Subject: Re: [PATCH v4 04/14] kmem accounting basic infrastructure
Posted by Michal Hocko on Thu, 11 Oct 2012 10:11:19 GMT
View Forum Message <> Reply to Message

On Mon 08-10-12 14:06:10, Glauber Costa wrote:
> This patch adds the basic infrastructure for the accounting of the slab
> caches. To control that, the following files are created:
>
> * memory.kmem.usage_in_bytes
> * memory.kmem.limit_in_bytes
> * memory.kmem.failcnt
> * memory.kmem.max_usage_in_bytes
>
> They have the same meaning of their user memory counterparts. They
> reflect the state of the "kmem" res_counter.
>
> Per cgroup slab memory accounting is not enabled until a limit is set

s/slab/kmem/ right?

> for the group. Once the limit is set the accounting cannot be disabled
> for that group. This means that after the patch is applied, no
> behavioral changes exists for whoever is still using memcg to control
> their memory usage, until memory.kmem.limit_in_bytes is set for the
> first time.
>
> We always account to both user and kernel resource_counters. This
> effectively means that an independent kernel limit is in place when the
> limit is set to a lower value than the user memory. A equal or higher
> value means that the user limit will always hit first, meaning that kmem
> is effectively unlimited.
>
> People who want to track kernel memory but not limit it, can set this
> limit to a very high number (like RESOURCE_MAX - 1page - that no one
> will ever hit, or equal to the user memory)
>
> [v4: make kmem files part of the main array;
>   do not allow limit to be set for non-empty cgroups]
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: Michal Hocko <mhocko@suse.cz>
> CC: Johannes Weiner <hannes@cmpxchg.org>
> Acked-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> ---
> mm/memcontrol.c | 123
++-
> 1 file changed, 122 insertions(+), 1 deletion(-)
>

> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 71d259e..ba855cc 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
[...]
> @@ -332,6 +337,26 @@ struct mem_cgroup {
> #endif
> };
>
> +/* internal only representation about the status of kmem accounting. */
> +enum {
> + KMEM_ACCOUNTED_ACTIVE = 0, /* accounted by this cgroup itself */
> +};
> +
> +/* first bit */
> +#define KMEM_ACCOUNTED_MASK 0x1
> +
> +#ifdef CONFIG_MEMCG_KMEM
> +static void memcg_kmem_set_active(struct mem_cgroup *memcg)
> +{
> + 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);
> +}
> +#endif

set_active vs. is_accounted. Is there any reason for inconsistency here?

> +
> /* Stuffs for move charges at task migration. */
> /*
>   * Types of charges to be moved. "move_charge_at_immitgrate" is treated as a
[...]
> @@ -3947,6 +3980,58 @@ 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 int memcg_update_kmem_limit(struct cgroup *cont, u64 val)
> +{
> + int ret = -EINVAL;
> +#ifdef CONFIG_MEMCG_KMEM
> + struct mem_cgroup *memcg = mem_cgroup_from_cont(cont);
> + /*

> + * For simplicity, we won't allow this to be disabled. It also can't
> + * be changed if the cgroup has children already, or if tasks had
> + * already joined.
> + *
> + * If tasks join before we set the limit, a person looking at
> + * kmem.usage_in_bytes will have no way to determine when it took
> + * place, which makes the value quite meaningless.
> + *
> + * After it first became limited, changes in the value of the limit are
> + * of course permitted.
> + *
> + * Taking the cgroup_lock is really offensive, but it is so far the only
> + * way to guarantee that no children will appear. There are plenty of
> + * other offenders, and they should all go away. Fine grained locking
> + * is probably the way to go here. When we are fully hierarchical, we
> + * can also get rid of the use_hierarchy check.
> + */
> + cgroup_lock();
> + mutex_lock(&set_limit_mutex);
> + if (!memcg->kmem_accounted && val != RESOURCE_MAX) {

Just a nit but wouldn't memcg_kmem_is_accounted(memcg) be better than
directly checking kmem_accounted?
Besides that I am not sure I fully understand RESOURCE_MAX test. Say I
want to have kmem accounting for monitoring so I do
echo -1 > memory.kmem.limit_in_bytes

so you set the value but do not activate it. Isn't this just a reminder
from the time when the accounting could be deactivated?

> + if (cgroup_task_count(cont) || (memcg->use_hierarchy &&
> +   !list_empty(&cont->children))) {
> +   ret = -EBUSY;
> +   goto out;
> + }
> + ret = res_counter_set_limit(&memcg->kmem, val);

VM_BUG_IN(ret) ?
There shouldn't be any usage when you enable it or something bad is
going on.

> + if (ret)
> +   goto out;
> +
> + memcg_kmem_set_active(memcg);
> + } else
> + ret = res_counter_set_limit(&memcg->kmem, val);
> +out:

> + mutex_unlock(&set_limit_mutex);
> + cgroup_unlock();
> +#endif
> + return ret;
> +}
> +
> +static void memcg_propagate_kmem(struct mem_cgroup *memcg,
> +   struct mem_cgroup *parent)
> +{
> + memcg->kmem_accounted = parent->kmem_accounted;
> +}
> +
> /*
>   * The user of this function is...
>   * RES_LIMIT.
[...]
--
Michal Hocko
SUSE Labs