Subject: Re: [PATCH v3 06/13] memcg: kmem controller infrastructure
Posted by Glauber Costa on Thu, 27 Sep 2012 11:31:57 GMT
View Forum Message <> Reply to Message

On 09/26/2012 07:51 PM, Michal Hocko wrote:
> On Tue 18-09-12 18:04:03, 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.
>
> Could you describe the API a bit here, please? I guess the
> kernel user is supposed to call memcg_kmem_newpage_charge and
> memcg_kmem_commit_charge resp. memcg_kmem_uncharge_page.
> All other kmem functions here are just helpers, right?
Yes, sir.

```
>>
>> [ 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 ]
>>
>> 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 |  97 +++++++++++++++++++++++++
>>  mm/memcontrol.c            | 177 +++++++++++++++++++++++++++++++++++++++++++++++++++
>>  2 files changed, 274 insertions(+)
>>
>> diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
>> index 8d9489f..82ede9a 100644
>> --- a/include/linux/memcontrol.h
>> +++ b/include/linux/memcontrol.h
>> @@ -21,6 +21,7 @@
>>  #define _LINUX_MEMCONTROL_H
>>  #include <linux/cgroup.h>
>>  #include <linux/vm_event_item.h>
>> +#include <linux/hardirq.h>
>>
>>  struct mem_cgroup;
>>  struct page_cgroup;
>> @@ -399,6 +400,17 @@ struct sock;
>>  #ifdef CONFIG_MEMCG_KMEM
>>  void sock_update_memcg(struct sock *sk);
>>  void sock_release_memcg(struct sock *sk);
>> +
>> +static inline bool memcg_kmem_enabled(void)
>> +{
>> + return true;
>> +}
>> +
>> +extern bool __memcg_kmem_newpage_charge(gfp_t gfp, struct mem_cgroup **memcg,
>> +     int order);
>> +extern void __memcg_kmem_commit_charge(struct page *page,
>> +           struct mem_cgroup *memcg, int order);
>> +extern void __memcg_kmem_uncharge_page(struct page *page, int order);
>>  #else
>>  static inline void sock_update_memcg(struct sock *sk)
>>  {
>> @@ -406,6 +418,91 @@ static inline void sock_update_memcg(struct sock *sk)
>>  static inline void sock_release_memcg(struct sock *sk)
>>  {
>>  }
>> +
>> +static inline bool memcg_kmem_enabled(void)
>> +{
>> + return false;
>> +}
>> +
>> +static inline bool
>> +__memcg_kmem_newpage_charge(gfp_t gfp, struct mem_cgroup **memcg, int order)
>> +{
>> + return false;
>> +}
>> +
>> +static inline void  __memcg_kmem_uncharge_page(struct page *page, int order)
>> +{
>> +}
>> +
>> +static inline void
>> +__memcg_kmem_commit_charge(struct page *page, struct mem_cgroup *memcg, int order)
>> +{
>> +}
>
```

> I think we shouldn't care about these for !MEMCG_KMEM. It should be
> sufficient to define the main three functions bellow as return true
> resp. NOOP. This would reduce the code churn a bit and also make it
> better maintainable.

>

Ok.

```
>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
>> index f3fd354..0f36a01 100644
>> --- a/mm/memcontrol.c
>> +++ b/mm/memcontrol.c
>> @@ -10,6 +10,10 @@
>>   * Copyright (C) 2009 Nokia Corporation
>>   * Author: Kirill A. Shutemov
>>   *
>> + * Kernel Memory Controller
>> + * Copyright (C) 2012 Parallels Inc. and Google Inc.
>> + * Authors: Glauber Costa and Suleiman Souhlal
>> + *
>>   * This program is free software; you can redistribute it and/or modify
>>   * it under the terms of the GNU General Public License as published by
>>   * the Free Software Foundation; either version 2 of the License, or
>> @@ -426,6 +430,9 @@ struct mem_cgroup *mem_cgroup_from_css(struct
cgroup_subsys_state *s)
>>  #include <net/ip.h>
>>
>>  static bool mem_cgroup_is_root(struct mem_cgroup *memcg);
>> +static int memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, u64 size);
>> +static void memcg_uncharge_kmem(struct mem_cgroup *memcg, u64 size);
>> +
>
> Why the forward declarations here? We can simply move definitions up
> before they are used for the first time, can't we? Besides that they are
> never used/defined from outside of KMEM_MEMCG.
>
```

I see your point, given the recent patch about gcc complaining about
those things. Will change.

```
>> +
>> + *_memcg = NULL;
>> + rcu_read_lock();
>> + p = rcu_dereference(current->mm->owner);
>> + memcg = mem_cgroup_from_task(p);
>
> mem_cgroup_from_task says it can return NULL. Do we care here? If not
> then please put VM_BUG_ON(!memcg) here.
>
>> + rcu_read_unlock();
>> +
>> + if (!memcg_can_account_kmem(memcg))
>> +  return true;
>> +
>> + mem_cgroup_get(memcg);
>
> I am confused. Why do we take a reference to memcg rather than css_get
> here? Ahh it is because we keep the reference while the page is
> allocated, right? Comment please.
ok.

>
> I am still not sure whether we need css_get here as well. How do you
> know that the current is not moved in parallel and it is a last task in
> a group which then can go away?

the reference count aquired by mem_cgroup_get will still prevent the
memcg from going away, no?

>> +
>> + /* The page allocation failed. Revert */
>> + if (!page) {
>> +  memcg_uncharge_kmem(memcg, PAGE_SIZE << order);
>> +  return;
>> + }
>> +
>> + pc = lookup_page_cgroup(page);
>> + lock_page_cgroup(pc);
>> + pc->mem_cgroup = memcg;
>> + SetPageCgroupUsed(pc);
>> + unlock_page_cgroup(pc);
>> +}
>> +
>> +void __memcg_kmem_uncharge_page(struct page *page, int order)
>> +{
>> + struct mem_cgroup *memcg = NULL;
>> + struct page_cgroup *pc;
>> +
>> +
>> + pc = lookup_page_cgroup(page);
>> + /*
>> +  * Fast unlocked return. Theoretically might have changed, have to
>> +  * check again after locking.
>> +  */
>> + if (!PageCgroupUsed(pc))
>> +  return;
>> +
>> + lock_page_cgroup(pc);
>> + if (PageCgroupUsed(pc)) {
>> +  memcg = pc->mem_cgroup;
>> +  ClearPageCgroupUsed(pc);
>> + }
>> + unlock_page_cgroup(pc);
>> +
>> + /*
>> +  * Checking if kmem accounted is enabled won't work for uncharge, since
>> +  * it is possible that the user enabled kmem tracking, allocated, and
>> +  * then disabled it again.
>
> disabling cannot happen, right?
>
```

not anymore, right. I can update the comment, but I still believe it is
a lot saner to trust information in page_cgroup.

```
>> +#ifdef CONFIG_MEMCG_KMEM
>> +int memcg_charge_kmem(struct mem_cgroup *memcg, gfp_t gfp, u64 size)
>> +{
>> + 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);
>
> A comment please? Why __GFP_IO is not considered for example?
>
>
```

Actually, I believe testing for GFP_WAIT and !GFP_NORETRY would be enough.

The rationale here is, of course, under which circumstance would it be
valid to call the oom killer? Which is, if the allocation can wait, and
can retry.

```
>
> You could save few lines and get rid of the strange nofail by:
> [...]
> +  res_counter_charge_nofail(&memcg->res, size, &fail_res);
> +  if (do_swap_account)
> +   res_counter_charge_nofail(&memcg->memsw, size,
> +       &fail_res);
> +  res_counter_charge_nofail(&memcg->kmem, size, &fail_res);
> +  return 0;
> + } else if (ret == -ENOMEM)
> +  return ret;
> + else
> +  ret = res_counter_charge(&memcg->kmem, size, &fail_res);
> +
> + if (ret) {
> +  res_counter_uncharge(&memcg->res, size);
> +  if (do_swap_account)
> +   res_counter_uncharge(&memcg->memsw, size);
> + }
>
```
indeed.