

---

Subject: Re: [PATCH v5 08/18] memcg: infrastructure to match an allocation to the right cache

Posted by [JoonSoo Kim](#) on Wed, 24 Oct 2012 18:10:29 GMT

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

---

2012/10/19 Glauber Costa <glommer@parallels.com>:

```
> @@ -2930,9 +2937,188 @@ int memcg_register_cache(struct mem_cgroup *memcg, struct
kmem_cache *s)
```

```
>
> void memcg_release_cache(struct kmem_cache *s)
> {
> + struct kmem_cache *root;
> + int id = memcg_css_id(s->memcg_params->memcg);
> +
> + if (s->memcg_params->is_root_cache)
> +     goto out;
> +
> + root = s->memcg_params->root_cache;
> + root->memcg_params->memcg_caches[id] = NULL;
> + mem_cgroup_put(s->memcg_params->memcg);
> +out:
>     kfree(s->memcg_params);
> }
```

memcg\_css\_id should be called after checking "s->memcg\_params->is\_root\_cache".  
Because when is\_root\_cache == true, memcg\_params has no memcg object.

```
> +/*
> + * This lock protects updaters, not readers. We want readers to be as fast as
> + * they can, and they will either see NULL or a valid cache value. Our model
> + * allow them to see NULL, in which case the root memcg will be selected.
> + *
> + * We need this lock because multiple allocations to the same cache from a non
> + * GFP_WAIT area will span more than one worker. Only one of them can create
> + * the cache.
> + */
> +static DEFINE_MUTEX(memcg_cache_mutex);
> +static struct kmem_cache *memcg_create_kmem_cache(struct mem_cgroup *memcg,
> + struct kmem_cache *cachep)
> +{
> + struct kmem_cache *new_cachep;
> + int idx;
> +
> + BUG_ON(!memcg_can_account_kmem(memcg));
> +
> + idx = memcg_css_id(memcg);
> +
```

```

> + mutex_lock(&memcg_cache_mutex);
> + new_cachep = cachep->memcg_params->memcg_caches[idx];
> + if (new_cachep)
> +     goto out;
> +
> + new_cachep = kmem_cache_dup(memcg, cachep);
> +
> + if (new_cachep == NULL) {
> +     new_cachep = cachep;
> +     goto out;
> + }
> +
> + mem_cgroup_get(memcg);
> + cachep->memcg_params->memcg_caches[idx] = new_cachep;
> + wmb(); /* the readers won't lock, make sure everybody sees it */

```

Is there any rmb() pair?

As far as I know, without rmb(), wmb() doesn't guarantee anything.

```

> + new_cachep->memcg_params->memcg = memcg;
> + new_cachep->memcg_params->root_cache = cachep;

```

It may be better these assignment before the statement  
"cachep->memcg\_params->memcg\_caches[idx] = new\_cachep".

Otherwise, it may produce race situation.

And assigning value to memcg\_params->memcg and root\_cache is redundant,  
because it is already done in memcg\_register\_cache().

```

> +/*
> + * Return the kmem_cache we're supposed to use for a slab allocation.
> + * We try to use the current memcg's version of the cache.
> + *
> + * If the cache does not exist yet, if we are the first user of it,
> + * we either create it immediately, if possible, or create it asynchronously
> + * in a workqueue.
> + * In the latter case, we will let the current allocation go through with
> + * the original cache.
> + *
> + * Can't be called in interrupt context or from kernel threads.
> + * This function needs to be called with rcu_read_lock() held.
> + */
> +struct kmem_cache * __memcg_kmem_get_cache(struct kmem_cache *cachep,
> +                                           gfp_t gfp)
> +{
> +    struct mem_cgroup *memcg;
> +    int idx;
> +

```

```
> +   if (cachep->memcg_params && cachep->memcg_params->memcg)
> +       return cachep;
```

In `__memcg_kmem_get_cache`, `cachep` may be always root cache.  
So checking "`cachep->memcg_params->memcg`" is somewhat strange.  
Is it right?

Thanks.

---