
Subject: Re: [PATCH v3 09/13] memcg: kmem accounting lifecycle management
Posted by [Michal Hocko](#) on Mon, 01 Oct 2012 12:15:53 GMT
[View Forum Message](#) <> [Reply to Message](#)

Based on the previous discussions I guess this one will get reworked, right?

On Tue 18-09-12 18:04:06, Glauber Costa wrote:

- > Because the assignment: memcg->kmem_accounted = true is done after the
- > jump labels increment, we guarantee that the root memcg will always be
- > selected until all call sites are patched (see memcg_kmem_enabled).
- > This guarantees that no mischarges are applied.
- >
- > Jump label decrement happens when the last reference count from the
- > memcg dies. This will only happen when the caches are all dead.
- >
- > -> /cgroups/memory/A/B/C
- >
- > * kmem limit set at A,
- > * A and B have no tasks,
- > * span a new task in in C.
- >
- > Because kmem_accounted is a boolean that was not set for C, no
- > accounting would be done. This is, however, not what we expect.
- >
- > The basic idea, is that when a cgroup is limited, we walk the tree
- > downwards and make sure that we store the information about the parent
- > being limited in kmem_accounted.
- >
- > We do the reverse operation when a formerly limited cgroup becomes
- > unlimited.
- >
- > Since kmem charges may outlive the cgroup existence, we need to be extra
- > careful to guarantee the memcg object will stay around for as long as
- > needed. Up to now, we were using a mem_cgroup_get()/put() pair in charge
- > and uncharge operations.
- >
- > Although this guarantees that the object will be around until the last
- > call to uncharge, this means an atomic update in every charge. We can do
- > better than that if we only issue get() in the first charge, and then
- > put() when the last charge finally goes away.
- >
- > [v3: merged all lifecycle related patches in one]
- >
- > Signed-off-by: Glauber Costa <glommer@parallels.com>
- > CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
- > CC: Christoph Lameter <cl@linux.com>
- > CC: Pekka Enberg <penberg@cs.helsinki.fi>

```

> CC: Michal Hocko <mhocko@suse.cz>
> CC: Johannes Weiner <hannes@cmpxchg.org>
> CC: Suleiman Souhlal <suleiman@google.com>
> ---
> mm/memcontrol.c | 123 ++++++-----
> 1 file changed, 112 insertions(+), 11 deletions(-)
>
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 0f36a01..720e4bb 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -287,7 +287,8 @@ struct mem_cgroup {
>  * Should the accounting and control be hierarchical, per subtree?
>  */
>  bool use_hierarchy;
> - bool kmem_accounted;
> +
> + unsigned long kmem_accounted; /* See KMEM_ACCOUNTED_*, below */
>
>  bool oom_lock;
>  atomic_t under_oom;
> @@ -340,6 +341,43 @@ struct mem_cgroup {
>  #endif
>  };
>
> +enum {
> + KMEM_ACCOUNTED_ACTIVE = 0, /* accounted by this cgroup itself */
> + KMEM_ACCOUNTED_PARENT, /* one of its parents is active */
> + KMEM_ACCOUNTED_DEAD, /* dead memcg, pending kmem charges */
> +};
> +
> +/* bits 0 and 1 */
> +#define KMEM_ACCOUNTED_MASK 0x3
> +
> +#ifdef CONFIG_MEMCG_KMEM
> +static bool memcg_kmem_set_active(struct mem_cgroup *memcg)
> +{
> + return !test_and_set_bit(KMEM_ACCOUNTED_ACTIVE, &memcg->kmem_accounted);
> +}
> +
> +static bool memcg_kmem_is_accounted(struct mem_cgroup *memcg)
> +{
> + return test_bit(KMEM_ACCOUNTED_ACTIVE, &memcg->kmem_accounted);
> +}
> +
> +static void memcg_kmem_set_active_parent(struct mem_cgroup *memcg)
> +{
> + set_bit(KMEM_ACCOUNTED_PARENT, &memcg->kmem_accounted);

```

```

> +}
> +
> +static void memcg_kmem_mark_dead(struct mem_cgroup *memcg)
> +{
> + if (test_bit(KMEM_ACCOUNTED_ACTIVE, &memcg->kmem_accounted))
> + set_bit(KMEM_ACCOUNTED_DEAD, &memcg->kmem_accounted);
> +}
> +
> +static bool memcg_kmem_dead(struct mem_cgroup *memcg)
> +{
> + return test_and_clear_bit(KMEM_ACCOUNTED_DEAD, &memcg->kmem_accounted);
> +}
> +#endif /* CONFIG_MEMCG_KMEM */
> +
> /* Stuff for move charges at task migration. */
> /*
> * Types of charges to be moved. "move_charge_at_immitgrate" is treated as a
> @@ -491,7 +529,7 @@ EXPORT_SYMBOL(tcp_proto_cgroup);
> static inline bool memcg_can_account_kmem(struct mem_cgroup *memcg)
> {
> return !mem_cgroup_disabled() && !mem_cgroup_is_root(memcg) &&
> - memcg->kmem_accounted;
> + (memcg->kmem_accounted & (KMEM_ACCOUNTED_MASK));
> }
>
> /*
> @@ -524,13 +562,9 @@ __memcg_kmem_newpage_charge(gfp_t gfp, struct mem_cgroup
**_memcg, int order)
> if (!memcg_can_account_kmem(memcg))
> return true;
>
> - mem_cgroup_get(memcg);
> -
> ret = memcg_charge_kmem(memcg, gfp, PAGE_SIZE << order) == 0;
> if (ret)
> *_memcg = memcg;
> - else
> - mem_cgroup_put(memcg);
>
> return ret;
> }
> @@ -589,7 +623,6 @@ void __memcg_kmem_uncharge_page(struct page *page, int order)
>
> WARN_ON(mem_cgroup_is_root(memcg));
> memcg_uncharge_kmem(memcg, PAGE_SIZE << order);
> - mem_cgroup_put(memcg);
> }
> #endif /* CONFIG_MEMCG_KMEM */

```

```

>
> @@ -4077,6 +4110,40 @@ static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype
*cft,
> len = scnprintf(str, sizeof(str), "%llu\n", (unsigned long long)val);
> return simple_read_from_buffer(buf, nbytes, ppos, str, len);
> }
> +
> +static void memcg_update_kmem_limit(struct mem_cgroup *memcg, u64 val)
> +{
> +#ifdef CONFIG_MEMCG_KMEM
> + struct mem_cgroup *iter;
> +
> + /*
> + * When we are doing hierarchical accounting, with an hierarchy like
> + * A/B/C, we need to start accounting kernel memory all the way up to C
> + * in case A start being accounted.
> + *
> + * So when we the cgroup first gets to be unlimited, we walk all the
> + * children of the current memcg and enable kmem accounting for them.
> + * Note that a separate bit is used there to indicate that the
> + * accounting happens due to the parent being accounted.
> + *
> + * note that memcg_kmem_set_active is a test-and-set routine, so we only
> + * arrive here once (since we never disable it)
> + */
> + mutex_lock(&set_limit_mutex);
> + if ((val != RESOURCE_MAX) && memcg_kmem_set_active(memcg)) {
> +
> + mem_cgroup_get(memcg);
> +
> + for_each_mem_cgroup_tree(iter, memcg) {
> + if (iter == memcg)
> + continue;
> + memcg_kmem_set_active_parent(iter);
> + }
> + }
> + mutex_unlock(&set_limit_mutex);
> +#endif
> +
> + /*
> + * The user of this function is...
> + * RES_LIMIT.
> @@ -4115,9 +4182,7 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
> if (ret)
> break;
>
> - /* For simplicity, we won't allow this to be disabled */

```

```

> - if (!memcg->kmem_accounted && val != RESOURCE_MAX)
> -   memcg->kmem_accounted = true;
> +   memcg_update_kmem_limit(memcg, val);
>   } else
>     return -EINVAL;
>   break;
> @@ -4791,6 +4856,20 @@ static int memcg_init_kmem(struct mem_cgroup *memcg, struct
cgroup_subsys *ss)
> static void kmem_cgroup_destroy(struct mem_cgroup *memcg)
> {
>   mem_cgroup_sockets_destroy(memcg);
> +
> + memcg_kmem_mark_dead(memcg);
> +
> + if (res_counter_read_u64(&memcg->kmem, RES_USAGE) != 0)
> +   return;
> +
> + /*
> +  * Charges already down to 0, undo mem_cgroup_get() done in the charge
> +  * path here, being careful not to race with memcg_uncharge_kmem: it is
> +  * possible that the charges went down to 0 between mark_dead and the
> +  * res_counter read, so in that case, we don't need the put
> +  */
> + if (memcg_kmem_dead(memcg))
> +   mem_cgroup_put(memcg);
> }
> #else
> static int memcg_init_kmem(struct mem_cgroup *memcg, struct cgroup_subsys *ss)
> @@ -5148,6 +5227,8 @@ mem_cgroup_create(struct cgroup *cont)
> }
>
> if (parent && parent->use_hierarchy) {
> + struct mem_cgroup __maybe_unused *p;
> +
>   res_counter_init(&memcg->res, &parent->res);
>   res_counter_init(&memcg->memsw, &parent->memsw);
>   res_counter_init(&memcg->kmem, &parent->kmem);
> @@ -5158,6 +5239,20 @@ mem_cgroup_create(struct cgroup *cont)
>   * mem_cgroup(see mem_cgroup_put).
>   */
>   mem_cgroup_get(parent);
> +#ifdef CONFIG_MEMCG_KMEM
> + /*
> +  * In case a parent is already limited when we create this, we
> +  * need him to propagate it now so we become limited as well.
> +  */
> +   mutex_lock(&set_limit_mutex);
> +   for (p = parent; p != NULL; p = parent_mem_cgroup(p)) {

```

```

> + if (memcg_kmem_is_accounted(p)) {
> +   memcg_kmem_set_active_parent(memcg);
> +   break;
> + }
> + }
> + mutex_unlock(&set_limit_mutex);
> + #endif
> } else {
>   res_counter_init(&memcg->res, NULL);
>   res_counter_init(&memcg->memsw, NULL);
> @@ -5871,9 +5966,15 @@ void memcg_uncharge_kmem(struct mem_cgroup *memcg, u64
size)
>   if (!memcg)
>     return;
>
> - res_counter_uncharge(&memcg->kmem, size);
>   res_counter_uncharge(&memcg->res, size);
>   if (do_swap_account)
>     res_counter_uncharge(&memcg->memsw, size);
> +
> + /* Not down to 0 */
> + if (res_counter_uncharge(&memcg->kmem, size))
> +   return;
> +
> + if (memcg_kmem_dead(memcg))
> +   mem_cgroup_put(memcg);
> }
> #endif /* CONFIG_MEMCG_KMEM */
> --
> 1.7.11.4
>
> --
> 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

```

--

Michal Hocko
SUSE Labs
