Subject: [PATCH v5 0/2] fix static_key disabling problem in memcg
Posted by Glauber Costa on Fri, 11 May 2012 20:11:15 GMT
View Forum Message <> Reply to Message

Hi, Tejun, Kame,

This series is composed of the two patches of the last fix, with no changes (only exception is the removal of x = false assignments that Tejun requested, that is done now). Note also that patch 1 of this series was reused by me in the slab accounting patches for memcg.

The first patch, that adds a mutex to memcg is dropped. I didn't posted it before so I could wait for Kame to get back from his vacations and properly review it.

Kame: Steven Rostedt pointed out that our analysis of the static branch updates were wrong, so the mutex is really not needed.

The key to understand that, is that atomic_inc_not_zero will only return right away if the value is not yet zero - as the name implies - but the update in the atomic variable only happens after the code is patched.

Therefore, if two callers enters with a key value of zero, both will be held at the jump_label_lock() call, effectively guaranteeing the behavior we need.

Glauber Costa (2):
  Always free struct memcg through schedule_work()
  decrement static keys on real destroy time

```
 include/net/sock.h        |    9 ++++++++
 mm/memcontrol.c           |   50 +++++++++++++++++++++++++++++++++++-----------
 net/ipv4/tcp_memcontrol.c |   32 +++++++++++++++++++++++------
 3 files changed, 71 insertions(+), 20 deletions(-)
```

--
1.7.7.6

Subject: [PATCH v5 1/2] Always free struct memcg through schedule_work()
Posted by Glauber Costa on Fri, 11 May 2012 20:11:16 GMT
View Forum Message <> Reply to Message

Right now we free struct memcg with kfree right after a rcu grace period, but defer it if we need to use vfree() to get rid of that memory area. We do that by need, because we need vfree to be called in a process context.

This patch unifies this behavior, by ensuring that even kfree will

happen in a separate thread. The goal is to have a stable place to
call the upcoming jump label destruction function outside the realm
of the complicated and quite far-reaching cgroup lock (that can't be
held when calling neither the cpu_hotplug.lock nor the jump_label_mutex)

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Tejun Heo <tj@kernel.org>
CC: Li Zefan <lizefan@huawei.com>
CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Johannes Weiner <hannes@cmpxchg.org>
CC: Michal Hocko <mhocko@suse.cz>
---
 mm/memcontrol.c | 24 ++++++++++++++++----------
 1 files changed, 13 insertions(+), 11 deletions(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 932a734..0b4b4c8 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -245,8 +245,8 @@ struct mem_cgroup {
    */
   struct rcu_head rcu_freeing;
   /*
-   * But when using vfree(), that cannot be done at
-   * interrupt time, so we must then queue the work.
+   * We also need some space for a worker in deferred freeing.
+   * By the time we call it, rcu_freeing is not longer in use.
    */
   struct work_struct work_freeing;
 };
@@ -4826,23 +4826,28 @@ out_free:
 }

 /*
- * Helpers for freeing a vzalloc()ed mem_cgroup by RCU,
+ * Helpers for freeing a kmalloc()ed/vzalloc()ed mem_cgroup by RCU,
  * but in process context. The work_freeing structure is overlaid
  * on the rcu_freeing structure, which itself is overlaid on memsw.
  */
-static void vfree_work(struct work_struct *work)
+static void free_work(struct work_struct *work)
 {
  struct mem_cgroup *memcg;
+ int size = sizeof(struct mem_cgroup);

  memcg = container_of(work, struct mem_cgroup, work_freeing);
- vfree(memcg);
+ if (size < PAGE_SIZE)

```
+  kfree(memcg);
+ else
+  vfree(memcg);
 }
-static void vfree_rcu(struct rcu_head *rcu_head)
+
+static void free_rcu(struct rcu_head *rcu_head)
 {
  struct mem_cgroup *memcg;

  memcg = container_of(rcu_head, struct mem_cgroup, rcu_freeing);
- INIT_WORK(&memcg->work_freeing, vfree_work);
+ INIT_WORK(&memcg->work_freeing, free_work);
  schedule_work(&memcg->work_freeing);
 }

@@ -4868,10 +4873,7 @@ static void __mem_cgroup_free(struct mem_cgroup *memcg)
   free_mem_cgroup_per_zone_info(memcg, node);

  free_percpu(memcg->stat);
- if (sizeof(struct mem_cgroup) < PAGE_SIZE)
-  kfree_rcu(memcg, rcu_freeing);
- else
-  call_rcu(&memcg->rcu_freeing, vfree_rcu);
+ call_rcu(&memcg->rcu_freeing, free_rcu);
 }

 static void mem_cgroup_get(struct mem_cgroup *memcg)
--
1.7.7.6
```

Subject: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by Glauber Costa on Fri, 11 May 2012 20:11:17 GMT
View Forum Message <> Reply to Message

We call the destroy function when a cgroup starts to be removed,
such as by a rmdir event.

However, because of our reference counters, some objects are still
inflight. Right now, we are decrementing the static_keys at destroy()
time, meaning that if we get rid of the last static_key reference,
some objects will still have charges, but the code to properly
uncharge them won't be run.

This becomes a problem specially if it is ever enabled again, because
now new charges will be added to the staled charges making keeping
it pretty much impossible.

We just need to be careful with the static branch activation:
since there is no particular preferred order of their activation,
we need to make sure that we only start using it after all
call sites are active. This is achieved by having a per-memcg
flag that is only updated after static_key_slow_inc() returns.
At this time, we are sure all sites are active.

This is made per-memcg, not global, for a reason:
it also has the effect of making socket accounting more
consistent. The first memcg to be limited will trigger static_key()
activation, therefore, accounting. But all the others will then be
accounted no matter what. After this patch, only limited memcgs
will have its sockets accounted.

[v2: changed a tcp limited flag for a generic proto limited flag]
[v3: update the current active flag only after the static_key update]
[v4: disarm_static_keys() inside free_work]
[v5: got rid of tcp_limit_mutex, now in the static_key interface]

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Tejun Heo <tj@kernel.org>
CC: Li Zefan <lizefan@huawei.com>
CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Johannes Weiner <hannes@cmpxchg.org>
CC: Michal Hocko <mhocko@suse.cz>
---
 include/net/sock.h | 9 +++++++++
 mm/memcontrol.c | 26 ++++++++++++++++++++++++--
 net/ipv4/tcp_memcontrol.c | 32 ++++++++++++++++++++++++++++++--
 3 files changed, 58 insertions(+), 9 deletions(-)

diff --git a/include/net/sock.h b/include/net/sock.h
index b3ebe6b..5c620bd 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -914,6 +914,15 @@ struct cg_proto {
  int   *memory_pressure;
  long   *sysctl_mem;
  /*
+ * active means it is currently active, and new sockets should
+ * be assigned to cgroups.
+ *
+ * activated means it was ever activated, and we need to
+ * disarm the static keys on destruction
+ */
+ bool   activated;
+ bool   active;

```
+ /*
   * memcg field is used to find which memcg we belong directly
   * Each memcg struct can hold more than one cg_proto, so container_of
   * won't really cut.
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 0b4b4c8..d1b0849 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -404,6 +404,7 @@ void sock_update_memcg(struct sock *sk)
 {
  if (mem_cgroup_sockets_enabled) {
   struct mem_cgroup *memcg;
+  struct cg_proto *cg_proto;

   BUG_ON(!sk->sk_prot->proto_cgroup);

@@ -423,9 +424,10 @@ void sock_update_memcg(struct sock *sk)

   rcu_read_lock();
   memcg = mem_cgroup_from_task(current);
-  if (!mem_cgroup_is_root(memcg)) {
+  cg_proto = sk->sk_prot->proto_cgroup(memcg);
+  if (!mem_cgroup_is_root(memcg) && cg_proto->active) {
    mem_cgroup_get(memcg);
-   sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
+   sk->sk_cgrp = cg_proto;
   }
   rcu_read_unlock();
  }
@@ -442,6 +444,14 @@ void sock_release_memcg(struct sock *sk)
  }
 }

+static void disarm_static_keys(struct mem_cgroup *memcg)
+{
+#ifdef CONFIG_INET
+ if (memcg->tcp_mem.cg_proto.activated)
+  static_key_slow_dec(&memcg_socket_limit_enabled);
+#endif
+}
+
 #ifdef CONFIG_INET
 struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
 {
@@ -452,6 +462,11 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
 }
 EXPORT_SYMBOL(tcp_proto_cgroup);
 #endif /* CONFIG_INET */
```

```
+#else
+static inline void disarm_static_keys(struct mem_cgroup *memcg)
+{
+}
+
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

 static void drain_all_stock_async(struct mem_cgroup *memcg);
@@ -4836,6 +4851,13 @@ static void free_work(struct work_struct *work)
  int size = sizeof(struct mem_cgroup);

  memcg = container_of(work, struct mem_cgroup, work_freeing);
+ /*
+  * We need to make sure that (at least for now), the jump label
+  * destruction code runs outside of the cgroup lock. schedule_work()
+  * will guarantee this happens. Be careful if you need to move this
+  * disarm_static_keys around
+  */
+ disarm_static_keys(memcg);
  if (size < PAGE_SIZE)
   kfree(memcg);
  else
diff --git a/net/ipv4/tcp_memcontrol.c b/net/ipv4/tcp_memcontrol.c
index 1517037..7ea4f79 100644
--- a/net/ipv4/tcp_memcontrol.c
+++ b/net/ipv4/tcp_memcontrol.c
@@ -74,9 +74,6 @@ void tcp_destroy_cgroup(struct mem_cgroup *memcg)
  percpu_counter_destroy(&tcp->tcp_sockets_allocated);

  val = res_counter_read_u64(&tcp->tcp_memory_allocated, RES_LIMIT);
-
- if (val != RESOURCE_MAX)
-  static_key_slow_dec(&memcg_socket_limit_enabled);
 }
 EXPORT_SYMBOL(tcp_destroy_cgroup);

@@ -107,10 +104,31 @@ static int tcp_update_limit(struct mem_cgroup *memcg, u64 val)
   tcp->tcp_prot_mem[i] = min_t(long, val >> PAGE_SHIFT,
           net->ipv4.sysctl_tcp_mem[i]);

- if (val == RESOURCE_MAX && old_lim != RESOURCE_MAX)
-  static_key_slow_dec(&memcg_socket_limit_enabled);
- else if (old_lim == RESOURCE_MAX && val != RESOURCE_MAX)
-  static_key_slow_inc(&memcg_socket_limit_enabled);
+ if (val == RESOURCE_MAX)
+  cg_proto->active = false;
+ else if (val != RESOURCE_MAX) {
+  /*
```

```
+   * ->activated needs to be written after the static_key update.
+   *  This is what guarantees that the socket activation function
+   *  is the last one to run. See sock_update_memcg() for details,
+   *  and note that we don't mark any socket as belonging to this
+   *  memcg until that flag is up.
+   *
+   *  We need to do this, because static_keys will span multiple
+   *  sites, but we can't control their order. If we mark a socket
+   *  as accounted, but the accounting functions are not patched in
+   *  yet, we'll lose accounting.
+   *
+   *  We never race with the readers in sock_update_memcg(), because
+   *  when this value change, the code to process it is not patched in
+   *  yet.
+   */
+  if (!cg_proto->activated) {
+   static_key_slow_inc(&memcg_socket_limit_enabled);
+   cg_proto->activated = true;
+  }
+  cg_proto->active = true;
+ }

  return 0;
 }
--
1.7.7.6
```

Subject: Re: [PATCH v5 1/2] Always free struct memcg through schedule_work()
Posted by KAMEZAWA Hiroyuki on Mon, 14 May 2012 00:56:20 GMT
View Forum Message <> Reply to Message

(2012/05/12 5:11), Glauber Costa wrote:

> Right now we free struct memcg with kfree right after a
> rcu grace period, but defer it if we need to use vfree() to get
> rid of that memory area. We do that by need, because we need vfree
> to be called in a process context.
>
> This patch unifies this behavior, by ensuring that even kfree will
> happen in a separate thread. The goal is to have a stable place to
> call the upcoming jump label destruction function outside the realm
> of the complicated and quite far-reaching cgroup lock (that can't be
> held when calling neither the cpu_hotplug.lock nor the jump_label_mutex)
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: Tejun Heo <tj@kernel.org>
> CC: Li Zefan <lizefan@huawei.com>

> CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> CC: Johannes Weiner <hannes@cmpxchg.org>
> CC: Michal Hocko <mhocko@suse.cz>

I think we'll need to revisit this, again.
for now,
Acked-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by KAMEZAWA Hiroyuki on Mon, 14 May 2012 00:59:04 GMT
View Forum Message <> Reply to Message

(2012/05/12 5:11), Glauber Costa wrote:

> We call the destroy function when a cgroup starts to be removed,
> such as by a rmdir event.
>
> However, because of our reference counters, some objects are still
> inflight. Right now, we are decrementing the static_keys at destroy()
> time, meaning that if we get rid of the last static_key reference,
> some objects will still have charges, but the code to properly
> uncharge them won't be run.
>
> This becomes a problem specially if it is ever enabled again, because
> now new charges will be added to the staled charges making keeping
> it pretty much impossible.
>
> We just need to be careful with the static branch activation:
> since there is no particular preferred order of their activation,
> we need to make sure that we only start using it after all
> call sites are active. This is achieved by having a per-memcg
> flag that is only updated after static_key_slow_inc() returns.
> At this time, we are sure all sites are active.
>
> This is made per-memcg, not global, for a reason:
> it also has the effect of making socket accounting more
> consistent. The first memcg to be limited will trigger static_key()
> activation, therefore, accounting. But all the others will then be
> accounted no matter what. After this patch, only limited memcgs
> will have its sockets accounted.
>
> [v2: changed a tcp limited flag for a generic proto limited flag]
> [v3: update the current active flag only after the static_key update]
> [v4: disarm_static_keys() inside free_work]
> [v5: got rid of tcp_limit_mutex, now in the static_key interface]
>

> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: Tejun Heo <tj@kernel.org>
> CC: Li Zefan <lizefan@huawei.com>
> CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> CC: Johannes Weiner <hannes@cmpxchg.org>
> CC: Michal Hocko <mhocko@suse.cz>

Thank you for your patient works.

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

BTW, what is the relationship between 1/2 and 2/2 ?

Thanks,
-Kame

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by Li Zefan on Mon, 14 May 2012 01:38:36 GMT
View Forum Message <> Reply to Message

> +static void disarm_static_keys(struct mem_cgroup *memcg)

> +{
> +#ifdef CONFIG_INET
> + if (memcg->tcp_mem.cg_proto.activated)
> + static_key_slow_dec(&memcg_socket_limit_enabled);
> +#endif
> +}

Move this inside the ifdef/endif below ?

Otherwise I think you'll get compile error if !CONFIG_INET...

> +
> #ifdef CONFIG_INET
> struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
> {
> @@ -452,6 +462,11 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
> }
> EXPORT_SYMBOL(tcp_proto_cgroup);
> #endif /* CONFIG_INET */
> +#else
> +static inline void disarm_static_keys(struct mem_cgroup *memcg)
> +{
> +}

> +
> #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by Tejun Heo on Mon, 14 May 2012 18:12:50 GMT
View Forum Message <> Reply to Message

On Fri, May 11, 2012 at 05:11:17PM -0300, Glauber Costa wrote:
> We call the destroy function when a cgroup starts to be removed,
> such as by a rmdir event.
>
> However, because of our reference counters, some objects are still
> inflight. Right now, we are decrementing the static_keys at destroy()
> time, meaning that if we get rid of the last static_key reference,
> some objects will still have charges, but the code to properly
> uncharge them won't be run.
>
> This becomes a problem specially if it is ever enabled again, because
> now new charges will be added to the staled charges making keeping
> it pretty much impossible.
>
> We just need to be careful with the static branch activation:
> since there is no particular preferred order of their activation,
> we need to make sure that we only start using it after all
> call sites are active. This is achieved by having a per-memcg
> flag that is only updated after static_key_slow_inc() returns.
> At this time, we are sure all sites are active.
>
> This is made per-memcg, not global, for a reason:
> it also has the effect of making socket accounting more
> consistent. The first memcg to be limited will trigger static_key()
> activation, therefore, accounting. But all the others will then be
> accounted no matter what. After this patch, only limited memcgs
> will have its sockets accounted.
>
> [v2: changed a tcp limited flag for a generic proto limited flag]
> [v3: update the current active flag only after the static_key update]
> [v4: disarm_static_keys() inside free_work]
> [v5: got rid of tcp_limit_mutex, now in the static_key interface]
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: Tejun Heo <tj@kernel.org>
> CC: Li Zefan <lizefan@huawei.com>
> CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> CC: Johannes Weiner <hannes@cmpxchg.org>
> CC: Michal Hocko <mhocko@suse.cz>

Generally looks sane to me. Please feel free to addmy Reviewed-by.

> + if (val == RESOURCE_MAX)
> + cg_proto->active = false;
> + else if (val != RESOURCE_MAX) {

Minor nitpick: CodingStyle says not to omit { } if other branches need
them.

Thanks.

--
tejun

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by Glauber Costa on Wed, 16 May 2012 06:03:08 GMT
View Forum Message <> Reply to Message

On 05/14/2012 04:59 AM, KAMEZAWA Hiroyuki wrote:
> (2012/05/12 5:11), Glauber Costa wrote:
>
>> We call the destroy function when a cgroup starts to be removed,
>> such as by a rmdir event.
>>
>> However, because of our reference counters, some objects are still
>> inflight. Right now, we are decrementing the static_keys at destroy()
>> time, meaning that if we get rid of the last static_key reference,
>> some objects will still have charges, but the code to properly
>> uncharge them won't be run.
>>
>> This becomes a problem specially if it is ever enabled again, because
>> now new charges will be added to the staled charges making keeping
>> it pretty much impossible.
>>
>> We just need to be careful with the static branch activation:
>> since there is no particular preferred order of their activation,
>> we need to make sure that we only start using it after all
>> call sites are active. This is achieved by having a per-memcg
>> flag that is only updated after static_key_slow_inc() returns.
>> At this time, we are sure all sites are active.
>>
>> This is made per-memcg, not global, for a reason:
>> it also has the effect of making socket accounting more
>> consistent. The first memcg to be limited will trigger static_key()
>> activation, therefore, accounting. But all the others will then be
>> accounted no matter what. After this patch, only limited memcgs
>> will have its sockets accounted.

>>
>> [v2: changed a tcp limited flag for a generic proto limited flag]
>> [v3: update the current active flag only after the static_key update]
>> [v4: disarm_static_keys() inside free_work]
>> [v5: got rid of tcp_limit_mutex, now in the static_key interface]
>>
>> Signed-off-by: Glauber Costa<glommer@parallels.com>
>> CC: Tejun Heo<tj@kernel.org>
>> CC: Li Zefan<lizefan@huawei.com>
>> CC: Kamezawa Hiroyuki<kamezawa.hiroyu@jp.fujitsu.com>
>> CC: Johannes Weiner<hannes@cmpxchg.org>
>> CC: Michal Hocko<mhocko@suse.cz>
>
>
> Thank you for your patient works.
>
> Acked-by: KAMEZAWA Hiroyuki<kamezawa.hiroyu@jp.fujitsu.com>
>
> BTW, what is the relationship between 1/2 and 2/2 ?

Can't do jump label patching inside an interrupt handler. They need to
happen when we free the structure, and I was about to add a worker
myself when I found out we already have one: just we don't always use it.

Before we merge it, let me just make sure the issue with config Li
pointed out don't exist. I did test it, but since I've reposted this
many times with multiple tiny changes - the type that will usually get
us killed, I'd be more comfortable with an extra round of testing if
someone spotted a possibility.

Who is merging this fix, btw ?
I find it to be entirely memcg related, even though it touches a file in
net (but a file with only memcg code in it)

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by Glauber Costa on Wed, 16 May 2012 07:03:47 GMT
View Forum Message <> Reply to Message

On 05/14/2012 05:38 AM, Li Zefan wrote:
>> +static void disarm_static_keys(struct mem_cgroup *memcg)
>
>> +{
>> +#ifdef CONFIG_INET
>> + if (memcg->tcp_mem.cg_proto.activated)
>> + static_key_slow_dec(&memcg_socket_limit_enabled);
>> +#endif
>> +}

>
>
> Move this inside the ifdef/endif below ?
>
> Otherwise I think you'll get compile error if !CONFIG_INET...

I don't fully get it.

We are supposed to provide a version of it for
CONFIG_CGROUP_MEM_RES_CTLR_KMEM and an empty version for
!CONFIG_CGROUP_MEM_RES_CTLR_KMEM

Inside the first, we take an action for CONFIG_INET, and no action for
!CONFIG_INET.

Bear in mind that the slab patches will add another test to that place,
and that's why I am doing it this way from the beginning.

Well, that said, I not only can be wrong, I very frequently am.

But I just compiled this one with and without CONFIG_INET, and it seems
to be going alright.

>> +
>> #ifdef CONFIG_INET
>> struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
>> {
>> @@ -452,6 +462,11 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
>> }
>> EXPORT_SYMBOL(tcp_proto_cgroup);
>> #endif /* CONFIG_INET */
>> +#else
>> +static inline void disarm_static_keys(struct mem_cgroup *memcg)
>> +{
>> +}
>> +
>> #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
>
> --
> To unsubscribe from this list: send the line "unsubscribe cgroups" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html

---

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by Glauber Costa on Wed, 16 May 2012 07:04:43 GMT

On 05/16/2012 10:03 AM, Glauber Costa wrote:
>> BTW, what is the relationship between 1/2 and 2/2 ?
> Can't do jump label patching inside an interrupt handler. They need to
> happen when we free the structure, and I was about to add a worker
> myself when I found out we already have one: just we don't always use it.
>
> Before we merge it, let me just make sure the issue with config Li
> pointed out don't exist. I did test it, but since I've reposted this
> many times with multiple tiny changes - the type that will usually get
> us killed, I'd be more comfortable with an extra round of testing if
> someone spotted a possibility.
>
> Who is merging this fix, btw ?
> I find it to be entirely memcg related, even though it touches a file in
> net (but a file with only memcg code in it)
>

For the record, I compiled test it many times, and the problem that Li
wondered about seems not to exist.

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by KAMEZAWA Hiroyuki on Wed, 16 May 2012 08:28:29 GMT

(2012/05/16 16:04), Glauber Costa wrote:

> On 05/16/2012 10:03 AM, Glauber Costa wrote:
>>> BTW, what is the relationship between 1/2 and 2/2 ?
>> Can't do jump label patching inside an interrupt handler. They need to
>> happen when we free the structure, and I was about to add a worker
>> myself when I found out we already have one: just we don't always use it.
>>
>> Before we merge it, let me just make sure the issue with config Li
>> pointed out don't exist. I did test it, but since I've reposted this
>> many times with multiple tiny changes - the type that will usually get
>> us killed, I'd be more comfortable with an extra round of testing if
>> someone spotted a possibility.
>>
>> Who is merging this fix, btw ?
>> I find it to be entirely memcg related, even though it touches a file in
>> net (but a file with only memcg code in it)
>>
>
> For the record, I compiled test it many times, and the problem that Li
> wondered about seems not to exist.

>

Ah...Hmm.....I guess dependency problem will be found in -mm if any rather than
netdev...

David, can this bug-fix patch goes via -mm tree ? Or will you pick up ?

CC'ed David Miller and Andrew Morton.

Thanks,
-Kame

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by Glauber Costa on Wed, 16 May 2012 08:30:25 GMT
View Forum Message <> Reply to Message

On 05/16/2012 12:28 PM, KAMEZAWA Hiroyuki wrote:
>> For the record, I compiled test it many times, and the problem that Li
>> > wondered about seems not to exist.
>> >
> Ah...Hmm.....I guess dependency problem will be found in -mm if any rather than
> netdev...

Yes. As I said, this only touches stuff in core memcg and the memcg
specific file. Any conflicts should come from other memcg fixes that may
have got into the tree...
> David, can this bug-fix patch goes via -mm tree ? Or will you pick up ?
>
> CC'ed David Miller and Andrew Morton.
>
> Thanks,
> -Kame
>
>

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by Glauber Costa on Wed, 16 May 2012 08:37:32 GMT
View Forum Message <> Reply to Message

On 05/16/2012 12:28 PM, KAMEZAWA Hiroyuki wrote:
> (2012/05/16 16:04), Glauber Costa wrote:
>
>> On 05/16/2012 10:03 AM, Glauber Costa wrote:
>>>> BTW, what is the relationship between 1/2 and 2/2 ?
>>> Can't do jump label patching inside an interrupt handler. They need to

>>> happen when we free the structure, and I was about to add a worker
>>> myself when I found out we already have one: just we don't always use it.
>>>
>>> Before we merge it, let me just make sure the issue with config Li
>>> pointed out don't exist. I did test it, but since I've reposted this
>>> many times with multiple tiny changes - the type that will usually get
>>> us killed, I'd be more comfortable with an extra round of testing if
>>> someone spotted a possibility.
>>>
>>> Who is merging this fix, btw ?
>>> I find it to be entirely memcg related, even though it touches a file in
>>> net (but a file with only memcg code in it)
>>>
>>
>> For the record, I compiled test it many times, and the problem that Li
>> wondered about seems not to exist.
>>
>
> Ah...Hmm.....I guess dependency problem will be found in -mm if any rather than
> netdev...
>
> David, can this bug-fix patch goes via -mm tree ? Or will you pick up ?
>

Another thing: Patch 2 in this series is of course dependent on patch 1
- which lives 100 % in memcg core. Without that, lockdep will scream
while disabling the static key.

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by akpm on Wed, 16 May 2012 20:57:55 GMT
View Forum Message <> Reply to Message

On Wed, 16 May 2012 11:03:47 +0400
Glauber Costa <glommer@parallels.com> wrote:

> On 05/14/2012 05:38 AM, Li Zefan wrote:
> >> +static void disarm_static_keys(struct mem_cgroup *memcg)
> >
> >> +{
> >> +#ifdef CONFIG_INET
> >> + if (memcg->tcp_mem.cg_proto.activated)
> >> + static_key_slow_dec(&memcg_socket_limit_enabled);
> >> +#endif
> >> +}
> >
> >
> > Move this inside the ifdef/endif below ?

> >
> > Otherwise I think you'll get compile error if !CONFIG_INET...
>
> I don't fully get it.
>
> We are supposed to provide a version of it for
> CONFIG_CGROUP_MEM_RES_CTLR_KMEM and an empty version for
> !CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>
> Inside the first, we take an action for CONFIG_INET, and no action for
> !CONFIG_INET.
>
> Bear in mind that the slab patches will add another test to that place,
> and that's why I am doing it this way from the beginning.
>
> Well, that said, I not only can be wrong, I very frequently am.
>
> But I just compiled this one with and without CONFIG_INET, and it seems
> to be going alright.
>

Yes, the ifdeffings in that area are rather nasty.

I wonder if it would be simpler to do away with the ifdef nesting.
At the top-level, just do

```
#if defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM) && defined(CONFIG_INET)
static void disarm_static_keys(struct mem_cgroup *memcg)
{
 if (memcg->tcp_mem.cg_proto.activated)
  static_key_slow_dec(&memcg_socket_limit_enabled);
}
#else
static inline void disarm_static_keys(struct mem_cgroup *memcg)
{
}
#endif
```

The tcp_proto_cgroup() definition could go inside that ifdef as well.

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by akpm on Wed, 16 May 2012 21:06:37 GMT
View Forum Message <> Reply to Message

On Fri, 11 May 2012 17:11:17 -0300
Glauber Costa <glommer@parallels.com> wrote:

> We call the destroy function when a cgroup starts to be removed,
> such as by a rmdir event.
>
> However, because of our reference counters, some objects are still
> inflight. Right now, we are decrementing the static_keys at destroy()
> time, meaning that if we get rid of the last static_key reference,
> some objects will still have charges, but the code to properly
> uncharge them won't be run.
>
> This becomes a problem specially if it is ever enabled again, because
> now new charges will be added to the staled charges making keeping
> it pretty much impossible.
>
> We just need to be careful with the static branch activation:
> since there is no particular preferred order of their activation,
> we need to make sure that we only start using it after all
> call sites are active. This is achieved by having a per-memcg
> flag that is only updated after static_key_slow_inc() returns.
> At this time, we are sure all sites are active.
>
> This is made per-memcg, not global, for a reason:
> it also has the effect of making socket accounting more
> consistent. The first memcg to be limited will trigger static_key()
> activation, therefore, accounting. But all the others will then be
> accounted no matter what. After this patch, only limited memcgs
> will have its sockets accounted.
>
> ...
>
> @@ -107,10 +104,31 @@ static int tcp_update_limit(struct mem_cgroup *memcg, u64 val)
> tcp->tcp_prot_mem[i] = min_t(long, val >> PAGE_SHIFT,
> net->ipv4.sysctl_tcp_mem[i]);
>
> - if (val == RESOURCE_MAX && old_lim != RESOURCE_MAX)
> - static_key_slow_dec(&memcg_socket_limit_enabled);
> - else if (old_lim == RESOURCE_MAX && val != RESOURCE_MAX)
> - static_key_slow_inc(&memcg_socket_limit_enabled);
> + if (val == RESOURCE_MAX)
> + cg_proto->active = false;
> + else if (val != RESOURCE_MAX) {
> + /*
> + * ->activated needs to be written after the static_key update.
> + * This is what guarantees that the socket activation function
> + * is the last one to run. See sock_update_memcg() for details,
> + * and note that we don't mark any socket as belonging to this
> + * memcg until that flag is up.
> + *

> + * We need to do this, because static_keys will span multiple
> + * sites, but we can't control their order. If we mark a socket
> + * as accounted, but the accounting functions are not patched in
> + * yet, we'll lose accounting.
> + *
> + * We never race with the readers in sock_update_memcg(), because
> + * when this value change, the code to process it is not patched in
> + * yet.
> + */
> + if (!cg_proto->activated) {
> + static_key_slow_inc(&memcg_socket_limit_enabled);
> + cg_proto->activated = true;
> + }

If two threads run this code concurrently, they can both see
cg_proto->activated==false and they will both run
static_key_slow_inc().

Hopefully there's some locking somewhere which prevents this, but it is
unobvious. We should comment this, probably at the cg_proto.activated
definition site. Or we should fix the bug ;)

> + cg_proto->active = true;
> +}
>
> return 0;
> }
>
> ...
>

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by akpm on Wed, 16 May 2012 21:13:42 GMT
View Forum Message <> Reply to Message

On Fri, 11 May 2012 17:11:17 -0300
Glauber Costa <glommer@parallels.com> wrote:

> We call the destroy function when a cgroup starts to be removed,
> such as by a rmdir event.
>
> However, because of our reference counters, some objects are still
> inflight. Right now, we are decrementing the static_keys at destroy()
> time, meaning that if we get rid of the last static_key reference,
> some objects will still have charges, but the code to properly
> uncharge them won't be run.

>
> This becomes a problem specially if it is ever enabled again, because
> now new charges will be added to the staled charges making keeping
> it pretty much impossible.
>
> We just need to be careful with the static branch activation:
> since there is no particular preferred order of their activation,
> we need to make sure that we only start using it after all
> call sites are active. This is achieved by having a per-memcg
> flag that is only updated after static_key_slow_inc() returns.
> At this time, we are sure all sites are active.
>
> This is made per-memcg, not global, for a reason:
> it also has the effect of making socket accounting more
> consistent. The first memcg to be limited will trigger static_key()
> activation, therefore, accounting. But all the others will then be
> accounted no matter what. After this patch, only limited memcgs
> will have its sockets accounted.

So I'm scratching my head over what the actual bug is, and how
important it is. AFAICT it will cause charging stats to exhibit some
inaccuracy when memcg's are being torn down?

I don't know how serious this in in the real world and so can't decide
which kernel version(s) we should fix.

When fixing bugs, please always fully describe the bug's end-user
impact, so that I and others can make these sorts of decisions.

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by KAMEZAWA Hiroyuki on Thu, 17 May 2012 00:07:11 GMT
View Forum Message <> Reply to Message

(2012/05/17 6:13), Andrew Morton wrote:

> On Fri, 11 May 2012 17:11:17 -0300
> Glauber Costa <glommer@parallels.com> wrote:
>
>> We call the destroy function when a cgroup starts to be removed,
>> such as by a rmdir event.
>>
>> However, because of our reference counters, some objects are still
>> inflight. Right now, we are decrementing the static_keys at destroy()
>> time, meaning that if we get rid of the last static_key reference,
>> some objects will still have charges, but the code to properly
>> uncharge them won't be run.
>>

>> This becomes a problem specially if it is ever enabled again, because
>> now new charges will be added to the staled charges making keeping
>> it pretty much impossible.
>>
>> We just need to be careful with the static branch activation:
>> since there is no particular preferred order of their activation,
>> we need to make sure that we only start using it after all
>> call sites are active. This is achieved by having a per-memcg
>> flag that is only updated after static_key_slow_inc() returns.
>> At this time, we are sure all sites are active.
>>
>> This is made per-memcg, not global, for a reason:
>> it also has the effect of making socket accounting more
>> consistent. The first memcg to be limited will trigger static_key()
>> activation, therefore, accounting. But all the others will then be
>> accounted no matter what. After this patch, only limited memcgs
>> will have its sockets accounted.
>
> So I'm scratching my head over what the actual bug is, and how
> important it is. AFAICT it will cause charging stats to exhibit some
> inaccuracy when memcg's are being torn down?
>
> I don't know how serious this in in the real world and so can't decide
> which kernel version(s) we should fix.
>
> When fixing bugs, please always fully describe the bug's end-user
> impact, so that I and others can make these sorts of decisions.
>

Ah, this was a bug report from me. tcp accounting can be easily broken.
Costa, could you include this ?
==

tcp memcontrol uses static_branch to optimize limit=RESOURCE_MAX case.
If all cgroup's limit=RESOUCE_MAX, resource usage is not accounted.
But it's buggy now.

For example, do following
while sleep 1;do
  echo 9223372036854775807 > /cgroup/memory/A/memory.kmem.tcp.limit_in_bytes;
  echo 300M > /cgroup/memory/A/memory.kmem.tcp.limit_in_bytes;
  done
and run network application under A. tcp's usage is sometimes accounted
and sometimes not accounted because of frequent changes of static_branch.
Then, you can see broken tcp.usage_in_bytes.
WARN_ON() is printed because res_counter->usage goes below 0.
==

kernel: ------------[cut here]----------
kernel: WARNING: at kernel/res_counter.c:96 res_counter_uncharge_locked+0x37/0x40()
 <snip>
kernel: Pid: 17753, comm: bash Tainted: G W   3.3.0+ #99
kernel: Call Trace:
kernel: <IRQ> [<ffffffff8104cc9f>] warn_slowpath_common+0x7f/0xc0
kernel: [<ffffffff810d7e88>] ? rb_reserve__next_event+0x68/0x470
kernel: [<ffffffff8104ccfa>] warn_slowpath_null+0x1a/0x20
kernel: [<ffffffff810b4e37>] res_counter_uncharge_locked+0x37/0x40
 ...
==

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by Glauber Costa on Thu, 17 May 2012 03:06:52 GMT
View Forum Message <> Reply to Message

On 05/17/2012 01:06 AM, Andrew Morton wrote:
> On Fri, 11 May 2012 17:11:17 -0300
> Glauber Costa<glommer@parallels.com> wrote:
>
>> We call the destroy function when a cgroup starts to be removed,
>> such as by a rmdir event.
>>
>> However, because of our reference counters, some objects are still
>> inflight. Right now, we are decrementing the static_keys at destroy()
>> time, meaning that if we get rid of the last static_key reference,
>> some objects will still have charges, but the code to properly
>> uncharge them won't be run.
>>
>> This becomes a problem specially if it is ever enabled again, because
>> now new charges will be added to the staled charges making keeping
>> it pretty much impossible.
>>
>> We just need to be careful with the static branch activation:
>> since there is no particular preferred order of their activation,
>> we need to make sure that we only start using it after all
>> call sites are active. This is achieved by having a per-memcg
>> flag that is only updated after static_key_slow_inc() returns.
>> At this time, we are sure all sites are active.
>>
>> This is made per-memcg, not global, for a reason:
>> it also has the effect of making socket accounting more
>> consistent. The first memcg to be limited will trigger static_key()
>> activation, therefore, accounting. But all the others will then be
>> accounted no matter what. After this patch, only limited memcgs
>> will have its sockets accounted.
>>

```
>> ...
>>
>> @@ -107,10 +104,31 @@ static int tcp_update_limit(struct mem_cgroup *memcg, u64 val)
>>     tcp->tcp_prot_mem[i] = min_t(long, val>>  PAGE_SHIFT,
>>             net->ipv4.sysctl_tcp_mem[i]);
>>
>> - if (val == RESOURCE_MAX&&  old_lim != RESOURCE_MAX)
>> -  static_key_slow_dec(&memcg_socket_limit_enabled);
>> - else if (old_lim == RESOURCE_MAX&&  val != RESOURCE_MAX)
>> -  static_key_slow_inc(&memcg_socket_limit_enabled);
>> + if (val == RESOURCE_MAX)
>> +  cg_proto->active = false;
>> + else if (val != RESOURCE_MAX) {
>> +  /*
>> +   * ->activated needs to be written after the static_key update.
>> +   *  This is what guarantees that the socket activation function
>> +   *  is the last one to run. See sock_update_memcg() for details,
>> +   *  and note that we don't mark any socket as belonging to this
>> +   *  memcg until that flag is up.
>> +   *
>> +   *  We need to do this, because static_keys will span multiple
>> +   *  sites, but we can't control their order. If we mark a socket
>> +   *  as accounted, but the accounting functions are not patched in
>> +   *  yet, we'll lose accounting.
>> +   *
>> +   *  We never race with the readers in sock_update_memcg(), because
>> +   *  when this value change, the code to process it is not patched in
>> +   *  yet.
>> +   */
>> +  if (!cg_proto->activated) {
>> +   static_key_slow_inc(&memcg_socket_limit_enabled);
>> +   cg_proto->activated = true;
>> +  }
>
> If two threads run this code concurrently, they can both see
> cg_proto->activated==false and they will both run
> static_key_slow_inc().
>
> Hopefully there's some locking somewhere which prevents this, but it is
> unobvious.  We should comment this, probably at the cg_proto.activated
> definition site.  Or we should fix the bug ;)
>
If that happens, locking in static_key_slow_inc will prevent any damage.
My previous version had explicit code to prevent that, but we were
pointed out that this is already part of the static_key expectations, so
that was dropped.
```

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by Glauber Costa on Thu, 17 May 2012 03:09:29 GMT
View Forum Message <> Reply to Message

On 05/17/2012 01:13 AM, Andrew Morton wrote:
> On Fri, 11 May 2012 17:11:17 -0300
> Glauber Costa<glommer@parallels.com> wrote:
>
>> We call the destroy function when a cgroup starts to be removed,
>> such as by a rmdir event.
>>
>> However, because of our reference counters, some objects are still
>> inflight. Right now, we are decrementing the static_keys at destroy()
>> time, meaning that if we get rid of the last static_key reference,
>> some objects will still have charges, but the code to properly
>> uncharge them won't be run.
>>
>> This becomes a problem specially if it is ever enabled again, because
>> now new charges will be added to the staled charges making keeping
>> it pretty much impossible.
>>
>> We just need to be careful with the static branch activation:
>> since there is no particular preferred order of their activation,
>> we need to make sure that we only start using it after all
>> call sites are active. This is achieved by having a per-memcg
>> flag that is only updated after static_key_slow_inc() returns.
>> At this time, we are sure all sites are active.
>>
>> This is made per-memcg, not global, for a reason:
>> it also has the effect of making socket accounting more
>> consistent. The first memcg to be limited will trigger static_key()
>> activation, therefore, accounting. But all the others will then be
>> accounted no matter what. After this patch, only limited memcgs
>> will have its sockets accounted.
>
> So I'm scratching my head over what the actual bug is, and how
> important it is. AFAICT it will cause charging stats to exhibit some
> inaccuracy when memcg's are being torn down?
>
> I don't know how serious this in in the real world and so can't decide
> which kernel version(s) we should fix.
>
> When fixing bugs, please always fully describe the bug's end-user
> impact, so that I and others can make these sorts of decisions.

Hi Andrew.

I believe that was described in patch 0/2 ?
In any case, this is something we need fixed, but it is not -stable

material or anything.

The bug leads to misaccounting when we quickly enable and disable limit
in a loop. We have a synthetic script to demonstrate that.

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by akpm on Thu, 17 May 2012 05:37:15 GMT
View Forum Message <> Reply to Message

On Thu, 17 May 2012 07:06:52 +0400 Glauber Costa <glommer@parallels.com> wrote:

> ...
> >> + else if (val != RESOURCE_MAX) {
> >> + /*
> >> +   * ->activated needs to be written after the static_key update.
> >> +   * This is what guarantees that the socket activation function
> >> +   * is the last one to run. See sock_update_memcg() for details,
> >> +   * and note that we don't mark any socket as belonging to this
> >> +   * memcg until that flag is up.
> >> +   *
> >> +   * We need to do this, because static_keys will span multiple
> >> +   * sites, but we can't control their order. If we mark a socket
> >> +   * as accounted, but the accounting functions are not patched in
> >> +   * yet, we'll lose accounting.
> >> +   *
> >> +   * We never race with the readers in sock_update_memcg(), because
> >> +   * when this value change, the code to process it is not patched in
> >> +   * yet.
> >> +   */
> >> + if (!cg_proto->activated) {
> >> +   static_key_slow_inc(&memcg_socket_limit_enabled);
> >> +   cg_proto->activated = true;
> >> + }
> >>
> > If two threads run this code concurrently, they can both see
> > cg_proto->activated==false and they will both run
> > static_key_slow_inc().
> >
> > Hopefully there's some locking somewhere which prevents this, but it is
> > unobvious. We should comment this, probably at the cg_proto.activated
> > definition site. Or we should fix the bug ;)
> >
> If that happens, locking in static_key_slow_inc will prevent any damage.
> My previous version had explicit code to prevent that, but we were
> pointed out that this is already part of the static_key expectations, so
> that was dropped.

This makes no sense. If two threads run that code concurrently,
key->enabled gets incremented twice. Nobody anywhere has a record that
this happened so it cannot be undone. key->enabled is now in an
unknown state.

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by Glauber Costa on Thu, 17 May 2012 09:52:13 GMT
View Forum Message <> Reply to Message

On 05/17/2012 09:37 AM, Andrew Morton wrote:
>> > If that happens, locking in static_key_slow_inc will prevent any damage.
>> > My previous version had explicit code to prevent that, but we were
>> > pointed out that this is already part of the static_key expectations, so
>> > that was dropped.
> This makes no sense. If two threads run that code concurrently,
> key->enabled gets incremented twice. Nobody anywhere has a record that
> this happened so it cannot be undone. key->enabled is now in an
> unknown state.

Kame, Tejun,

Andrew is right. It seems we will need that mutex after all. Just this
is not a race, and neither something that should belong in the
static_branch interface.

We want to make sure that enabled is not updated before the jump label
update, because we need a specific ordering guarantee at the patched
sites. And *that*, the interface guarantees, and we were wrong to
believe it did not. That is a correction issue for the accounting, and
that part is right.

But when we disarm it, we'll need to make sure that happened only once,
otherwise we may never unpatch it. That, or we'd need that to be a
counter. The jump label interface does not - and should not - keep track
of how many updates happened to a key. That's the role of whoever is
using it.

If you agree with the above, I'll send this patch again with the correction.

Andrew, thank you very much. Do you spot anything else here?

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by KAMEZAWA Hiroyuki on Thu, 17 May 2012 10:18:09 GMT
View Forum Message <> Reply to Message

(2012/05/17 18:52), Glauber Costa wrote:

> On 05/17/2012 09:37 AM, Andrew Morton wrote:
>>>> If that happens, locking in static_key_slow_inc will prevent any damage.
>>>> My previous version had explicit code to prevent that, but we were
>>>> pointed out that this is already part of the static_key expectations, so
>>>> that was dropped.
>> This makes no sense. If two threads run that code concurrently,
>> key->enabled gets incremented twice. Nobody anywhere has a record that
>> this happened so it cannot be undone. key->enabled is now in an
>> unknown state.
>
> Kame, Tejun,
>
> Andrew is right. It seems we will need that mutex after all. Just this
> is not a race, and neither something that should belong in the
> static_branch interface.
>

Hmm....how about having

res_counter_xchg_limit(res, &old_limit, new_limit);

if (!cg_proto->updated && old_limit == RESOURCE_MAX)
 ....update labels...

Then, no mutex overhead maybe and activated will be updated only once.
Ah, but please fix in a way you like. Above is an example.

Thanks,
-Kame
(*) I'm sorry I won't be able to read e-mails, tomorrow.

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by Glauber Costa on Thu, 17 May 2012 10:22:05 GMT
View Forum Message <> Reply to Message

On 05/17/2012 02:18 PM, KAMEZAWA Hiroyuki wrote:
> (2012/05/17 18:52), Glauber Costa wrote:
>
>> On 05/17/2012 09:37 AM, Andrew Morton wrote:
>>>>> If that happens, locking in static_key_slow_inc will prevent any damage.
>>>>> My previous version had explicit code to prevent that, but we were
>>>>> pointed out that this is already part of the static_key expectations, so
>>>>> that was dropped.
>>> This makes no sense. If two threads run that code concurrently,

>>> key->enabled gets incremented twice. Nobody anywhere has a record that
>>> this happened so it cannot be undone. key->enabled is now in an
>>> unknown state.
>>
>> Kame, Tejun,
>>
>> Andrew is right. It seems we will need that mutex after all. Just this
>> is not a race, and neither something that should belong in the
>> static_branch interface.
>>
>
>
> Hmm....how about having
>
> res_counter_xchg_limit(res,&old_limit, new_limit);
>
> if (!cg_proto->updated&& old_limit == RESOURCE_MAX)
> ....update labels...
>
> Then, no mutex overhead maybe and activated will be updated only once.
> Ah, but please fix in a way you like. Above is an example.

I think a mutex is a lot cleaner than adding a new function to the
res_counter interface.

We could do a counter, and then later decrement the key until the
counter reaches zero, but between those two, I still think a mutex here
is preferable.

Only that, instead of coming up with a mutex of ours, we could export
and reuse set_limit_mutex from memcontrol.c

> Thanks,
> -Kame
> (*) I'm sorry I won't be able to read e-mails, tomorrow.
>
Ok Kame. I am not in a terrible hurry to fix this, it doesn't seem to be
hurting any real workload.

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by KAMEZAWA Hiroyuki on Thu, 17 May 2012 10:27:09 GMT
View Forum Message <> Reply to Message

(2012/05/17 19:22), Glauber Costa wrote:

> On 05/17/2012 02:18 PM, KAMEZAWA Hiroyuki wrote:

>> (2012/05/17 18:52), Glauber Costa wrote:
>>
>>> On 05/17/2012 09:37 AM, Andrew Morton wrote:
>>>>>> If that happens, locking in static_key_slow_inc will prevent any damage.
>>>>>> My previous version had explicit code to prevent that, but we were
>>>>>> pointed out that this is already part of the static_key expectations, so
>>>>>> that was dropped.
>>>> This makes no sense. If two threads run that code concurrently,
>>>> key->enabled gets incremented twice. Nobody anywhere has a record that
>>>> this happened so it cannot be undone. key->enabled is now in an
>>>> unknown state.
>>>
>>> Kame, Tejun,
>>>
>>> Andrew is right. It seems we will need that mutex after all. Just this
>>> is not a race, and neither something that should belong in the
>>> static_branch interface.
>>>
>>
>>
>> Hmm....how about having
>>
>> res_counter_xchg_limit(res,&old_limit, new_limit);
>>
>> if (!cg_proto->updated&& old_limit == RESOURCE_MAX)
>> ....update labels...
>>
>> Then, no mutex overhead maybe and activated will be updated only once.
>> Ah, but please fix in a way you like. Above is an example.
>
> I think a mutex is a lot cleaner than adding a new function to the
> res_counter interface.
>
> We could do a counter, and then later decrement the key until the
> counter reaches zero, but between those two, I still think a mutex here
> is preferable.
>
> Only that, instead of coming up with a mutex of ours, we could export
> and reuse set_limit_mutex from memcontrol.c
>

ok, please.

thx,
-Kame

>

>> Thanks,
>> -Kame
>> (*) I'm sorry I won't be able to read e-mails, tomorrow.
>>
> Ok Kame. I am not in a terrible hurry to fix this, it doesn't seem to be
> hurting any real workload.
>
>

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by Tejun Heo on Thu, 17 May 2012 15:19:47 GMT
View Forum Message <> Reply to Message

On Thu, May 17, 2012 at 01:52:13PM +0400, Glauber Costa wrote:
> Andrew is right. It seems we will need that mutex after all. Just
> this is not a race, and neither something that should belong in the
> static_branch interface.

Yeah, with a completely different comment. It just needs to wrap
->activated alteration and static key inc/dec, right?

Thanks.

--
tejun

Subject: Re: [PATCH v5 2/2] decrement static keys on real destroy time
Posted by akpm on Thu, 17 May 2012 17:02:53 GMT
View Forum Message <> Reply to Message

On Thu, 17 May 2012 13:52:13 +0400 Glauber Costa <glommer@parallels.com> wrote:

> Andrew is right. It seems we will need that mutex after all. Just this
> is not a race, and neither something that should belong in the
> static_branch interface.

Well, a mutex is one way. Or you could do something like

```
 if (!test_and_set_bit(CGPROTO_ACTIVATED, &cg_proto->flags))
  static_key_slow_inc(&memcg_socket_limit_enabled);
```
