Subject: Re: [PATCH v4 06/14] memcg: kmem controller infrastructure
Posted by Glauber Costa on Fri, 12 Oct 2012 07:45:46 GMT
View Forum Message <> Reply to Message

On 10/11/2012 04:42 PM, Michal Hocko wrote:
> On Mon 08-10-12 14:06:12, 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 acessors. 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.
>>
>> Users of this functionality shall interact with the memcg core code
>> through the following functions:
>>
>> memcg_kmem_newpage_charge: will return true if the group can handle the
>> allocation. At this point, struct page is not
>> yet allocated.
>>
>> memcg_kmem_commit_charge: will either revert the charge, if struct page
>> allocation failed, or embed memcg information
>> into page_cgroup.
>>
>> memcg_kmem_uncharge_page: called at free time, will revert the charge.
>>
>> [v2: improved comments and standardized function names]
>> [v3: handle no longer opaque, functions not exported,
>> even more comments]
>> [v4: reworked Used bit handling and surroundings for more clarity]
>> [v5: simplified code for kmemcg compiled out and core functions in
>> memcontrol.c, moved kmem code to the middle to avoid forward decls]
>>
>> 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 | 95 ++++++++++++++++++++++++++++
>> mm/memcontrol.c | 173 ++--
>> 2 files changed, 263 insertions(+), 5 deletions(-)
>>
>
> Just a nit. Hmm we are far from being consisten in using vs. not using
> externs in header files for function declarations but I do not see any

> reason why to use them here. Names are just longer without any
> additional value.
>

Neither do I.
I don't like externs for functions, I am just using them because they
seem to be used quite extensively around...

> [...]
>> +static int memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, u64 size)
>> +{
>> + struct res_counter *fail_res;
>> + struct mem_cgroup *_memcg;
>> + int ret = 0;
>> + bool may_oom;
>> +
>> + /*
>> + * Conditions under which we can wait for the oom_killer.
>> + * __GFP_NORETRY should be masked by __mem_cgroup_try_charge,
>> + * but there is no harm in being explicit here
>> + */
>> + may_oom = (gfp & __GFP_WAIT) && !(gfp & __GFP_NORETRY);
>
> Well we _have to_ check __GFP_NORETRY here because if we don't then we
> can end up in OOM. mem_cgroup_do_charge returns CHARGE_NOMEM for
> __GFP_NORETRY (without doing any reclaim) and of oom==true we decrement
> oom retries counter and eventually hit OOM killer. So the comment is
> misleading.

I will update. What i understood from your last message is that we don't
really need to, because try_charge will do it.

>> +
>> + _memcg = memcg;
>> + ret = __mem_cgroup_try_charge(NULL, gfp, size >> PAGE_SHIFT,
>> +   &_memcg, may_oom);
>> +
>> + if (!ret) {
>> + ret = res_counter_charge(&memcg->kmem, size, &fail_res);
>
> Now that I'm thinking about the charging ordering we should charge the
> kmem first because we would like to hit kmem limit before we hit u+k
> limit, don't we.
> Say that you have kmem limit 10M and the total limit 50M. Current `u'
> would be 40M and this charge would cause kmem to hit the `k' limit. I
> think we should fail to charge kmem before we go to u+k and potentially
> reclaim/oom.
> Or has this been alredy discussed and I just do not remember?

>
This has never been discussed as far as I remember. We charged u first
since day0, and you are so far the first one to raise it...

One of the things in favor of charging 'u' first is that
mem_cgroup_try_charge is already equipped to make a lot of decisions,
like when to allow reclaim, when to bypass charges, and it would be good
if we can reuse all that.

You oom-based argument makes some sense, if all other scenarios are
unchanged by this, I can change it. I will give this some more
consideration.

>> + if (ret) {
>> +   res_counter_uncharge(&memcg->res, size);
>> +   if (do_swap_account)
>> +   res_counter_uncharge(&memcg->memsw, size);
>> + }
> [...]
>> +bool
>> +__memcg_kmem_newpage_charge(gfp_t gfp, struct mem_cgroup **_memcg, int order)
>> +{
>> + struct mem_cgroup *memcg;
>> + int ret;
>> +
>> + *_memcg = NULL;
>> + memcg = try_get_mem_cgroup_from_mm(current->mm);
>> +
>> + /*
>> + * very rare case described in mem_cgroup_from_task. Unfortunately there
>> + * isn't much we can do without complicating this too much, and it would
>> + * be gfp-dependent anyway. Just let it go
>> + */
>> + if (unlikely(!memcg))
>> + return true;
>> +
>> + if (!memcg_can_account_kmem(memcg)) {
>> + css_put(&memcg->css);
>> + return true;
>> + }
>> +
> /*
>   * Keep reference on memcg while the page is charged to prevent
>   * group from vanishing because allocation can outlive their
>   * tasks. The reference is dropped in __memcg_kmem_uncharge_page
>   */
>
> please

I can do that, but keep in mind this piece of code is going away soon =)