Subject: Re: [PATCH 11/23] slub: consider a memcg parameter in kmem_create_cache
Posted by Glauber Costa on Wed, 25 Apr 2012 14:37:22 GMT
View Forum Message <> Reply to Message

On 04/24/2012 10:38 PM, KAMEZAWA Hiroyuki wrote:
> (2012/04/21 6:57), Glauber Costa 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 */
>> +
>
>

> why 400 ?

Quite arbitrary. Just large enough to hold all caches there are
currently in a system + modules. (Right now I have around 140
in a normal fedora installation)

>> +/* 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);
>
>
> No lock here ? you need find_first_zero_bit_and_set_atomic() or some.
> Rather than that, I think you can use lib/idr.c::ida_simple_get().

This function is called from within kmem_cache_create(), that usually
already do locking. The slub, for instance, uses the slub_lock() for all
cache creation, and the slab do something quite similar. (All right, I
should have mentioned that in comments)

But as for idr, I don't think it is a bad idea. I will take a look.

>> @@ -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
>
>
> I'm sorry what's is this warning for ?
this is inside ifndef (not defined), so this means anyone trying to pass
a memcg in that situation, is doing something really wrong.

I was actually going for BUG() on this one, but changed my mind

Thinking again, I could probably do this:

if (WARN_ON(memcg != NULL))
  memcg = NULL;

this way we can keep going without killing the kernel as well as
protecting the function.

>
>> @@ -5265,6 +5283,11 @@ static char *create_unique_id(struct kmem_cache *s)
>>   if (p != name + 1)
>>   *p++ = '-';
>>   p += sprintf(p, "%07d", s->size);
>> +
>> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>> + if (s->memcg_params.memcg)
>> + p += sprintf(p, "-%08d", memcg_css_id(s->memcg_params.memcg));
>> +#endif
>>   BUG_ON(p> name + ID_STR_LENGTH - 1);
>>   return name;
>>   }

>
>
> So, you use 'id' in user interface. Should we provide 'id' as memory.id file ?

We could.
But that is not the cache name, this is for alias files.

The cache name has css_id:dcache_name, so we'll see something like
2:container1

The css_id plays the role of avoiding name duplicates, since all we use
is the last dentry to derive the name.

So I guess if need arises to go search in sysfs for the slub stuff, it
gets easy enough to correlate so we don't need to export the id.