
Subject: Re: [PATCH 11/23] slub: consider a memcg parameter in
kmem_create_cache

Posted by [Glauber Costa](#) on Wed, 02 May 2012 15:18:04 GMT

[View Forum Message](#) <> [Reply to Message](#)

On 04/30/2012 04:51 PM, Suleiman Souhlal wrote:

> On Fri, Apr 20, 2012 at 2:57 PM, Glauber Costa<glommer@parallels.com> wrote:

>> Allow a memcg parameter to be passed during cache creation.

>> The slub allocator will only merge caches that belong to

>> the same memcg.

>>

>> Default function is created as a wrapper, passing NULL

>> to the memcg version. We only merge caches that belong

>> to the same memcg.

>>

>> From the memcontrol.c side, 3 helper functions are created:

>>

>> 1) memcg_css_id: because slub needs a unique cache name

>> for sysfs. Since this is visible, but not the canonical

>> location for slab data, the cache name is not used, the

>> css_id should suffice.

>>

>> 2) mem_cgroup_register_cache: is responsible for assigning

>> a unique index to each cache, and other general purpose

>> setup. The index is only assigned for the root caches. All

>> others are assigned index == -1.

>>

>> 3) mem_cgroup_release_cache: can be called from the root cache

>> destruction, and will release the index for other caches.

>>

>> This index mechanism was developed by Suleiman Souhlal.

>>

>> 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>

>> CC: Suleiman Souhlal<suleiman@google.com>

>> ---

>> include/linux/memcontrol.h | 14 ++++++

>> include/linux/slab.h | 6 +++++

>> mm/memcontrol.c | 29 ++++++

>> mm/slub.c | 31 ++++++

>> 4 files changed, 76 insertions(+), 4 deletions(-)

>>

>> diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h

>> index f94efd2..99e14b9 100644

```

>> --- a/include/linux/memcontrol.h
>> +++ b/include/linux/memcontrol.h
>> @@ -26,6 +26,7 @@ struct mem_cgroup;
>> struct page_cgroup;
>> struct page;
>> struct mm_struct;
>> +struct kmem_cache;
>>
>> /* Stats that can be updated by kernel. */
>> enum mem_cgroup_page_stat_item {
>> @@ -440,7 +441,20 @@ struct sock;
>> #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>> void sock_update_memcg(struct sock *sk);
>> void sock_release_memcg(struct sock *sk);
>> +int memcg_css_id(struct mem_cgroup *memcg);
>> +void mem_cgroup_register_cache(struct mem_cgroup *memcg,
>> +                               struct kmem_cache *s);
>> +void mem_cgroup_release_cache(struct kmem_cache *cachep);
>> #else
>> +static inline void mem_cgroup_register_cache(struct mem_cgroup *memcg,
>> +                                             struct kmem_cache *s)
>> +{
>> +}
>> +
>> +static inline void mem_cgroup_release_cache(struct kmem_cache *cachep)
>> +{
>> +}
>> +
>> static inline void sock_update_memcg(struct sock *sk)
>> {
>> }
>> diff --git a/include/linux/slab.h b/include/linux/slab.h
>> index a5127e1..c7a7e05 100644
>> --- a/include/linux/slab.h
>> +++ b/include/linux/slab.h
>> @@ -321,6 +321,12 @@ extern void *__kmalloc_track_caller(size_t, gfp_t, unsigned long);
>> __kmalloc(size, flags)
>> #endif /* DEBUG_SLAB */
>>
>> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>> +#define MAX_KMEM_CACHE_TYPES 400
>> +#else
>> +#define MAX_KMEM_CACHE_TYPES 0
>> +#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
>> +
>> #ifdef CONFIG_NUMA
>> /*
>> * kmalloc_node_track_caller is a special version of kmalloc_node that

```

```

>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
>> index 36f1e6b..0015ed0 100644
>> --- a/mm/memcontrol.c
>> +++ b/mm/memcontrol.c
>> @@ -323,6 +323,11 @@ struct mem_cgroup {
>> #endif
>> };
>>
>> +int memcg_css_id(struct mem_cgroup *memcg)
>> +{
>> +    return css_id(&memcg->css);
>> +}
>> +
>> /* Stuffs for move charges at task migration. */
>> /*
>> * Types of charges to be moved. "move_charge_at_immitgrate" is treated as a
>> @@ -461,6 +466,30 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
>> }
>> EXPORT_SYMBOL(tcp_proto_cgroup);
>> #endif /* CONFIG_INET */
>> +
>> +/* Bitmap used for allocating the cache id numbers. */
>> +static DECLARE_BITMAP(cache_types, MAX_KMEM_CACHE_TYPES);
>> +
>> +void mem_cgroup_register_cache(struct mem_cgroup *memcg,
>> +                               struct kmem_cache *cachep)
>> +{
>> +    int id = -1;
>> +
>> +    cachep->memcg_params.memcg = memcg;
>> +
>> +    if (!memcg) {
>> +        id = find_first_zero_bit(cache_types, MAX_KMEM_CACHE_TYPES);
>> +        BUG_ON(id < 0 || id >= MAX_KMEM_CACHE_TYPES);
>> +        __set_bit(id, cache_types);
>> +    } else
>> +        INIT_LIST_HEAD(&cachep->memcg_params.destroyed_list);
>> +    cachep->memcg_params.id = id;
>> +}
>> +
>> +void mem_cgroup_release_cache(struct kmem_cache *cachep)
>> +{
>> +    __clear_bit(cachep->memcg_params.id, cache_types);
>> +}
>> #endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
>>
>> static void drain_all_stock_async(struct mem_cgroup *memcg);
>> diff --git a/mm/slub.c b/mm/slub.c

```

```

>> index 2652e7c..86e40cc 100644
>> --- a/mm/slub.c
>> +++ b/mm/slub.c
>> @@ -32,6 +32,7 @@
>> #include<linux/prefetch.h>
>>
>> #include<trace/events/kmem.h>
>> +#include<linux/memcontrol.h>
>>
>> /*
>>  * Lock order:
>> @@ -3880,7 +3881,7 @@ static int slab_unmergeable(struct kmem_cache *s)
>>     return 0;
>> }
>>
>> -static struct kmem_cache *find_mergeable(size_t size,
>> +static struct kmem_cache *find_mergeable(struct mem_cgroup *memcg, size_t size,
>>     size_t align, unsigned long flags, const char *name,
>>     void (*ctor)(void *))
>> {
>> @@ -3916,21 +3917,29 @@ static struct kmem_cache *find_mergeable(size_t size,
>>     if (s->size - size >= sizeof(void *))
>>         continue;
>>
>> +     if (memcg && s->memcg_params.memcg != memcg)
>> +         continue;
>> +
>>     return s;
>> }
>> return NULL;
>> }
>>
>> -struct kmem_cache *kmem_cache_create(const char *name, size_t size,
>> -     size_t align, unsigned long flags, void (*ctor)(void *))
>> +struct kmem_cache *
>> +kmem_cache_create_memcg(struct mem_cgroup *memcg, const char *name, size_t size,
>> +     size_t align, unsigned long flags, void (*ctor)(void *))
>> {
>>     struct kmem_cache *s;
>>
>>     if (WARN_ON(!name))
>>         return NULL;
>>
>> +#ifndef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>> +     WARN_ON(memcg != NULL);
>> +#endif
>> +
>>     down_write(&slub_lock);

```

```

>> -    s = find_mergeable(size, align, flags, name, ctor);
>> +    s = find_mergeable(memcg, size, align, flags, name, ctor);
>>     if (s) {
>>         s->refcount++;
>>         /*
>> @@ -3954,12 +3963,15 @@ struct kmem_cache *kmem_cache_create(const char *name,
size_t size,
>>         size, align, flags, ctor)) {
>>         list_add(&s->list, &slab_caches);
>>         up_write(&slub_lock);
>> +         mem_cgroup_register_cache(memcg, s);
>
> Do the kmalloc caches get their id registered correctly?
>

```

For the slub, it seems to work okay. But I had to use the trick that for the memcg-specific kmalloc caches, they come from the normal caches rather than the special kmalloc pool. Since we are already paying the penalty of dealing with the memcg finding, I hope this is okay.

For the slab, my investigation wasn't that deep. But basic functionality works okay.