Subject: Re: [PATCH v2 06/11] memcg: kmem controller infrastructure
Posted by Michal Hocko on Tue, 14 Aug 2012 17:25:40 GMT
View Forum Message <> Reply to Message

On Thu 09-08-12 17:01:14, Glauber Costa wrote:
> This patch introduces infrastructure for tracking kernel memory pages to
> a given memcg. This will happen whenever the caller includes the flag
> __GFP_KMEMCG flag, and the task belong to a memcg other than the root.
>
> In memcontrol.h those functions are wrapped in inline accessors. The
> idea is to later on, patch those with static branches, so we don't incur
> any overhead when no mem cgroups with limited kmem are being used.
>
> [v2: improved comments and standardized function names]
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: Christoph Lameter <cl@linux.com>
> CC: Pekka Enberg <penberg@cs.helsinki.fi>
> CC: Michal Hocko <mhocko@suse.cz>
> CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> CC: Johannes Weiner <hannes@cmpxchg.org>
> ---
> include/linux/memcontrol.h | 79 ++++++++++++++++++++++
> mm/memcontrol.c | 185 ++
> 2 files changed, 264 insertions(+)
>
> diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
> index 8d9489f..75b247e 100644
> --- a/include/linux/memcontrol.h
> +++ b/include/linux/memcontrol.h
[...]
> +/**
> + * memcg_kmem_new_page: verify if a new kmem allocation is allowed.
> + * @gfp: the gfp allocation flags.
> + * @handle: a pointer to the memcg this was charged against.
> + * @order: allocation order.
> + *
> + * returns true if the memcg where the current task belongs can hold this
> + * allocation.
> + *
> + * We return true automatically if this allocation is not to be accounted to
> + * any memcg.
> + */
> +static __always_inline bool
> +memcg_kmem_new_page(gfp_t gfp, void *handle, int order)
> +{
> + if (!memcg_kmem_on)
> + return true;

> + if (!(gfp & __GFP_KMEMCG) || (gfp & __GFP_NOFAIL))

OK, I see the point behind __GFP_NOFAIL but it would deserve a comment
or a mention in the changelog.

[...]
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 54e93de..e9824c1 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
[...]
> +EXPORT_SYMBOL(__memcg_kmem_new_page);

Why is this exported?

> +
> +void __memcg_kmem_commit_page(struct page *page, void *handle, int order)
> +{
> + struct page_cgroup *pc;
> + struct mem_cgroup *memcg = handle;
> +
> + if (!memcg)
> + return;
> +
> + WARN_ON(mem_cgroup_is_root(memcg));
> + /* The page allocation must have failed. Revert */
> + if (!page) {
> + size_t size = PAGE_SIZE << order;
> +
> + memcg_uncharge_kmem(memcg, size);
> + mem_cgroup_put(memcg);
> + return;
> + }
> +
> + pc = lookup_page_cgroup(page);
> + lock_page_cgroup(pc);
> + pc->mem_cgroup = memcg;
> + SetPageCgroupUsed(pc);

Don't we need a write barrier before assigning memcg? Same as
__mem_cgroup_commit_charge. This tests the Used bit always from within
lock_page_cgroup so it should be safe but I am not 100% sure about the
rest of the code.

[...]
> +EXPORT_SYMBOL(__memcg_kmem_free_page);

Why is the symbol exported?

> #endif /* CONFIG_MEMCG_KMEM */
>
> #if defined(CONFIG_INET) && defined(CONFIG_MEMCG_KMEM)
> @@ -5759,3 +5878,69 @@ static int __init enable_swap_account(char *s)
> __setup("swapaccount=", enable_swap_account);
>
> #endif
> +
> +#ifdef CONFIG_MEMCG_KMEM
> +int memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, s64 delta)
> +{
> + struct res_counter *fail_res;
> + struct mem_cgroup *_memcg;
> + int ret;
> + bool may_oom;
> + bool nofail = false;
> +
> + may_oom = (gfp & __GFP_WAIT) && (gfp & __GFP_FS) &&
> +   !(gfp & __GFP_NORETRY);

This deserves a comment.

> +
> + ret = 0;
> +
> + if (!memcg)
> + return ret;
> +
> + _memcg = memcg;
> + ret = __mem_cgroup_try_charge(NULL, gfp, delta / PAGE_SIZE,
> +   &_memcg, may_oom);

This is really dangerous because atomic allocation which seem to be
possible could result in deadlocks because of the reclaim. Also, as I
have mentioned in the other email in this thread. Why should we reclaim
just because of kernel allocation when we are not reclaiming any of it
because shrink_slab is ignored in the memcg reclaim.

> +
> + if (ret == -EINTR) {
> + nofail = true;
> + /*
> +   * __mem_cgroup_try_charge() chosed to bypass to root due to
> +   * OOM kill or fatal signal. Since our only options are to
> +   * either fail the allocation or charge it to this cgroup, do
> +   * it as a temporary condition. But we can't fail. From a
> +   * kmem/slab perspective, the cache has already been selected,

> +   * by mem_cgroup_get_kmem_cache(), so it is too late to change
> +   * our minds
> +   */
> + res_counter_charge_nofail(&memcg->res, delta, &fail_res);
> + if (do_swap_account)
> +   res_counter_charge_nofail(&memcg->memsw, delta,
> +   &fail_res);

Hmmm, this is kind of ugly but I guess unvoidable with the current
implementation. Oh well...

> + ret = 0;
> + } else if (ret == -ENOMEM)
> + return ret;
> +
> + if (nofail)
> + res_counter_charge_nofail(&memcg->kmem, delta, &fail_res);
> + else
> + ret = res_counter_charge(&memcg->kmem, delta, &fail_res);
> +
> + if (ret) {
> + res_counter_uncharge(&memcg->res, delta);
> + if (do_swap_account)
> +   res_counter_uncharge(&memcg->memsw, delta);
> + }
> +
> + return ret;
> +}
> +
[...]

--
Michal Hocko
SUSE Labs