Subject: Re: [PATCH v2 06/11] memcg: kmem controller infrastructure
Posted by Glauber Costa on Mon, 13 Aug 2012 08:07:46 GMT
View Forum Message <> Reply to Message

On 08/11/2012 09:11 AM, Greg Thelen wrote:
> On Thu, Aug 09 2012, 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
>> @@ -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 +400,11 @@ struct sock;
>> #ifdef CONFIG_MEMCG_KMEM
>> void sock_update_memcg(struct sock *sk);
>> void sock_release_memcg(struct sock *sk);
>> +
>> +#define memcg_kmem_on 1
>> +bool __memcg_kmem_new_page(gfp_t gfp, void *handle, int order);
>> +void __memcg_kmem_commit_page(struct page *page, void *handle, int order);

>> +void __memcg_kmem_free_page(struct page *page, int order);
>> #else
>> static inline void sock_update_memcg(struct sock *sk)
>> {
>> @@ -406,6 +412,79 @@ static inline void sock_update_memcg(struct sock *sk)
>> static inline void sock_release_memcg(struct sock *sk)
>> {
>> }
>> +
>> +#define memcg_kmem_on 0
>> +static inline bool
>> +__memcg_kmem_new_page(gfp_t gfp, void *handle, int order)
>> +{
>> + return false;
>> +}
>> +
>> +static inline void __memcg_kmem_free_page(struct page *page, int order)
>> +{
>> +}
>> +
>> +static inline void
>> +__memcg_kmem_commit_page(struct page *page, struct mem_cgroup *handle, int order)
>> +{
>> +}
>> #endif /* CONFIG_MEMCG_KMEM */
>> +
>> +/**
>> + * 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))
>> + return true;
>> + if (in_interrupt() || (!current->mm) || (current->flags & PF_KTHREAD))
>> + return true;
>> + return __memcg_kmem_new_page(gfp, handle, order);

```
>> +}
>> +
>> +/**
>> + * memcg_kmem_free_page: uncharge pages from memcg
>> + * @page: pointer to struct page being freed
>> + * @order: allocation order.
>> + *
>> + * there is no need to specify memcg here, since it is embedded in page_cgroup
>> + */
>> +static __always_inline void
>> +memcg_kmem_free_page(struct page *page, int order)
>> +{
>> + if (memcg_kmem_on)
>> +  __memcg_kmem_free_page(page, order);
>> +}
>> +
>> +/**
>> + * memcg_kmem_commit_page: embeds correct memcg in a page
>> + * @handle: a pointer to the memcg this was charged against.
>> + * @page: pointer to struct page recently allocated
>> + * @handle: the memcg structure we charged against
>> + * @order: allocation order.
>> + *
>> + * Needs to be called after memcg_kmem_new_page, regardless of success or
>> + * failure of the allocation. if @page is NULL, this function will revert the
>> + * charges. Otherwise, it will commit the memcg given by @handle to the
>> + * corresponding page_cgroup.
>> + */
>> +static __always_inline void
>> +memcg_kmem_commit_page(struct page *page, struct mem_cgroup *handle, int order)
>> +{
>> + if (memcg_kmem_on)
>> +  __memcg_kmem_commit_page(page, handle, order);
>> +}
>>  #endif /* _LINUX_MEMCONTROL_H */
>>
>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
>> index 54e93de..e9824c1 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
>> @@ -434,6 +438,9 @@ 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, s64 delta);
>> +static void memcg_uncharge_kmem(struct mem_cgroup *memcg, s64 delta);
>> +
>>  void sock_update_memcg(struct sock *sk)
>>  {
>>   if (mem_cgroup_sockets_enabled) {
>> @@ -488,6 +495,118 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
>>  }
>>  EXPORT_SYMBOL(tcp_proto_cgroup);
>>  #endif /* CONFIG_INET */
>> +
>> +static inline bool memcg_kmem_enabled(struct mem_cgroup *memcg)
>> +{
>> + return !mem_cgroup_disabled() && !mem_cgroup_is_root(memcg) &&
>> +  memcg->kmem_accounted;
>> +}
>> +
>> +/*
>> + * We need to verify if the allocation against current->mm->owner's memcg is
>> + * possible for the given order. But the page is not allocated yet, so we'll
>> + * need a further commit step to do the final arrangements.
>> + *
>> + * It is possible for the task to switch cgroups in this mean time, so at
>> + * commit time, we can't rely on task conversion any longer.  We'll then use
>> + * the handle argument to return to the caller which cgroup we should commit
>> + * against
>> + *
>> + * Returning true means the allocation is possible.
>> + */
>> +bool __memcg_kmem_new_page(gfp_t gfp, void *_handle, int order)
>> +{
>> + struct mem_cgroup *memcg;
>> + struct mem_cgroup **handle = (struct mem_cgroup **)_handle;
>> + bool ret = true;
>> + size_t size;
>> + struct task_struct *p;
>> +
>> + *handle = NULL;
>> + rcu_read_lock();
>> + p = rcu_dereference(current->mm->owner);
```

```
>> + memcg = mem_cgroup_from_task(p);
>> + if (!memcg_kmem_enabled(memcg))
>> +  goto out;
>> +
>> + mem_cgroup_get(memcg);
>> +
>> + size = PAGE_SIZE << order;
>> + ret = memcg_charge_kmem(memcg, gfp, size) == 0;
>> + if (!ret) {
>> +  mem_cgroup_put(memcg);
>> +  goto out;
>> + }
>> +
>> + *handle = memcg;
>> +out:
>> + rcu_read_unlock();
>> + return ret;
>> +}
>> +EXPORT_SYMBOL(__memcg_kmem_new_page);
>
> While running f853d89 from git://github.com/glommer/linux.git , I hit a
> lockdep issue.  To create this I allocated and held reference to some
> kmem in the context of a kmem limited memcg.  Then I moved the
> allocating process out of memcg and then deleted the memcg.  Due to the
> kmem reference the struct mem_cgroup is still active but invisible in
> cgroupfs namespace.  No problems yet.  Then I killed the user process
> which freed the kmem from the now unlinked memcg.  Dropping the kmem
> caused the memcg ref to hit zero.  Then the memcg is deleted but that
> acquires a non-irqsafe spinlock in softirq which annoys lockdep.  I
> think the lock in question is the mctz below:
>
>     mem_cgroup_remove_exceeded(struct mem_cgroup *memcg,
>         struct mem_cgroup_per_zone *mz,
>         struct mem_cgroup_tree_per_zone *mctz)
>     {
>      spin_lock(&mctz->lock);
>      __mem_cgroup_remove_exceeded(memcg, mz, mctz);
>      spin_unlock(&mctz->lock);
>     }
>
> Perhaps your patches expose this problem by being the first time we call
> __mem_cgroup_free() from softirq (this is just an educated guess).  I'm
> not sure how this would interact with Ying's soft limit rework:
> https://lwn.net/Articles/501338/
>
```

Thanks for letting me know, Greg,

I'll try to reproduce this today and see how it goes.