Subject: Re: [PATCH v5 06/18] consider a memcg parameter in kmem_create_cache
Posted by Glauber Costa on Wed, 24 Oct 2012 08:42:09 GMT
View Forum Message <> Reply to Message

On 10/23/2012 09:50 PM, JoonSoo Kim wrote:
> 2012/10/19 Glauber Costa <glommer@parallels.com>:
>
>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
>> index 6a1e096..59f6d54 100644
>> --- a/mm/memcontrol.c
>> +++ b/mm/memcontrol.c
>> @@ -339,6 +339,12 @@ struct mem_cgroup {
>> #if defined(CONFIG_MEMCG_KMEM) && defined(CONFIG_INET)
>> struct tcp_memcontrol tcp_mem;
>> #endif
>> +#if defined(CONFIG_MEMCG_KMEM)
>> + /* analogous to slab_common's slab_caches list. per-memcg */
>> + struct list_head memcg_slab_caches;
>> + /* Not a spinlock, we can take a lot of time walking the list */
>> + struct mutex slab_caches_mutex;
>> +#endif
>> };
>
> It is better to change name of "slab_caches_mutex to something
> representing slab cache mutex of memcg.
>

Why do you think so? I particularly think keeping the same name is
better, because it shows us that this is a memcg version of the global
mutex, and wraps all caches. One is unlikely to confuse between them,
because one of them is a global name, and the other is always inside a
struct.

>> +int memcg_register_cache(struct mem_cgroup *memcg, struct kmem_cache *s)
>> +{
>> + size_t size = sizeof(struct memcg_cache_params);
>> +
>> + if (!memcg_kmem_enabled())
>> +   return 0;
>> +
>> + s->memcg_params = kzalloc(size, GFP_KERNEL);
>> + if (!s->memcg_params)
>> +   return -ENOMEM;
>> +
>> + if (memcg)
>> +   s->memcg_params->memcg = memcg;
>> + return 0;

>> +}
>
> Now, I don't read full-patchset and I have not enough knowledge about cgroup.
> So I have a question.
> When memcg_kmem_enable, creation kmem_cache of normal user(not
> included in cgroup) also allocate memcg_params.
> Is it right behavior?
>
Yes. I would even add that *specially* the normal kmem cache gets this.
This is where it will hold the vector of memcg caches.

Also note this only happens when a memcg becomes kmem-limited, so people
not using memcg at all won't pay this price.

This should be documented over struct memcg_params.

>> +static inline bool cache_match_memcg(struct kmem_cache *cachep,
>> +   struct mem_cgroup *memcg)
>> +{
>> +   return (is_root_cache(cachep) && !memcg) ||
>> +   (cachep->memcg_params->memcg == memcg);
>> +}
>
> When cachep->memcg_params == NULL and memcg is not NULL, NULL pointer
> deref may be occurred.
> Is there no situation like above?
>

is_root_cache already test for this, so if this is the case, we'll exit
the conditional on that test.

Every cache without a valid memcg_params pointer is a root cache.

If memcg != NULL, and memcg_params == NULL, this is a serious bug.
I just didn't BUG(), because I really didn't see the point. It is just
added instructions, and we're unlikely to make that far in that case.

>>   dump_stack();
>> @@ -66,7 +68,8 @@ static int kmem_cache_sanity_check(const char *name, size_t size)
>>   return 0;
>> }
>> #else
>> -static inline int kmem_cache_sanity_check(const char *name, size_t size)
>> +static inline int kmem_cache_sanity_check(struct mem_cgroup *memcg,
>> +   const char *name, size_t size)
>> {
>>   return 0;

>> }
>> @@ -97,8 +100,9 @@ static inline int kmem_cache_sanity_check(const char *name, size_t
size)
>> * as davem.
>> */
>>
>> -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 = NULL;
>> int err = 0;
>> @@ -106,7 +110,7 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t
size, size_t align
>> get_online_cpus();
>> mutex_lock(&slab_mutex);
>>
>> - if (!kmem_cache_sanity_check(name, size) == 0)
>> + if (!kmem_cache_sanity_check(memcg, name, size) == 0)
>> goto out_locked;
>
> This compare is somewhat ugly.
> How about "if(kmem_cache_sanity_check(memcg, name, size))"?
>

Well, the check was already there =)
At least it becomes clear that the only thing I am changing is that i
know pass a memcg pointer.

As for the proposed change, it gives the impression that if you succeed
in the check, you will exit the function.

At least this way, it becomes clear that you exit when you fail a
sanity_check. So taking the semantics into consideration, I don't really
dislike it.

>> {
>> @@ -3916,17 +3917,20 @@ static struct kmem_cache *find_mergeable(size_t size,
>> if (s->size - size >= sizeof(void *))
>> continue;
>>
>> + if (!cache_match_memcg(s, memcg))
>> + continue;
>> return s;
>> }
>> return NULL;

```
>>  }
>>
>> -struct kmem_cache *__kmem_cache_alias(const char *name, size_t size,
>> -               size_t align, unsigned long flags, void (*ctor)(void *))
>> +struct kmem_cache *
>> +__kmem_cache_alias(struct mem_cgroup *memcg, const char *name, size_t size,
>> +                   size_t align, unsigned long flags, void (*ctor)(void *))
>>  {
>>         struct kmem_cache *s;
>>
>> -       s = find_mergeable(size, align, flags, name, ctor);
>> +       s = find_mergeable(memcg, size, align, flags, name, ctor);
>>         if (s) {
>>                 s->refcount++;
>>                 /*
>
> If your intention is that find_mergeable() works for memcg-slab-caches properly,
> it cannot works properly with this code.
> When memcg is not NULL, slab cache is only added to memcg's slab cache list.
> find_mergeable() only interate on original-slab-cache list.
> So memcg slab cache never be mergeable.
>
```

Damn, you're right. In earlier submissions, we had a single list =(
I'll have to adapt this code, thanks for spotting this!