Subject: [PATCH v6 0/2] fix static_key disabling problem in memcg
Posted by Glauber Costa on Tue, 22 May 2012 10:25:37 GMT
View Forum Message <> Reply to Message

Andrew,

This is a respin of the last fixes I sent for sock memcg problems with
the static_keys enablement. The first patch is still unchanged, and for
the second, I am using a flags field as for your suggestion.
Indeed, I found a flags field to be more elegant, while still
maintaining a fast path for the readers.

Kame, will you please take a look and see if this would work okay?

Tejun, are you happy with the current state of the comments explaining
the scenario?

Thank you very much for your time

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

```
 include/linux/memcontrol.h |    5 ++++
 include/net/sock.h         |   11 +++++++++
 mm/memcontrol.c            |   53 +++++++++++++++++++++++++++++++++----------
 net/ipv4/tcp_memcontrol.c  |   34 +++++++++++++++++++++++-----
 4 files changed, 83 insertions(+), 20 deletions(-)
```

--
1.7.7.6

Subject: [PATCH v6 1/2] Always free struct memcg through schedule_work()
Posted by Glauber Costa on Tue, 22 May 2012 10:25:38 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>
Acked-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Tejun Heo <tj@kernel.org>
CC: Li Zefan <lizefan@huawei.com>
CC: Johannes Weiner <hannes@cmpxchg.org>
CC: Michal Hocko <mhocko@suse.cz>
CC: Andrew Morton <akpm@linux-foundation.org>
---
 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 v6 2/2] decrement static keys on real destroy time
Posted by Glauber Costa on Tue, 22 May 2012 10:25:39 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]
[v6: changed active and activated to a flags field, as suggested by akpm]

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>
CC: Andrew Morton <akpm@linux-foundation.org>
---
 include/linux/memcontrol.h | 5 +++++
 include/net/sock.h | 11 ++++++++++
 mm/memcontrol.c | 29 ++++++++++++++++++++++++++--
 net/ipv4/tcp_memcontrol.c | 34 ++++++++++++++++++++++++++++++++-------
 4 files changed, 70 insertions(+), 9 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index f94efd2..9dc0b86 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -436,6 +436,11 @@ enum {
  OVER_LIMIT,
 };

+enum sock_flag_bits {
+ MEMCG_SOCK_ACTIVE,
+ MEMCG_SOCK_ACTIVATED,
+};
+
 struct sock;
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
 void sock_update_memcg(struct sock *sk);
diff --git a/include/net/sock.h b/include/net/sock.h

```
index b3ebe6b..1742db7 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -913,6 +913,7 @@ struct cg_proto {
  struct percpu_counter *sockets_allocated; /* Current number of sockets. */
  int   *memory_pressure;
  long   *sysctl_mem;
+ unsigned long  flags;
  /*
   * memcg field is used to find which memcg we belong directly
   * Each memcg struct can hold more than one cg_proto, so container_of
@@ -928,6 +929,16 @@ struct cg_proto {
 extern int proto_register(struct proto *prot, int alloc_slab);
 extern void proto_unregister(struct proto *prot);

+static inline bool memcg_proto_active(struct cg_proto *cg_proto)
+{
+ return cg_proto->flags & (1 << MEMCG_SOCK_ACTIVE);
+}
+
+static inline bool memcg_proto_activated(struct cg_proto *cg_proto)
+{
+ return cg_proto->flags & (1 << MEMCG_SOCK_ACTIVATED);
+}
+
 #ifdef SOCK_REFCNT_DEBUG
 static inline void sk_refcnt_debug_inc(struct sock *sk)
 {
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 0b4b4c8..22434bf 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) && memcg_proto_active(cg_proto)) {
    mem_cgroup_get(memcg);
```

```
-   sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
+   sk->sk_cgrp = cg_proto;
    }
    rcu_read_unlock();
   }
@@ -451,9 +453,25 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
  return &memcg->tcp_mem.cg_proto;
 }
 EXPORT_SYMBOL(tcp_proto_cgroup);
+
+static void disarm_sock_keys(struct mem_cgroup *memcg)
+{
+ if (!memcg_proto_activated(&memcg->tcp_mem.cg_proto))
+  return;
+ static_key_slow_dec(&memcg_socket_limit_enabled);
+}
+#else
+static void disarm_sock_keys(struct mem_cgroup *memcg)
+{
+}
 #endif /* CONFIG_INET */
 #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */

+static void disarm_static_keys(struct mem_cgroup *memcg)
+{
+ disarm_sock_keys(memcg);
+}
+
 static void drain_all_stock_async(struct mem_cgroup *memcg);

 static struct mem_cgroup_per_zone *
@@ -4836,6 +4854,13 @@ 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..3b8fa25 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 +104,33 @@ 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)
+  clear_bit(MEMCG_SOCK_ACTIVE, &cg_proto->flags);
+ else if (val != RESOURCE_MAX) {
+  /*
+   *  The active bit 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.
+   *
+   *  The activated bit is used to guarantee that no two writers will
+   *  do the update in the same memcg. Without that, we can't properly
+   *  shutdown the static key.
+   */
+  if (!test_and_set_bit(MEMCG_SOCK_ACTIVATED, &cg_proto->flags))
+   static_key_slow_inc(&memcg_socket_limit_enabled);
+  set_bit(MEMCG_SOCK_ACTIVE, &cg_proto->flags);
+ }

  return 0;
```

}
--
1.7.7.6

Subject: Re: [PATCH v6 2/2] decrement static keys on real destroy time
Posted by akpm on Tue, 22 May 2012 22:46:10 GMT
View Forum Message <> Reply to Message

(cc davem)

On Tue, 22 May 2012 14:25:39 +0400
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.
>
> [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]
> [v6: changed active and activated to a flags field, as suggested by akpm]

A few things...

> include/linux/memcontrol.h | 5 +++++
> include/net/sock.h | 11 ++++++++++
> mm/memcontrol.c | 29 ++++++++++++++++++++++++++--
> net/ipv4/tcp_memcontrol.c | 34 ++++++++++++++++++++++++++-------
> 4 files changed, 70 insertions(+), 9 deletions(-)
>
> diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
> index f94efd2..9dc0b86 100644
> --- a/include/linux/memcontrol.h
> +++ b/include/linux/memcontrol.h
> @@ -436,6 +436,11 @@ enum {
>   OVER_LIMIT,
> };
>
> +enum sock_flag_bits {
> + MEMCG_SOCK_ACTIVE,
> + MEMCG_SOCK_ACTIVATED,
> +};

I don't see why this was defined in memcontrol.h. It is enumerating
the bits in sock.h's cg_proto.flags, so why not define it in sock.h?
This is changed in the appended patch.

Also, in the v5 patch these flags were documented, as they should be.
Version 6 forgot to do this. This is changed in the appended patch.

And version 6 doesn't describe what sock_flag_bits actually does. It
should. This is changed in the appended patch.

And the name seems inappropriate to me. Should it not be enum
cg_proto_flag_bits? Or, probably better, cg_proto_flags? This I did
not change.

> struct sock;
> #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
> void sock_update_memcg(struct sock *sk);
> diff --git a/include/net/sock.h b/include/net/sock.h
> index b3ebe6b..1742db7 100644
> --- a/include/net/sock.h
> +++ b/include/net/sock.h
> @@ -913,6 +913,7 @@ struct cg_proto {
>   struct percpu_counter *sockets_allocated; /* Current number of sockets. */
>   int   *memory_pressure;
>   long   *sysctl_mem;
> + unsigned long flags;
>   /*
>   * memcg field is used to find which memcg we belong directly

>   * Each memcg struct can hold more than one cg_proto, so container_of
> @@ -928,6 +929,16 @@ struct cg_proto {
> extern int proto_register(struct proto *prot, int alloc_slab);
> extern void proto_unregister(struct proto *prot);
>
> +static inline bool memcg_proto_active(struct cg_proto *cg_proto)
> +{
> + return cg_proto->flags & (1 << MEMCG_SOCK_ACTIVE);
> +}
> +
> +static inline bool memcg_proto_activated(struct cg_proto *cg_proto)
> +{
> + return cg_proto->flags & (1 << MEMCG_SOCK_ACTIVATED);
> +}

Here, we're open-coding kinda-test_bit(). Why do that? These flags are
modified with set_bit() and friends, so we should read them with the
matching test_bit()?

Also, these bool-returning functions will return values other than 0
and 1. That probably works OK and I don't know what the C standards
and implementations do about this. But it seems unclean and slightly
risky to have a "bool" value of 32! Converting these functions to use
test_bit() fixes this - test_bit() returns only 0 or 1.

test_bit() is slightly more expensive than the above. If this is
considered to be an issue then I guess we could continue to use this
approach. But I do think a code comment is needed, explaining and
justifying the unusual decision to bypass the bitops API. Also these
functions should tell the truth and return an "int" type.

> #ifdef SOCK_REFCNT_DEBUG
> static inline void sk_refcnt_debug_inc(struct sock *sk)
> {
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 0b4b4c8..22434bf 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) && memcg_proto_active(cg_proto)) {
>     mem_cgroup_get(memcg);
> -   sk->sk_cgrp = sk->sk_prot->proto_cgroup(memcg);
> +   sk->sk_cgrp = cg_proto;
>    }
>    rcu_read_unlock();
>   }
> @@ -451,9 +453,25 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
>   return &memcg->tcp_mem.cg_proto;
>  }
>  EXPORT_SYMBOL(tcp_proto_cgroup);
> +
> +static void disarm_sock_keys(struct mem_cgroup *memcg)
> +{
> + if (!memcg_proto_activated(&memcg->tcp_mem.cg_proto))
> +  return;
> + static_key_slow_dec(&memcg_socket_limit_enabled);
> +}
> +#else
> +static void disarm_sock_keys(struct mem_cgroup *memcg)
> +{
> +}
>  #endif /* CONFIG_INET */
>  #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
>
> +static void disarm_static_keys(struct mem_cgroup *memcg)
> +{
> + disarm_sock_keys(memcg);
> +}
```

Why does this function exist? Its single caller could call
disarm_sock_keys() directly.

```
>  static void drain_all_stock_async(struct mem_cgroup *memcg);
>
>  static struct mem_cgroup_per_zone *
> @@ -4836,6 +4854,13 @@ 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.
```

This is a poor comment - it failed to tell the reader *why* that code
must run outside the cgroup lock.

>   schedule_work()
> + * will guarantee this happens. Be careful if you need to move this
> + * disarm_static_keys around

It's a bit difficult for the reader to be careful when he isn't told
what the risks are.

> + */
> + 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..3b8fa25 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 +104,33 @@ 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)
> + clear_bit(MEMCG_SOCK_ACTIVE, &cg_proto->flags);
> + else if (val != RESOURCE_MAX) {
> + /*
> +   * The active bit 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.
> + *
> + * The activated bit is used to guarantee that no two writers will
> + * do the update in the same memcg. Without that, we can't properly
> + * shutdown the static key.
> + */

This comment needlessly overflows 80 cols and has a pointless and
unconventional double-space indenting. I already provided a patch
which fixes this and a few other things, but that was ignored when you
did the v6.

> + if (!test_and_set_bit(MEMCG_SOCK_ACTIVATED, &cg_proto->flags))
> +   static_key_slow_inc(&memcg_socket_limit_enabled);
> + set_bit(MEMCG_SOCK_ACTIVE, &cg_proto->flags);
> + }

So here are suggested changes from *some* of the above discussion.
Please consider, incorporate, retest and send us a v7?

From: Andrew Morton <akpm@linux-foundation.org>
Subject: memcg-decrement-static-keys-at-real-destroy-time-v6-fix

- move enum sock_flag_bits into sock.h

- document enum sock_flag_bits

- convert memcg_proto_active() and memcg_proto_activated() to test_bit()

- redo tcp_update_limit() comment to 80 cols

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

```
 include/linux/memcontrol.h |    5 -----
 include/net/sock.h         |   15 ++++++++++++--
 net/ipv4/tcp_memcontrol.c  |   30 ++++++++++++++++++++++++----------
 3 files changed, 28 insertions(+), 22 deletions(-)

diff -puN  include/linux/memcontrol.h~memcg-decrement-static-keys-at-re al-destroy-time-v6-fix
include/linux/memcontrol.h
---  a/include/linux/memcontrol.h~memcg-decrement-static-keys-at- real-destroy-time-v6-fix
+++ a/include/linux/memcontrol.h
@@ -405,11 +405,6 @@ enum {
  OVER_LIMIT,
 };

-enum sock_flag_bits {
- MEMCG_SOCK_ACTIVE,
- MEMCG_SOCK_ACTIVATED,
-};
-
 struct sock;
 #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
 void sock_update_memcg(struct sock *sk);
diff -puN  include/net/sock.h~memcg-decrement-static-keys-at-real-destr oy-time-v6-fix
include/net/sock.h
---  a/include/net/sock.h~memcg-decrement-static-keys-at-real-des troy-time-v6-fix
+++ a/include/net/sock.h
@@ -46,6 +46,7 @@
 #include <linux/list_nulls.h>
 #include <linux/timer.h>
 #include <linux/cache.h>
+#include <linux/bitops.h>
 #include <linux/lockdep.h>
 #include <linux/netdevice.h>
 #include <linux/skbuff.h> /* struct sk_buff */
@@ -921,6 +922,16 @@ struct proto {
 #endif
 };

+/*
+ * Bits in struct cg_proto.flags
+ */
+enum sock_flag_bits {
+ /* Currently active and new sockets should be assigned to cgroups */
+ MEMCG_SOCK_ACTIVE,
+ /* It was ever activated; we must disarm static keys on destruction */
+ MEMCG_SOCK_ACTIVATED,
+};
+
 struct cg_proto {
```

```
  void   (*enter_memory_pressure)(struct sock *sk);
  struct res_counter *memory_allocated; /* Current allocated memory. */
@@ -945,12 +956,12 @@ extern void proto_unregister(struct prot

 static inline bool memcg_proto_active(struct cg_proto *cg_proto)
 {
- return cg_proto->flags & (1 << MEMCG_SOCK_ACTIVE);
+ return test_bit(MEMCG_SOCK_ACTIVE, &cg_proto->flags);
 }

 static inline bool memcg_proto_activated(struct cg_proto *cg_proto)
 {
- return cg_proto->flags & (1 << MEMCG_SOCK_ACTIVATED);
+ return test_bit(MEMCG_SOCK_ACTIVATED, &cg_proto->flags);
 }

 #ifdef SOCK_REFCNT_DEBUG
diff -puN  net/ipv4/tcp_memcontrol.c~memcg-decrement-static-keys-at-rea l-destroy-time-v6-fix
net/ipv4/tcp_memcontrol.c
---  a/net/ipv4/tcp_memcontrol.c~memcg-decrement-static-keys-at-r eal-destroy-time-v6-fix
+++ a/net/ipv4/tcp_memcontrol.c
@@ -108,24 +108,24 @@ static int tcp_update_limit(struct mem_c
   clear_bit(MEMCG_SOCK_ACTIVE, &cg_proto->flags);
  else if (val != RESOURCE_MAX) {
   /*
-   *  The active bit 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.
+   * The active bit 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 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.
```

```
+   * 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.
    *
-   *  The activated bit is used to guarantee that no two writers will
-   *  do the update in the same memcg. Without that, we can't properly
-   *  shutdown the static key.
+   * The activated bit is used to guarantee that no two writers
+   * will do the update in the same memcg. Without that, we can't
+   * properly shutdown the static key.
    */
   if (!test_and_set_bit(MEMCG_SOCK_ACTIVATED, &cg_proto->flags))
    static_key_slow_inc(&memcg_socket_limit_enabled);

–
```

Subject: Re: [PATCH v6 2/2] decrement static keys on real destroy time
Posted by akpm on Tue, 22 May 2012 23:11:07 GMT
View Forum Message <> Reply to Message

On Tue, 22 May 2012 15:46:10 -0700
Andrew Morton <akpm@linux-foundation.org> wrote:

> > +static inline bool memcg_proto_active(struct cg_proto *cg_proto)
> > +{
> > + return cg_proto->flags & (1 << MEMCG_SOCK_ACTIVE);
> > +}
> > +
> > +static inline bool memcg_proto_activated(struct cg_proto *cg_proto)
> > +{
> > + return cg_proto->flags & (1 << MEMCG_SOCK_ACTIVATED);
> > +}
>
> Here, we're open-coding kinda-test_bit(). Why do that? These flags are
> modified with set_bit() and friends, so we should read them with the
> matching test_bit()?
>
> Also, these bool-returning functions will return values other than 0
> and 1. That probably works OK and I don't know what the C standards
> and implementations do about this. But it seems unclean and slightly
> risky to have a "bool" value of 32! Converting these functions to use
> test_bit() fixes this - test_bit() returns only 0 or 1.
>
> test_bit() is slightly more expensive than the above. If this is
> considered to be an issue then I guess we could continue to use this
> approach. But I do think a code comment is needed, explaining and
> justifying the unusual decision to bypass the bitops API. Also these
> functions should tell the truth and return an "int" type.

Joe corrected (and informed) me:

: 6.3.1.2p1:
:
: "When any scalar value is converted to _Bool, the result is 0
: if the value compares equal to 0; otherwise, the result is 1."

So the above functions will be given compiler-generated scalar-to-boolean
conversion.

test_bit() already does internal scalar-to-boolean conversion. The
compiler doesn't know that, so if we convert the above functions to use
test_bit(), we'll end up performing scalar-to-boolean-to-boolean
conversion, which is dumb.

I assume that a way of fixing this is to change test_bit() to return
bool type. That's a bit scary.

A less scary way would be to add a new

bool test_bit_bool(int nr, const unsigned long *addr);

which internally calls test_bit() but somehow avoids the
compiler-generated conversion of the test_bit() return value into a
bool. I haven't actually thought of a way of doing this ;)

Subject: Re: [PATCH v6 2/2] decrement static keys on real destroy time
Posted by Glauber Costa on Wed, 23 May 2012 09:16:36 GMT
View Forum Message <> Reply to Message

On 05/23/2012 02:46 AM, Andrew Morton wrote:
> Here, we're open-coding kinda-test_bit(). Why do that? These flags are
> modified with set_bit() and friends, so we should read them with the
> matching test_bit()?

My reasoning was to be as cheap as possible, as you noted yourself two
paragraphs below.

> Also, these bool-returning functions will return values other than 0
> and 1. That probably works OK and I don't know what the C standards
> and implementations do about this. But it seems unclean and slightly
> risky to have a "bool" value of 32! Converting these functions to use
> test_bit() fixes this - test_bit() returns only 0 or 1.
>
> test_bit() is slightly more expensive than the above. If this is
> considered to be an issue then I guess we could continue to use this

> approach. But I do think a code comment is needed, explaining and
> justifying the unusual decision to bypass the bitops API. Also these
> functions should tell the truth and return an "int" type.
>
>> >
>> > +static void disarm_static_keys(struct mem_cgroup *memcg)
>> > +{
>> > + disarm_sock_keys(memcg);
>> > +}
> Why does this function exist? Its single caller could call
> disarm_sock_keys() directly.

It exists to make it clear that this is the point in which static keys
should be disabled. I already have a patchset that introduces other
static keys, that should, of course, also be disabled here.

I am totally fine with calling directly disarm_sock_keys(), and then in
that series wrap it in disarm_static_keys, IOW, defer its introduction,
if that's how you prefer.

>
>> >   static void drain_all_stock_async(struct mem_cgroup *memcg);
>> >
>> >   static struct mem_cgroup_per_zone *
>> > @@ -4836,6 +4854,13 @@ 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.
> This is a poor comment - it failed to tell the reader*why* that code
> must run outside the cgroup lock.

Ok, will update.

>> >   schedule_work()
>> > + * will guarantee this happens. Be careful if you need to move this
>> > + * disarm_static_keys around
> It's a bit difficult for the reader to be careful when he isn't told
> what the risks are.

Ok, will update.

>> > + */
>> > + 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..3b8fa25 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 +104,33 @@ 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)
>> >  +  clear_bit(MEMCG_SOCK_ACTIVE,&cg_proto->flags);
>> >  + else if (val != RESOURCE_MAX) {
>> >  +  /*
>> >  +   *  The active bit 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.
>> >  +   *
>> >  +   *  The activated bit is used to guarantee that no two writers will
>> >  +   *  do the update in the same memcg. Without that, we can't properly
>> >  +   *  shutdown the static key.
>> >  +   */
> This comment needlessly overflows 80 cols and has a pointless and
```

> unconventional double-space indenting. I already provided a patch
> which fixes this and a few other things, but that was ignored when you
> did the v6.

Sorry, I missed it.
>
>> > + if (!test_and_set_bit(MEMCG_SOCK_ACTIVATED,&cg_proto->flags))
>> > +   static_key_slow_inc(&memcg_socket_limit_enabled);
>> > + set_bit(MEMCG_SOCK_ACTIVE,&cg_proto->flags);
>> > + }
> So here are suggested changes from*some* of the above discussion.
> Please consider, incorporate, retest and send us a v7?

How do you want me to do it? Should I add your patch ontop of mine,
and then another one that tweaks whatever else is left, or should I just
merge those changes into the patches I have?

Subject: Re: [PATCH v6 2/2] decrement static keys on real destroy time
Posted by akpm on Wed, 23 May 2012 20:33:54 GMT
View Forum Message <> Reply to Message

On Wed, 23 May 2012 13:16:36 +0400
Glauber Costa <glommer@parallels.com> wrote:

> On 05/23/2012 02:46 AM, Andrew Morton wrote:
> > Here, we're open-coding kinda-test_bit(). Why do that? These flags are
> > modified with set_bit() and friends, so we should read them with the
> > matching test_bit()?
>
> My reasoning was to be as cheap as possible, as you noted yourself two
> paragraphs below.

These aren't on any fast path, are they?

Plus: you failed in that objective! The C compiler's internal
scalar->bool conversion makes these functions no more efficient than
test_bit().

> > So here are suggested changes from*some* of the above discussion.
> > Please consider, incorporate, retest and send us a v7?
>
> How do you want me to do it? Should I add your patch ontop of mine,
> and then another one that tweaks whatever else is left, or should I just
> merge those changes into the patches I have?

A brand new patch, I guess. I can sort out the what-did-he-change view
at this end.
