Subject: [PATCH 2/2] Make res_counter hierarchical
Posted by Pavel Emelianov on Fri, 07 Mar 2008 15:32:20 GMT
View Forum Message <> Reply to Message

This allows us two things basically:

1. If the subgroup has the limit higher than its parent has
   then the one will get more memory than allowed.
2. When we will need to account for a resource in more than
   one place, we'll be able to use this technics.

   Look, consider we have a memory limit and swap limit. The
   memory limit is the limit for the sum of RSS, page cache
   and swap usage. To account for this gracefuly, we'll set
   two counters:

    res_counter mem_counter;
    res_counter swap_counter;

   attach mm to the swap one

    mm->mem_cnt = &swap_counter;

   and make the swap_counter be mem's child. That's it. If we
   want hierarchical support, then the tree will look like this:

   mem_counter_top
    swap_counter_top <- mm_struct living at top
     mem_counter_sub
      swap_counter_sub <- mm_struct living at sub

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---
 include/linux/res_counter.h | 11 ++++++++++-
 kernel/res_counter.c | 36 ++++++++++++++++++++++++++++++++-------
 mm/memcontrol.c | 9 ++++++---
 3 files changed, 45 insertions(+), 11 deletions(-)

diff --git a/include/linux/res_counter.h b/include/linux/res_counter.h
index 2c4deb5..a27105e 100644
--- a/include/linux/res_counter.h
+++ b/include/linux/res_counter.h
@@ -41,6 +41,10 @@ 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;
 };

 /**
@@ -80,7 +84,12 @@ 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.
diff --git a/kernel/res_counter.c b/kernel/res_counter.c
index f1f20c2..046f6f4 100644
--- a/kernel/res_counter.c
+++ b/kernel/res_counter.c
@@ -13,10 +13,11 @@
 #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)
@@ -36,10 +37,26 @@ int res_counter_charge(struct res_counter *counter, unsigned long val)
 {
  int ret;
  unsigned long flags;
+ struct res_counter *c, *unroll_c;
+
+ 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)
+   goto unroll;
+ }
+ local_irq_restore(flags);
+ return 0;

- spin_lock_irqsave(&counter->lock, flags);
- ret = res_counter_charge_locked(counter, val);
- spin_unlock_irqrestore(&counter->lock, flags);
+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;
 }

@@ -54,10 +71,15 @@ void res_counter_uncharge_locked(struct res_counter *counter,
unsigned long val)
 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 --git a/mm/memcontrol.c b/mm/memcontrol.c
index e5c741a..61db79c 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -976,19 +976,22 @@ static void free_mem_cgroup_per_zone_info(struct mem_cgroup
*mem, int node)
 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;
   init_mm.mem_cgroup = mem;
- } 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);

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

--
1.5.3.4
```

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by KAMEZAWA Hiroyuki on Sat, 08 Mar 2008 04:44:23 GMT
View Forum Message <> Reply to Message

On Fri, 07 Mar 2008 18:32:20 +0300
Pavel Emelyanov <xemul@openvz.org> wrote:

> This allows us two things basically:
>
> 1. If the subgroup has the limit higher than its parent has
>   then the one will get more memory than allowed.
> 2. When we will need to account for a resource in more than
>   one place, we'll be able to use this technics.
>
>   Look, consider we have a memory limit and swap limit. The
>   memory limit is the limit for the sum of RSS, page cache
>   and swap usage. To account for this gracefuly, we'll set

>   two counters:
>
>   res_counter mem_counter;
>   res_counter swap_counter;
>
>   attach mm to the swap one
>
>   mm->mem_cnt = &swap_counter;
>
>   and make the swap_counter be mem's child. That's it. If we
>   want hierarchical support, then the tree will look like this:
>
>   mem_counter_top
>   swap_counter_top <- mm_struct living at top
>   mem_counter_sub
>   swap_counter_sub <- mm_struct living at sub
>
Hmm? seems strange.

IMO, a parent's usage is just sum of all childs'.
And, historically, memory overcommit is done agaist "memory usage + swap".

How about this ?
    <mem_counter_top, swap_counter_top>
 <mem_counter_sub, swap_counter_sub>
 <mem_counter_sub, swap_counter_sub>
 <mem_counter_sub, swap_counter_sub>

   mem_counter_top.usage == sum of all mem_coutner_sub.usage
   swap_counter_sub.usage = sum of all swap_counter_sub.usage

> @@ -976,19 +976,22 @@ static void free_mem_cgroup_per_zone_info(struct mem_cgroup
*mem, int node)
> 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;
>   init_mm.mem_cgroup = mem;
> - } 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);
>
I have no objection to add some hierarchical support to res_counter.

But we should wait to add it to mem_cgroup because we have to add
some amount of codes to handle hierarchy under mem_cgroup in reasonable way.
for example)
 - hierarchical memory reclaim
 - keeping fairness between sub memory controllers.
   etc...

Thanks,
-Kame

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by Balbir Singh on Sat, 08 Mar 2008 19:06:42 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> This allows us two things basically:
>
> 1. If the subgroup has the limit higher than its parent has
>   then the one will get more memory than allowed.

But should we allow such configuration? I suspect that we should catch such
things at the time of writing the limit.

> 2. When we will need to account for a resource in more than
>   one place, we'll be able to use this technics.
>
>   Look, consider we have a memory limit and swap limit. The
>   memory limit is the limit for the sum of RSS, page cache
>   and swap usage. To account for this gracefuly, we'll set
>   two counters:
>

>   res_counter mem_counter;
>   res_counter swap_counter;
>
>   attach mm to the swap one
>
>   mm->mem_cnt = &swap_counter;
>
>   and make the swap_counter be mem's child. That's it. If we
>   want hierarchical support, then the tree will look like this:
>
>   mem_counter_top
>   swap_counter_top <- mm_struct living at top
>   mem_counter_sub
>   swap_counter_sub <- mm_struct living at sub
>

Hmm... not sure about this one. What I want to see is a resource counter hierarchy to mimic the container hierarchy. Then ensure that all limits are set sanely. I am planning to implement shares support on to of resource counters.

> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
> include/linux/res_counter.h | 11 ++++++++++-
> kernel/res_counter.c | 36 ++++++++++++++++++++++++++++++++-------
> mm/memcontrol.c | 9 ++++++---
> 3 files changed, 45 insertions(+), 11 deletions(-)
>
> diff --git a/include/linux/res_counter.h b/include/linux/res_counter.h
> index 2c4deb5..a27105e 100644
> --- a/include/linux/res_counter.h
> +++ b/include/linux/res_counter.h
> @@ -41,6 +41,10 @@ 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;
>   };
>
>   /**
> @@ -80,7 +84,12 @@ 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.
> diff --git a/kernel/res_counter.c b/kernel/res_counter.c
> index f1f20c2..046f6f4 100644
> --- a/kernel/res_counter.c
> +++ b/kernel/res_counter.c
> @@ -13,10 +13,11 @@
> #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)
> @@ -36,10 +37,26 @@ int res_counter_charge(struct res_counter *counter, unsigned long val)
> {
>   int ret;
>   unsigned long flags;
> + struct res_counter *c, *unroll_c;
> +
> + 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)
> +   goto unroll;

We'd like to know which resource counter failed to allow charging, so that we
can reclaim from that mem_res_cgroup.

> + }
> + local_irq_restore(flags);
> + return 0;
>

> - spin_lock_irqsave(&counter->lock, flags);
> - ret = res_counter_charge_locked(counter, val);
> - spin_unlock_irqrestore(&counter->lock, flags);
> +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;
> }
>
> @@ -54,10 +71,15 @@ void res_counter_uncharge_locked(struct res_counter *counter,
unsigned long val)
> 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 --git a/mm/memcontrol.c b/mm/memcontrol.c
> index e5c741a..61db79c 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -976,19 +976,22 @@ static void free_mem_cgroup_per_zone_info(struct mem_cgroup
*mem, int node)
> 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;

> init_mm.mem_cgroup = mem;
> - } 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);
>
> 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

---

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by Pavel Emelianov on Tue, 11 Mar 2008 08:15:56 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> On Fri, 07 Mar 2008 18:32:20 +0300
> Pavel Emelyanov <xemul@openvz.org> wrote:
>
>> This allows us two things basically:
>>
>> 1. If the subgroup has the limit higher than its parent has
>> then the one will get more memory than allowed.
>> 2. When we will need to account for a resource in more than
>> one place, we'll be able to use this technics.
>>
>> Look, consider we have a memory limit and swap limit. The
>> memory limit is the limit for the sum of RSS, page cache
>> and swap usage. To account for this gracefuly, we'll set

>>   two counters:
>>
>>   res_counter mem_counter;
>>   res_counter swap_counter;
>>
>>   attach mm to the swap one
>>
>>   mm->mem_cnt = &swap_counter;
>>
>>   and make the swap_counter be mem's child. That's it. If we
>>   want hierarchical support, then the tree will look like this:
>>
>>   mem_counter_top
>>   swap_counter_top <- mm_struct living at top
>>   mem_counter_sub
>>   swap_counter_sub <- mm_struct living at sub
>>
> Hmm? seems strange.
>
> IMO, a parent's usage is just sum of all childs'.
> And, historically, memory overcommit is done agaist "memory usage + swap".
>
> How about this ?
>   <mem_counter_top, swap_counter_top>
> <mem_counter_sub, swap_counter_sub>
> <mem_counter_sub, swap_counter_sub>
> <mem_counter_sub, swap_counter_sub>
>
>   mem_counter_top.usage == sum of all mem_coutner_sub.usage
>   swap_counter_sub.usage = sum of all swap_counter_sub.usage

I've misprinted in y tree, sorry.
The correct hierarchy as I see it is

<mem_couter_0>
 + -- <swap_counter_0>
 + -- <mem_counter_1>
 | + -- <swap_counter_1>
 | + -- <mem_counter_11>
 | | + -- <swap_counter_11>
 | + -- <mem_counter_12>
 | + -- <swap_counter_12>
 + -- <mem_counter_2>
 | + -- <swap_counter_2>
 | + -- <mem_counter_21>
 | | + -- <swap_counter_21>
 | + -- <mem_counter_22>
 | + -- <swap_counter_22>

```
 + -- <mem_counter_N>
      + -- <swap_counter_N>
      + -- <mem_counter_N1>
      |     + -- <swap_counter_N1>
      + -- <mem_counter_N2>
           + -- <swap_counter_N2>
```

>
>> @@ -976,19 +976,22 @@ static void free_mem_cgroup_per_zone_info(struct mem_cgroup
*mem, int node)
>> 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;
>>   init_mm.mem_cgroup = mem;
>> - } 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);
>>
> I have no objection to add some hierarchical support to res_counter.
>
> But we should wait to add it to mem_cgroup because we have to add
> some amount of codes to handle hierarchy under mem_cgroup in reasonable way.
> for example)
> - hierarchical memory reclaim
> - keeping fairness between sub memory controllers.
>   etc...
>
> Thanks,
> -Kame
>
>

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by Pavel Emelianov on Tue, 11 Mar 2008 08:17:59 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelyanov wrote:
>> This allows us two things basically:
>>
>> 1. If the subgroup has the limit higher than its parent has
>>   then the one will get more memory than allowed.
>
> But should we allow such configuration? I suspect that we should catch such
> things at the time of writing the limit.

We cannot catch this at the limit-set-time. See, if you have a cgroup A
with a 1GB limit and the usage is 999Mb, then creating a subgroup B with
even 500MB limit will cause the A group consume 1.5GB of memory
effectively.

>> 2. When we will need to account for a resource in more than
>>   one place, we'll be able to use this technics.
>>
>>   Look, consider we have a memory limit and swap limit. The
>>   memory limit is the limit for the sum of RSS, page cache
>>   and swap usage. To account for this gracefuly, we'll set
>>   two counters:
>>
>>   res_counter mem_counter;
>>   res_counter swap_counter;
>>
>>   attach mm to the swap one
>>
>>   mm->mem_cnt = &swap_counter;
>>
>>   and make the swap_counter be mem's child. That's it. If we
>>   want hierarchical support, then the tree will look like this:
>>
>>   mem_counter_top
>>   swap_counter_top <- mm_struct living at top
>>   mem_counter_sub
>>   swap_counter_sub <- mm_struct living at sub
>>
>

> Hmm... not sure about this one. What I want to see is a resource counter
> hierarchy to mimic the container hierarchy. Then ensure that all limits are set
> sanely. I am planning to implement shares support on to of resource counters.
>
>
>> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>>
>> ---
>> include/linux/res_counter.h | 11 ++++++++++-
>> kernel/res_counter.c | 36 ++++++++++++++++++++++++++++++++-------
>> mm/memcontrol.c | 9 ++++++---
>> 3 files changed, 45 insertions(+), 11 deletions(-)
>>
>> diff --git a/include/linux/res_counter.h b/include/linux/res_counter.h
>> index 2c4deb5..a27105e 100644
>> --- a/include/linux/res_counter.h
>> +++ b/include/linux/res_counter.h
>> @@ -41,6 +41,10 @@ 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;
>> };
>>
>> /**
>> @@ -80,7 +84,12 @@ 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.
>> diff --git a/kernel/res_counter.c b/kernel/res_counter.c
>> index f1f20c2..046f6f4 100644
>> --- a/kernel/res_counter.c
>> +++ b/kernel/res_counter.c
>> @@ -13,10 +13,11 @@
>> #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)
>> @@ -36,10 +37,26 @@ int res_counter_charge(struct res_counter *counter, unsigned long
val)
>>  {
>>   int ret;
>>   unsigned long flags;
>> + struct res_counter *c, *unroll_c;
>> +
>> + 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)
>> +   goto unroll;
>
> We'd like to know which resource counter failed to allow charging, so that we
> can reclaim from that mem_res_cgroup.
>
>> + }
>> + local_irq_restore(flags);
>> + return 0;
>>
>> - spin_lock_irqsave(&counter->lock, flags);
>> - ret = res_counter_charge_locked(counter, val);
>> - spin_unlock_irqrestore(&counter->lock, flags);
>> +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;
>>  }
>>
>> @@ -54,10 +71,15 @@ void res_counter_uncharge_locked(struct res_counter *counter,
unsigned long val)
```

>> 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 --git a/mm/memcontrol.c b/mm/memcontrol.c
>> index e5c741a..61db79c 100644
>> --- a/mm/memcontrol.c
>> +++ b/mm/memcontrol.c
>> @@ -976,19 +976,22 @@ static void free_mem_cgroup_per_zone_info(struct mem_cgroup
*mem, int node)
>> 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;
>>   init_mm.mem_cgroup = mem;
>> - } 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);
>>
>>   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 2/2] Make res_counter hierarchical
Posted by Balbir Singh on Tue, 11 Mar 2008 08:24:24 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> Balbir Singh wrote:
>> Pavel Emelyanov wrote:
>>> This allows us two things basically:
>>>
>>> 1. If the subgroup has the limit higher than its parent has
>>>   then the one will get more memory than allowed.
>> But should we allow such configuration? I suspect that we should catch such
>> things at the time of writing the limit.
>
> We cannot catch this at the limit-set-time. See, if you have a cgroup A
> with a 1GB limit and the usage is 999Mb, then creating a subgroup B with
> even 500MB limit will cause the A group consume 1.5GB of memory
> effectively.
>

No... If you propagate the charge of the child up to the parent, then it won't.
If each page charged to a child is also charged to the parent, this cannot
happen. The code you have below does that right?

>>> 2. When we will need to account for a resource in more than
>>>   one place, we'll be able to use this technics.
>>>
>>>   Look, consider we have a memory limit and swap limit. The
>>>   memory limit is the limit for the sum of RSS, page cache
>>>   and swap usage. To account for this gracefuly, we'll set
>>>   two counters:
>>>
>>>   res_counter mem_counter;
>>>   res_counter swap_counter;
>>>
>>>   attach mm to the swap one
>>>
>>>   mm->mem_cnt = &swap_counter;

>>>
>>> and make the swap_counter be mem's child. That's it. If we
>>> want hierarchical support, then the tree will look like this:
>>>
>>> mem_counter_top
>>>   swap_counter_top <- mm_struct living at top
>>>   mem_counter_sub
>>>   swap_counter_sub <- mm_struct living at sub
>>>
>> Hmm... not sure about this one. What I want to see is a resource counter
>> hierarchy to mimic the container hierarchy. Then ensure that all limits are set
>> sanely. I am planning to implement shares support on to of resource counters.
>>
>>
>>> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>>>
>>> ---
>>> include/linux/res_counter.h | 11 ++++++++++-
>>> kernel/res_counter.c | 36 ++++++++++++++++++++++++++++++------
>>> mm/memcontrol.c | 9 ++++++---
>>> 3 files changed, 45 insertions(+), 11 deletions(-)
>>>
>>> diff --git a/include/linux/res_counter.h b/include/linux/res_counter.h
>>> index 2c4deb5..a27105e 100644
>>> --- a/include/linux/res_counter.h
>>> +++ b/include/linux/res_counter.h
>>> @@ -41,6 +41,10 @@ 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;
>>> };
>>>
>>> /**
>>> @@ -80,7 +84,12 @@ 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.
>>> diff --git a/kernel/res_counter.c b/kernel/res_counter.c
>>> index f1f20c2..046f6f4 100644
>>> --- a/kernel/res_counter.c
>>> +++ b/kernel/res_counter.c
>>> @@ -13,10 +13,11 @@
>>> #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)
>>> @@ -36,10 +37,26 @@ int res_counter_charge(struct res_counter *counter, unsigned long
val)
>>> {
>>>   int ret;
>>>   unsigned long flags;
>>> + struct res_counter *c, *unroll_c;
>>> +
>>> + 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)
>>> +   goto unroll;
>> We'd like to know which resource counter failed to allow charging, so that we
>> can reclaim from that mem_res_cgroup.
>>

This is also important, so that we can reclaim from the nodes that go over their
limit.

>>> + }
>>> + local_irq_restore(flags);
>>> + return 0;
>>>
>>> - spin_lock_irqsave(&counter->lock, flags);
>>> - ret = res_counter_charge_locked(counter, val);
>>> - spin_unlock_irqrestore(&counter->lock, flags);

```
>>> +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;
>>>  }
>>>
>>> @@ -54,10 +71,15 @@ void res_counter_uncharge_locked(struct res_counter *counter,
unsigned long val)
>>>  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 --git a/mm/memcontrol.c b/mm/memcontrol.c
>>> index e5c741a..61db79c 100644
>>> --- a/mm/memcontrol.c
>>> +++ b/mm/memcontrol.c
>>> @@ -976,19 +976,22 @@ static void free_mem_cgroup_per_zone_info(struct mem_cgroup
*mem, int node)
>>>  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;
>>>    init_mm.mem_cgroup = mem;
>>> - } 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);
>>>
>>>   memset(&mem->info, 0, sizeof(mem->info));
>>>
>>
>
> --
> To unsubscribe, send a message with 'unsubscribe linux-mm' in
> the body to majordomo@kvack.org. For more info on Linux MM,
> see: http://www.linux-mm.org/ .
> Don't email: email@kvack.org

--
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: [PATCH 2/2] Make res_counter hierarchical
Posted by KAMEZAWA Hiroyuki on Tue, 11 Mar 2008 08:32:25 GMT
View Forum Message <> Reply to Message

On Tue, 11 Mar 2008 11:15:56 +0300
Pavel Emelyanov <xemul@openvz.org> wrote:
> > Hmm? seems strange.
> >
> > IMO, a parent's usage is just sum of all childs'.
> > And, historically, memory overcommit is done agaist "memory usage + swap".
> >
> > How about this ?
> >   <mem_counter_top, swap_counter_top>
> > <mem_counter_sub, swap_counter_sub>
> > <mem_counter_sub, swap_counter_sub>

> > <mem_counter_sub, swap_counter_sub>
> >
> > mem_counter_top.usage == sum of all mem_coutner_sub.usage
> > swap_counter_sub.usage = sum of all swap_counter_sub.usage
>
> I've misprinted in y tree, sorry.
> The correct hierarchy as I see it is
>
thank you.

> <mem_couter_0>
> + -- <swap_counter_0>
> + -- <mem_counter_1>
> | + -- <swap_counter_1>
> | + -- <mem_counter_11>
> | | + -- <swap_counter_11>
> | + -- <mem_counter_12>
> | + -- <swap_counter_12>
> + -- <mem_counter_2>
> | + -- <swap_counter_2>
> | + -- <mem_counter_21>
> | | + -- <swap_counter_21>
> | + -- <mem_counter_22>
> | + -- <swap_counter_22>
> + -- <mem_counter_N>
> + -- <swap_counter_N>
> + -- <mem_counter_N1>
> | + -- <swap_counter_N1>
> + -- <mem_counter_N2>
> + -- <swap_counter_N2>
>
please let me confirm.

- swap_counter_X.limit can be defined independent from mem_counter_X.limit ?
- swap_conter_N1's limit and swap_counter_N's have some relationship ?

Thanks,
-kame

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

---

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by Pavel Emelianov on Tue, 11 Mar 2008 08:38:50 GMT

KAMEZAWA Hiroyuki wrote:
> On Tue, 11 Mar 2008 11:15:56 +0300
> Pavel Emelyanov <xemul@openvz.org> wrote:
>>> Hmm? seems strange.
>>>
>>> IMO, a parent's usage is just sum of all childs'.
>>> And, historically, memory overcommit is done agaist "memory usage + swap".
>>>
>>> How about this ?
>>>   <mem_counter_top, swap_counter_top>
>>> <mem_counter_sub, swap_counter_sub>
>>> <mem_counter_sub, swap_counter_sub>
>>> <mem_counter_sub, swap_counter_sub>
>>>
>>>   mem_counter_top.usage == sum of all mem_coutner_sub.usage
>>>   swap_counter_sub.usage = sum of all swap_counter_sub.usage
>> I've misprinted in y tree, sorry.
>> The correct hierarchy as I see it is
>>
> thank you.
>
>> <mem_couter_0>
>> + -- <swap_counter_0>
>> + -- <mem_counter_1>
>> | + -- <swap_counter_1>
>> | + -- <mem_counter_11>
>> | | + -- <swap_counter_11>
>> | + -- <mem_counter_12>
>> | + -- <swap_counter_12>
>> + -- <mem_counter_2>
>> | + -- <swap_counter_2>
>> | + -- <mem_counter_21>
>> | | + -- <swap_counter_21>
>> | + -- <mem_counter_22>
>> | + -- <swap_counter_22>
>> + -- <mem_counter_N>
>>   + -- <swap_counter_N>
>>   + -- <mem_counter_N1>
>> | + -- <swap_counter_N1>
>>   + -- <mem_counter_N2>
>>   + -- <swap_counter_N2>
>>
> please let me confirm.
>
> - swap_counter_X.limit can be defined independent from mem_counter_X.limit ?
> - swap_conter_N1's limit and swap_counter_N's have some relationship ?

No. The mem_counter_N_limit is the limit for all the memory, that the
Nth group consumes. This includes the RSS, page cache and swap for this
group and all the child groups. Since RSS and page cache are accounted
together, this limit tracks the sum of (memory + swap) values over the
subtree started at the given group.

> Thanks,
> -kame
>
>

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

---

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by Pavel Emelianov on Tue, 11 Mar 2008 08:40:04 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelyanov wrote:
>> Balbir Singh wrote:
>>> Pavel Emelyanov wrote:
>>>> This allows us two things basically:
>>>>
>>>> 1. If the subgroup has the limit higher than its parent has
>>>> then the one will get more memory than allowed.
>>> But should we allow such configuration? I suspect that we should catch such
>>> things at the time of writing the limit.
>> We cannot catch this at the limit-set-time. See, if you have a cgroup A
>> with a 1GB limit and the usage is 999Mb, then creating a subgroup B with
>> even 500MB limit will cause the A group consume 1.5GB of memory
>> effectively.
>>
>
> No... If you propagate the charge of the child up to the parent, then it won't.
> If each page charged to a child is also charged to the parent, this cannot
> happen. The code you have below does that right?

Yup! What you described is available with this patch only.

>>>> 2. When we will need to account for a resource in more than
>>>> one place, we'll be able to use this technics.
>>>>
>>>> Look, consider we have a memory limit and swap limit. The
>>>> memory limit is the limit for the sum of RSS, page cache

>>>> and swap usage. To account for this gracefuly, we'll set
>>>> two counters:
>>>>
>>>>   res_counter mem_counter;
>>>>   res_counter swap_counter;
>>>>
>>>> attach mm to the swap one
>>>>
>>>>   mm->mem_cnt = &swap_counter;
>>>>
>>>> and make the swap_counter be mem's child. That's it. If we
>>>> want hierarchical support, then the tree will look like this:
>>>>
>>>> mem_counter_top
>>>>   swap_counter_top <- mm_struct living at top
>>>>   mem_counter_sub
>>>>   swap_counter_sub <- mm_struct living at sub
>>>>
>>> Hmm... not sure about this one. What I want to see is a resource counter
>>> hierarchy to mimic the container hierarchy. Then ensure that all limits are set
>>> sanely. I am planning to implement shares support on to of resource counters.
>>>
>>>
>>>> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>>>>
>>>> ---
>>>> include/linux/res_counter.h | 11 ++++++++++-
>>>> kernel/res_counter.c | 36 ++++++++++++++++++++++++++++++++-------
>>>> mm/memcontrol.c | 9 ++++++---
>>>> 3 files changed, 45 insertions(+), 11 deletions(-)
>>>>
>>>> diff --git a/include/linux/res_counter.h b/include/linux/res_counter.h
>>>> index 2c4deb5..a27105e 100644
>>>> --- a/include/linux/res_counter.h
>>>> +++ b/include/linux/res_counter.h
>>>> @@ -41,6 +41,10 @@ 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;
>>>> };
>>>>
>>>> /**
>>>> @@ -80,7 +84,12 @@ 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.
>>>> diff --git a/kernel/res_counter.c b/kernel/res_counter.c
>>>> index f1f20c2..046f6f4 100644
>>>> --- a/kernel/res_counter.c
>>>> +++ b/kernel/res_counter.c
>>>> @@ -13,10 +13,11 @@
>>>> #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)
>>>> @@ -36,10 +37,26 @@ int res_counter_charge(struct res_counter *counter, unsigned long
val)
>>>> {
>>>>   int ret;
>>>>   unsigned long flags;
>>>> + struct res_counter *c, *unroll_c;
>>>> +
>>>> + 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)
>>>> +   goto unroll;
>>> We'd like to know which resource counter failed to allow charging, so that we
>>> can reclaim from that mem_res_cgroup.
>>>
>>
> This is also important, so that we can reclaim from the nodes that go over their

> limit.

Agree. I'll think over how to provide this facility.

>>>> + }
>>>> + local_irq_restore(flags);
>>>> + return 0;
>>>>
>>>> - spin_lock_irqsave(&counter->lock, flags);
>>>> - ret = res_counter_charge_locked(counter, val);
>>>> - spin_unlock_irqrestore(&counter->lock, flags);
>>>> +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;
>>>> }
>>>>
>>>> @@ -54,10 +71,15 @@ void res_counter_uncharge_locked(struct res_counter *counter,
unsigned long val)
>>>> 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 --git a/mm/memcontrol.c b/mm/memcontrol.c
>>>> index e5c741a..61db79c 100644
>>>> --- a/mm/memcontrol.c
>>>> +++ b/mm/memcontrol.c
>>>> @@ -976,19 +976,22 @@ static void free_mem_cgroup_per_zone_info(struct
mem_cgroup *mem, int node)
>>>> 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;
>>>>    init_mm.mem_cgroup = mem;
>>>> - } 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);
>>>>
>>>>   memset(&mem->info, 0, sizeof(mem->info));
>>>>
>> --
>> To unsubscribe, send a message with 'unsubscribe linux-mm' in
>> the body to majordomo@kvack.org.  For more info on Linux MM,
>> see: http://www.linux-mm.org/ .
>> Don't email: <a href=mailto:"dont@kvack.org"> email@kvack.org </a>
>
>
```

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

---

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by Paul Menage on Tue, 11 Mar 2008 08:57:43 GMT
View Forum Message <> Reply to Message

On Tue, Mar 11, 2008 at 1:15 AM, Pavel Emelyanov <xemul@openvz.org> wrote:
>
> <mem_couter_0>
>   + -- <swap_counter_0>
>   + -- <mem_counter_1>
> | + -- <swap_counter_1>

> | + -- <mem_counter_11>
> | | + -- <swap_counter_11>
> | + -- <mem_counter_12>
> | + -- <swap_counter_12>
> + -- <mem_counter_2>
> | + -- <swap_counter_2>
> | + -- <mem_counter_21>
> | | + -- <swap_counter_21>
> | + -- <mem_counter_22>
> | + -- <swap_counter_22>
> + -- <mem_counter_N>
> + -- <swap_counter_N>
> + -- <mem_counter_N1>
> | + -- <swap_counter_N1>
> + -- <mem_counter_N2>
> + -- <swap_counter_N2>
>

The idea of hierarchy is good, but I don't think this particular hierarchy works for memory.

Main memory and swap space are very different resources, with very different performance characteristics. Suppose you have a 2G machine, and you want to guarantee each job 1GB of main memory, plus give them the option of 1GB of swap for when they go over the 1G main memory limit. With the hierarchy given above, you've need to give each job a 2GB mem.limit and a 1GB swap.limit, and so there would be no main memory isolation.

My feeling is that people are going to want to limit swap and main memory usage as two independent resource hierarchies more often than they're going to want to limit overall virtual memory. But assuming that there are people who need to do the latter, then you should make it configurable how the hierarchies fit together.

Alternatively, you could make it possible for a res_counter to have multiple parents (each of which constrains the overall usage of it and its siblings), and have three counters for each cgroup:

- vm_counter: overall virtual memory limit for group, parent = parent_mem_cgroup->vm_counter

- mem_counter: main memory limit for group, parents = vm_counter, parent_mem_cgroup->mem_counter

- swap_counter: swap limit for group, parents = vm_counter, parent_mem_cgroup->swap_counter

Paul

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

---

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by Daisuke Nishimura on Tue, 11 Mar 2008 09:03:20 GMT
View Forum Message <> Reply to Message

Hi.

> No. The mem_counter_N_limit is the limit for all the memory, that the
> Nth group consumes. This includes the RSS, page cache and swap for this
> group and all the child groups. Since RSS and page cache are accounted
> together, this limit tracks the sum of (memory + swap) values over the
> subtree started at the given group.
>
It seems a bit confusing for me, because current memcg manages
only RSS and page cache, not swap.

>>>> IMO, a parent's usage is just sum of all childs'.
>>>> And, historically, memory overcommit is done agaist "memory usage + swap".
>>>>
>>>> How about this ?
>>>> <mem_counter_top, swap_counter_top>
>>>> <mem_counter_sub, swap_counter_sub>
>>>> <mem_counter_sub, swap_counter_sub>
>>>> <mem_counter_sub, swap_counter_sub>
>>>>
>>>>   mem_counter_top.usage == sum of all mem_coutner_sub.usage
>>>>   swap_counter_sub.usage = sum of all swap_counter_sub.usage

I prefer Kamezawa-san's idea.

Thanks,
Daisuke Nishimura.

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

---

Subject: Re: [PATCH 2/2] Make res_counter hierarchical

On Tue, 11 Mar 2008 11:38:50 +0300
Pavel Emelyanov <xemul@openvz.org> wrote:

> >> <mem_couter_0>
> >> + -- <swap_counter_0>
> >> + -- <mem_counter_1>
> >> | + -- <swap_counter_1>
> >> | + -- <mem_counter_11>
> >> | | + -- <swap_counter_11>
> >> | + -- <mem_counter_12>
> >> | + -- <swap_counter_12>
> >> + -- <mem_counter_2>
> >> | + -- <swap_counter_2>
> >> | + -- <mem_counter_21>
> >> | | + -- <swap_counter_21>
> >> | + -- <mem_counter_22>
> >> | + -- <swap_counter_22>
> >> + -- <mem_counter_N>
> >>   + -- <swap_counter_N>
> >>   + -- <mem_counter_N1>
> >> | + -- <swap_counter_N1>
> >>   + -- <mem_counter_N2>
> >>   + -- <swap_counter_N2>
> >>
> > please let me confirm.
> >
> > - swap_counter_X.limit can be defined independent from mem_counter_X.limit ?
> > - swap_conter_N1's limit and swap_counter_N's have some relationship ?
>
> No. The mem_counter_N_limit is the limit for all the memory, that the
> Nth group consumes. This includes the RSS, page cache and swap for this
> group and all the child groups. Since RSS and page cache are accounted
> together, this limit tracks the sum of (memory + swap) values over the
> subtree started at the given group.
>

Hmm, how should I set limit to allow "tons of swap but small limit to memory".

Thanks,
-Kame

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by Paul Menage on Tue, 11 Mar 2008 09:11:49 GMT
View Forum Message <> Reply to Message

On Tue, Mar 11, 2008 at 2:13 AM, KAMEZAWA Hiroyuki
<kamezawa.hiroyu@jp.fujitsu.com> wrote:
> or remove all relationship among counters of *different* type of resources.
> user-land-daemon will do enough jobs.
>

Yes, that would be my preferred choice, if people agree that
hierarchically limiting overall virtual memory isn't useful. (I don't
think I have a use for it myself).

Paul

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

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by KAMEZAWA Hiroyuki on Tue, 11 Mar 2008 09:13:25 GMT
View Forum Message <> Reply to Message

On Tue, 11 Mar 2008 01:57:43 -0700
"Paul Menage" <menage@google.com> wrote:

> Alternatively, you could make it possible for a res_counter to have
> multiple parents (each of which constrains the overall usage of it and
> its siblings), and have three counters for each cgroup:
>
> - vm_counter: overall virtual memory limit for group, parent =
> parent_mem_cgroup->vm_counter
>
> - mem_counter: main memory limit for group, parents = vm_counter,
> parent_mem_cgroup->mem_counter
>
> - swap_counter: swap limit for group, parents = vm_counter,
> parent_mem_cgroup->swap_counter
>
or remove all relationship among counters of *different* type of resources.
user-land-daemon will do enough jobs.

Thanks,
-Kame

_____

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

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by Balbir Singh on Tue, 11 Mar 2008 09:16:58 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On Tue, Mar 11, 2008 at 2:13 AM, KAMEZAWA Hiroyuki
> <kamezawa.hiroyu@jp.fujitsu.com> wrote:
>> or remove all relationship among counters of *different* type of resources.
>> user-land-daemon will do enough jobs.
>>
>
> Yes, that would be my preferred choice, if people agree that
> hierarchically limiting overall virtual memory isn't useful. (I don't
> think I have a use for it myself).
>

Virtual limits are very useful. I have a patch ready to send out.
They limit the amount of paging a cgroup can do (virtual limit - RSS limit).
Some times end users want to set virtual limit == RSS limit, so that the cgroup
OOMs on cross the RSS limit.

--
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: [PATCH 2/2] Make res_counter hierarchical
Posted by KAMEZAWA Hiroyuki on Tue, 11 Mar 2008 09:39:03 GMT
View Forum Message <> Reply to Message

On Tue, 11 Mar 2008 14:46:58 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

> Paul Menage wrote:
> > On Tue, Mar 11, 2008 at 2:13 AM, KAMEZAWA Hiroyuki
> > <kamezawa.hiroyu@jp.fujitsu.com> wrote:

> >> or remove all relationship among counters of *different* type of resources.
> >> user-land-daemon will do enough jobs.
> >>
> >
> > Yes, that would be my preferred choice, if people agree that
> > hierarchically limiting overall virtual memory isn't useful. (I don't
> > think I have a use for it myself).
> >
>
> Virtual limits are very useful. I have a patch ready to send out.
> They limit the amount of paging a cgroup can do (virtual limit - RSS limit).
> Some times end users want to set virtual limit == RSS limit, so that the cgroup
> OOMs on cross the RSS limit.
>
I have no objection to adding virtual limit itself.
(It can be considered as extended ulimit.)

But if you'd like to add relationship between virtual-limit/memory-usage-limit,
please take care to make it clear that relationship is reaseonable.

- memory-usage includes page-cache.
- memory-usage doesn't include hugepages.
- How to treat MAP_NORESERVE is depends on over-commit-memory type.
  how cgroup does ?
- shared memory will be conuted per mmap.

Thanks,
-Kame

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

---

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by Balbir Singh on Tue, 11 Mar 2008 09:53:04 GMT
View Forum Message <> Reply to Message

KAMEZAWA Hiroyuki wrote:
> On Tue, 11 Mar 2008 14:46:58 +0530
> Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>
>> Paul Menage wrote:
>>> On Tue, Mar 11, 2008 at 2:13 AM, KAMEZAWA Hiroyuki
>>> <kamezawa.hiroyu@jp.fujitsu.com> wrote:
>>>> or remove all relationship among counters of *different* type of resources.

>>>> user-land-daemon will do enough jobs.
>>>>
>>> Yes, that would be my preferred choice, if people agree that
>>> hierarchically limiting overall virtual memory isn't useful. (I don't
>>> think I have a use for it myself).
>>>
>> Virtual limits are very useful. I have a patch ready to send out.
>> They limit the amount of paging a cgroup can do (virtual limit - RSS limit).
>> Some times end users want to set virtual limit == RSS limit, so that the cgroup
>> OOMs on cross the RSS limit.
>>
> I have no objection to adding virtual limit itself.
> (It can be considered as extended ulimit.)
>
> But if you'd like to add relationship between virtual-limit/memory-usage-limit,
> please take care to make it clear that relationship is reaseonable.
>

No, I don't want to add a relationship, just plain virtual memory limits and let
the system administrators determine what works for them.

--
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: [PATCH 2/2] Make res_counter hierarchical
Posted by KAMEZAWA Hiroyuki on Tue, 11 Mar 2008 10:03:18 GMT
View Forum Message <> Reply to Message

On Tue, 11 Mar 2008 15:23:04 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> > But if you'd like to add relationship between virtual-limit/memory-usage-limit,
> > please take care to make it clear that relationship is reaseonable.
> >
>
> No, I don't want to add a relationship, just plain virtual memory limits and let
> the system administrators determine what works for them.
>
ok :)

Thanks,
-Kame

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

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by Paul Menage on Tue, 11 Mar 2008 15:56:08 GMT
View Forum Message <> Reply to Message

On Tue, Mar 11, 2008 at 2:16 AM, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>
> Paul Menage wrote:
> > On Tue, Mar 11, 2008 at 2:13 AM, KAMEZAWA Hiroyuki
> > <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> >> or remove all relationship among counters of *different* type of resources.
> >> user-land-daemon will do enough jobs.
> >>
> >
> > Yes, that would be my preferred choice, if people agree that
> > hierarchically limiting overall virtual memory isn't useful. (I don't
> > think I have a use for it myself).
> >
>
> Virtual limits are very useful. I have a patch ready to send out.
> They limit the amount of paging a cgroup can do (virtual limit - RSS limit).

Ah, from this should I assume that you're talking about virtual
address space limits, not virtual memory limits?

My comment above was referring to Pavel's proposal to limit total
virtual memory (RAM + swap) for a cgroup, and then limit swap as a
subset of that, which basically makes it impossible to limit the RAM
usage of cgroups properly if you also want to allow swap usage.

Virtual address space limits are somewhat orthogonal to that.

Paul

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

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by Balbir Singh on Tue, 11 Mar 2008 15:59:38 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On Tue, Mar 11, 2008 at 2:16 AM, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>> Paul Menage wrote:
>> > On Tue, Mar 11, 2008 at 2:13 AM, KAMEZAWA Hiroyuki
>> > <kamezawa.hiroyu@jp.fujitsu.com> wrote:
>> >> or remove all relationship among counters of *different* type of resources.
>> >> user-land-daemon will do enough jobs.
>> >>
>> >
>> > Yes, that would be my preferred choice, if people agree that
>> > hierarchically limiting overall virtual memory isn't useful. (I don't
>> > think I have a use for it myself).
>> >
>>
>> Virtual limits are very useful. I have a patch ready to send out.
>> They limit the amount of paging a cgroup can do (virtual limit - RSS limit).
>
> Ah, from this should I assume that you're talking about virtual
> address space limits, not virtual memory limits?
>
> My comment above was referring to Pavel's proposal to limit total
> virtual memory (RAM + swap) for a cgroup, and then limit swap as a
> subset of that, which basically makes it impossible to limit the RAM
> usage of cgroups properly if you also want to allow swap usage.
>
> Virtual address space limits are somewhat orthogonal to that.
>

Yes, I was referring to Virtual address limits (along the lines of RLIMIT_AS). I guess it's just confusing terminology. I have patches for Virtual address limits. I should send them out soon.

--
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: [PATCH 2/2] Make res_counter hierarchical
Posted by yamamoto on Wed, 12 Mar 2008 23:05:41 GMT
View Forum Message <> Reply to Message

> @@ -36,10 +37,26 @@ int res_counter_charge(struct res_counter *counter, unsigned long val)
> {
> int ret;
> unsigned long flags;
> + struct res_counter *c, *unroll_c;
> +
> + 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)
> + goto unroll;
> + }
> + local_irq_restore(flags);
> + return 0;
>
> - spin_lock_irqsave(&counter->lock, flags);
> - ret = res_counter_charge_locked(counter, val);
> - spin_unlock_irqrestore(&counter->lock, flags);
> +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;
> }

what prevents the topology (in particular, ->parent pointers) from
changing behind us?

YAMAMOTO Takashi

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

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by yamamoto on Wed, 12 Mar 2008 23:36:50 GMT
View Forum Message <> Reply to Message

> > @@ -36,10 +37,26 @@ int res_counter_charge(struct res_counter *counter, unsigned long val)
> > {
> >   int ret;
> >   unsigned long flags;
> > + struct res_counter *c, *unroll_c;
> > +
> > + 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)
> > +   goto unroll;
> > + }
> > + local_irq_restore(flags);
> > + return 0;
> >
> > - spin_lock_irqsave(&counter->lock, flags);
> > - ret = res_counter_charge_locked(counter, val);
> > - spin_unlock_irqrestore(&counter->lock, flags);
> > +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;
> > }
>
> what prevents the topology (in particular, ->parent pointers) from
> changing behind us?
>
> YAMAMOTO Takashi

to answer myself: cgroupfs rename doesn't allow topological changes
in the first place.

btw, i think you need to do the same for res_counter_limit_check_locked
as well. i'm skeptical about doing these complicated stuffs in kernel,
esp. in this potentially performance critical code.

YAMAMOTO Takashi

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by Pavel Emelianov on Thu, 13 Mar 2008 08:56:58 GMT
View Forum Message <> Reply to Message

YAMAMOTO Takashi wrote:
>> @@ -36,10 +37,26 @@ int res_counter_charge(struct res_counter *counter, unsigned long val)
>> {
>>   int ret;
>>   unsigned long flags;
>> + struct res_counter *c, *unroll_c;
>> +
>> + 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)
>> +   goto unroll;
>> + }
>> + local_irq_restore(flags);
>> + return 0;
>>
>> - spin_lock_irqsave(&counter->lock, flags);
>> - ret = res_counter_charge_locked(counter, val);
>> - spin_unlock_irqrestore(&counter->lock, flags);
>> +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;
>> }
>
> what prevents the topology (in particular, ->parent pointers) from
> changing behind us?

The res_counter client must provide this. Currently cgroup subsystem does this.

> YAMAMOTO Takashi
>

Subject: Re: [PATCH 2/2] Make res_counter hierarchical
Posted by Balbir Singh on Wed, 02 Apr 2008 15:26:55 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> This allows us two things basically:
>

Pavel,

Do you have any further updates on this. I think we need a way of being able to implement reclaim per hierarchy as mentioned earlier. Do you want me to take a look at it?

--
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: [PATCH 2/2] Make res_counter hierarchical
Posted by Pavel Emelianov on Thu, 03 Apr 2008 12:06:06 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelyanov wrote:
>> This allows us two things basically:
>>
>
> Pavel,
>
> Do you have any further updates on this. I think we need a way of being able to

No. Unfortunately I stopped following the discussion at some point
and decided that nobody liked this patch that much.

> implement reclaim per hierarchy as mentioned earlier. Do you want me to take a
> look at it?

Yes, sure. I'm now busy (among other stuff) with kmemsize controller, hope I
can finish its polishing and testing till summer :(

_____