;
+   goto unroll;
+  }
+ }
+ local_irq_restore(flags);
+ return 0;
+
+unroll:
+ for (unroll_c = counter; unroll_c != c; unroll_c = unroll_c->parent) {
+  spin_lock(&unroll_c->lock);
+  res_counter_uncharge_locked(unroll_c, val);
+  spin_unlock(&unroll_c->lock);
+ }
+ local_irq_restore(flags);
  return ret;
 }

@@ -55,10 +76,15 @@ void res_counter_uncharge_locked(struct
 void res_counter_uncharge(struct res_counter *counter, unsigned long val)
 {
  unsigned long flags;
+ struct res_counter *c;

- spin_lock_irqsave(&counter->lock, flags);
- res_counter_uncharge_locked(counter, val);
- spin_unlock_irqrestore(&counter->lock, flags);
+ local_irq_save(flags);
```

```
+ for (c = counter; c != NULL; c = c->parent) {
+  spin_lock(&c->lock);
+  res_counter_uncharge_locked(c, val);
+  spin_unlock(&c->lock);
+ }
+ local_irq_restore(flags);
 }


diff -puN mm/memcontrol.c~memory-controller-hierarchy-support mm/memcontrol.c
--- linux-2.6.25-rc8/mm/memcontrol.c~memory-controller-hierarchy-support 2008-04-19
11:00:28.000000000 +0530
+++ linux-2.6.25-rc8-balbir/mm/memcontrol.c 2008-04-19 11:00:28.000000000 +0530
@@ -138,6 +138,13 @@ struct mem_cgroup {
   * statistics.
   */
  struct mem_cgroup_stat stat;
+
+ /*
+  * When reclaiming in a hierarchy, we need to know, which child
+  * we reclaimed last from. This helps us avoid hitting the first
+  * child over and over again
+  */
+ struct mem_cgroup *last_scanned_child;
 };
 static struct mem_cgroup init_mem_cgroup;

@@ -244,6 +251,12 @@ struct mem_cgroup *mem_cgroup_from_task(
     struct mem_cgroup, css);
 }

+static struct mem_cgroup*
+mem_cgroup_from_res_counter(struct res_counter *counter)
+{
+ return container_of(counter, struct mem_cgroup, res);
+}
+
 static inline int page_cgroup_locked(struct page *page)
 {
  return bit_spin_is_locked(PAGE_CGROUP_LOCK_BIT, &page->page_cgroup);
@@ -508,6 +521,86 @@ unsigned long mem_cgroup_isolate_pages(u
 }

 /*
+ * Charge mem and check if it is over it's limit. If so, reclaim from
+ * mem. This function can call itself recursively (as we walk up the
+ * hierarchy).
+ */
```

```
+static int mem_cgroup_charge_and_reclaim(struct mem_cgroup *mem, gfp_t gfp_mask)
+{
+ int ret = 0;
+ unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
+ struct res_counter *counter_over_limit;
+ struct mem_cgroup *mem_over_limit;
+ struct cgroup *cgroup, *cgrp, *curr_cgroup;
+
+ while (res_counter_charge(&mem->res, PAGE_SIZE, &counter_over_limit)) {
+  if (!(gfp_mask & __GFP_WAIT))
+   goto out;
+
+  /*
+   * Is one of our ancestors over limit ?
+   */
+  if (counter_over_limit) {
+   mem_over_limit =
+    mem_cgroup_from_res_counter(counter_over_limit);
+
+   if (mem != mem_over_limit)
+    ret = mem_cgroup_charge_and_reclaim(
+      mem_over_limit, gfp_mask);
+  }
+
+  if (try_to_free_mem_cgroup_pages(mem, gfp_mask))
+   continue;
+
+  /*
+   * try_to_free_mem_cgroup_pages() might not give us a full
+   * picture of reclaim. Some pages are reclaimed and might be
+   * moved to swap cache or just unmapped from the cgroup.
+   * Check the limit again to see if the reclaim reduced the
+   * current usage of the cgroup before giving up
+   */
+  if (res_counter_check_under_limit(&mem->res))
+   continue;
+
+  /*
+   * Now scan all children under the group. This is required
+   * to support hierarchies
+   */
+  if (!mem->last_scanned_child)
+   cgroup = list_first_entry(&mem->css.cgroup->children,
+      struct cgroup, sibling);
+  else
+   cgroup = mem->last_scanned_child->css.cgroup;
+
+  curr_cgroup = mem->css.cgroup;
```

```
+
+  /*
+   * Ideally we need to hold cgroup_mutex here
+   */
+  list_for_each_entry_safe_from(cgroup, cgrp,
+    &curr_cgroup->children, sibling) {
+   struct mem_cgroup *mem_child;
+
+   mem_child = mem_cgroup_from_cont(cgroup);
+   ret = try_to_free_mem_cgroup_pages(mem_child,
+         gfp_mask);
+   mem->last_scanned_child = mem_child;
+   if (ret == 0)
+    break;
+  }
+
+  if (!nr_retries--) {
+   mem_cgroup_out_of_memory(mem, gfp_mask);
+   ret = -ENOMEM;
+   break;
+  }
+ }
+
+out:
+ return ret;
+}
+
+/*
  * Charge the memory controller for page usage.
  * Return
  * 0 if the charge was successful
@@ -519,7 +612,6 @@ static int mem_cgroup_charge_common(stru
  struct mem_cgroup *mem;
  struct page_cgroup *pc;
  unsigned long flags;
- unsigned long nr_retries = MEM_CGROUP_RECLAIM_RETRIES;
  struct mem_cgroup_per_zone *mz;

  if (mem_cgroup_subsys.disabled)
@@ -570,28 +662,8 @@ retry:
  css_get(&mem->css);
  rcu_read_unlock();

- while (res_counter_charge(&mem->res, PAGE_SIZE)) {
-  if (!(gfp_mask & __GFP_WAIT))
-   goto out;
-
-  if (try_to_free_mem_cgroup_pages(mem, gfp_mask))
```

```
-   continue;
-
-  /*
-   * try_to_free_mem_cgroup_pages() might not give us a full
-   * picture of reclaim. Some pages are reclaimed and might be
-   * moved to swap cache or just unmapped from the cgroup.
-   * Check the limit again to see if the reclaim reduced the
-   * current usage of the cgroup before giving up
-   */
-  if (res_counter_check_under_limit(&mem->res))
-   continue;
-
-  if (!nr_retries--) {
-   mem_cgroup_out_of_memory(mem, gfp_mask);
-   goto out;
-  }
- }
+ if (mem_cgroup_charge_and_reclaim(mem, gfp_mask))
+  goto out;

  pc->ref_cnt = 1;
  pc->mem_cgroup = mem;
@@ -986,19 +1058,23 @@ static void free_mem_cgroup_per_zone_inf
 static struct cgroup_subsys_state *
 mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
 {
- struct mem_cgroup *mem;
+ struct mem_cgroup *mem, *parent;
  int node;

  if (unlikely((cont->parent) == NULL)) {
   mem = &init_mem_cgroup;
   page_cgroup_cache = KMEM_CACHE(page_cgroup, SLAB_PANIC);
- } else
+  parent = NULL;
+ } else {
   mem = kzalloc(sizeof(struct mem_cgroup), GFP_KERNEL);
+  parent = mem_cgroup_from_cont(cont->parent);
+ }

  if (mem == NULL)
   return ERR_PTR(-ENOMEM);

- res_counter_init(&mem->res);
+ res_counter_init(&mem->res, parent ? &parent->res : NULL);
+ mem->last_scanned_child = NULL;

  memset(&mem->info, 0, sizeof(mem->info));
```

_

--
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