Subject: Re: [PATCH v3 13/28] slub: create duplicate cache
Posted by Glauber Costa on Tue, 29 May 2012 15:56:23 GMT
View Forum Message <> Reply to Message

On 05/29/2012 06:36 PM, Christoph Lameter wrote:
```
> On Fri, 25 May 2012, Glauber Costa wrote:
>
>> index dacd1fb..4689034 100644
>> --- a/mm/memcontrol.c
>> +++ b/mm/memcontrol.c
>> @@ -467,6 +467,23 @@ struct cg_proto *tcp_proto_cgroup(struct mem_cgroup *memcg)
>>   EXPORT_SYMBOL(tcp_proto_cgroup);
>>   #endif /* CONFIG_INET */
>>
>> +char *mem_cgroup_cache_name(struct mem_cgroup *memcg, struct kmem_cache *cachep)
>> +{
>> + char *name;
>> + struct dentry *dentry;
>> +
>> + rcu_read_lock();
>> + dentry = rcu_dereference(memcg->css.cgroup->dentry);
>> + rcu_read_unlock();
>> +
>> + BUG_ON(dentry == NULL);
>> +
>> + name = kasprintf(GFP_KERNEL, "%s(%d:%s)",
>> +     cachep->name, css_id(&memcg->css), dentry->d_name.name);
>> +
>> + return name;
>> +}
>
> Function allocates a string that is supposed to be disposed of by the
> caller. That needs to be documented and maybe even the name needs to
> reflect that.
```

Okay, I can change it.

```
>> --- a/mm/slub.c
>> +++ b/mm/slub.c
>> @@ -4002,6 +4002,38 @@ struct kmem_cache *kmem_cache_create(const char *name,
size_t size,
>>   }
>>   EXPORT_SYMBOL(kmem_cache_create);
>>
>> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>> +struct kmem_cache *kmem_cache_dup(struct mem_cgroup *memcg,
>> +      struct kmem_cache *s)
>> +{
```

```
>> + char *name;
>> + struct kmem_cache *new;
>> +
>> + name = mem_cgroup_cache_name(memcg, s);
>> + if (!name)
>> +  return NULL;
>> +
>> + new = kmem_cache_create_memcg(memcg, name, s->objsize, s->align,
>> +         (s->allocflags&  ~SLAB_PANIC), s->ctor);
>
```

> Hmmm... A full duplicate of the slab cache? We may have many sparsely
> used portions of the per node and per cpu structure as a result.

I've already commented on patch 0, but I will repeat it here. This approach leads to more fragmentation, yes, but this is exactly to be less intrusive.

With a full copy, all I need to do is:

1) relay the allocation to the right cache.
2) account for a new page when it is needed.

How does the cache work from inside? I don't care.

Accounting pages seems just crazy to me. If new allocators come in the future, organizing the pages in a different way, instead of patching it here and there, we need to totally rewrite this.

If those allocators happen to depend on a specific placement for performance, then we're destroying this as well too.

```
>
>> +  * prevent it from being deleted. If kmem_cache_destroy() is
>> +  * called for the root cache before we call it for a child cache,
>> +  * it will be queued for destruction when we finally drop the
>> +  * reference on the child cache.
>> +  */
>> + if (new) {
>> +  down_write(&slub_lock);
>> +  s->refcount++;
>> +  up_write(&slub_lock);
>> + }
>
```

> Why do you need to increase the refcount? You made a full copy right?

Yes, but I don't want this copy to go away while we have other caches around.

So, in the memcg internals, I used a different reference counter, to avoid messing with this one. I could use that, and leave the original refcnt alone. Would you prefer this?