Subject: Re: [PATCH 05/10] memcg: Slab accounting.
Posted by Glauber Costa on Wed, 29 Feb 2012 17:00:33 GMT
View Forum Message <> Reply to Message

On 02/28/2012 08:31 PM, Suleiman Souhlal wrote:
> On Tue, Feb 28, 2012 at 5:24 AM, Glauber Costa<glommer@parallels.com> wrote:
>> On 02/27/2012 07:58 PM, Suleiman Souhlal wrote:
>>>
>>> Introduce per-cgroup kmem_caches for memcg slab accounting, that
>>> get created the first time we do an allocation of that type in the
>>> cgroup.
>>> If we are not permitted to sleep in that allocation, the cache
>>> gets created asynchronously.
>>
>> And then we allocate from the root cgroup?
>
> Yes, the allocation will go to the root cgroup (or not get accounted
> at all if you don't have CONFIG_CGROUP_MEM_RES_CTLR_KMEM_ACCT_ROOT).
> Once the workqueue runs and creates the memcg cache, all the
> allocations of that type will start using it.
>
>>> The cgroup cache gets used in subsequent allocations, and permits
>>> accounting of slab on a per-page basis.
>>>
>>> The per-cgroup kmem_caches get looked up at slab allocation time,
>>> in a MAX_KMEM_CACHE_TYPES-sized array in the memcg structure, based
>>> on the original kmem_cache's id, which gets allocated when the original
>>> cache gets created.
>>>
>>> Allocations that cannot be attributed to a cgroup get charged to
>>> the root cgroup.
>>>
>>> Each cgroup kmem_cache has a refcount that dictates the lifetime
>>> of the cache: We destroy a cgroup cache when its cgroup has been
>>> destroyed and there are no more active objects in the cache.
>>
>>
>> Since we already track the number of pages in the slab, why do we need a
>> refcnt?
>
> I must be missing something, but I don't see a counter of the number
> of active pages in the cache in the code. :-(

That's fine. Remember I did my work on the slub, so the internals of the
slab are to me fuzzy at the very best.

If you make sure that this refcnt never leaves slab.c, it should be
okay. (since for the slub, it definitely is not needed - we do track the

# of pages).

One thing we could do is both provide "slab_nr_pages()" (or a better
more generic name) as I did in my series, and then you draw this value
from the refcnt, and the slub from a different location.

>>> diff --git a/include/linux/slab.h b/include/linux/slab.h
>>> index 573c809..fe21a91 100644
>>> --- a/include/linux/slab.h
>>> +++ b/include/linux/slab.h
>>> @@ -21,6 +21,7 @@
>>> #define SLAB_POISON 0x00000800UL /* DEBUG: Poison objects */
>>> #define SLAB_HWCACHE_ALIGN 0x00002000UL /* Align objs on cache
>>> lines */
>>> #define SLAB_CACHE_DMA 0x00004000UL /* Use GFP_DMA
>>> memory */
>>> +#define SLAB_MEMCG 0x00008000UL /* memcg kmem_cache */
>>> #define SLAB_STORE_USER 0x00010000UL /* DEBUG: Store the
>>> last owner for bug hunting */
>>> #define SLAB_PANIC 0x00040000UL /* Panic if
>>> kmem_cache_create() fails */
>>> /*
>>
>>
>> We'll get to this later, but I dislike adding this flag, since we can just
>> test for existence of a pointer that we need to track anyway in
>> the slab structure.
>
> I might be able to remove this flag. I'll try to get that done in v2.
>
>>
>> This may create some problems when we track it for root memcg, but this is
>> something your patchset does, and I believe we shouldn't.
>>
>>
>>> diff --git a/include/linux/slab_def.h b/include/linux/slab_def.h
>>> index fbd1117..449a0de 100644
>>> --- a/include/linux/slab_def.h
>>> +++ b/include/linux/slab_def.h
>>> @@ -41,6 +41,10 @@ struct kmem_cache {
>>> /* force GFP flags, e.g. GFP_DMA */
>>> gfp_t gfpflags;
>>>
>>> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>>> + int id; /* id used for slab accounting */
>>> +#endif
>>> +
>>

>> What role does it play? Is it the same as the array index in my patchset?
>
> Yes, this is the index into the memcg slab array.
> The id gets allocated when someone does kmem_cache_create().
>
>
>>> size_t colour; /* cache colouring range */
>>> unsigned int colour_off; /* colour offset */
>>> struct kmem_cache *slabp_cache;
>>> @@ -51,7 +55,7 @@ struct kmem_cache {
>>> void (*ctor)(void *obj);
>>>
>>> /* 4) cache creation/removal */
>>> - const char *name;
>>> + char *name;
>>> struct list_head next;
>>>
>>> /* 5) statistics */
>>> @@ -78,9 +82,26 @@ struct kmem_cache {
>>> * variables contain the offset to the user object and its size.
>>> */
>>> int obj_offset;
>>> - int obj_size;
>>> #endif /* CONFIG_DEBUG_SLAB */
>>>
>>> +#if defined(CONFIG_DEBUG_SLAB) ||
>>> defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM)
>>> + int obj_size;
>>> +#endif
>>> +
>>> +#ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
>>> + /* Original cache parameters, used when creating a memcg cache */
>>> + size_t orig_align;
>>> + unsigned long orig_flags;
>>> +
>>> + struct mem_cgroup *memcg;
>>> +
>>> + /* Who we copied from when creating cpuset cache */
>>> + struct kmem_cache *orig_cache;
>>> +
>>> + atomic_t refcnt;
>>> + struct list_head destroyed_list; /* Used when deleting cpuset
>>> cache */
>>> +#endif /* CONFIG_CGROUP_MEM_RES_CTLR_KMEM */
>>
>> I think you're adding way to many things here.
>>
>> I prefer the approach I took of having a memcg pointer, and then having that

>> stuff into memcg. It will be better for people not interested in this
>> feature - like if you compile this in, but then does not mount memcg.
>
> Given that there are only on the order of a hundred different
> kmem_caches, when slab accounting is disabled, I'm not sure the 52
> bytes (or 64?) that are being added here are a big concern.
>
> If you really think this is important, I can move them to a different structure.
>
Yes.

I think this should live in a memcg structure (be it internal or in
memcontrol.h, I don't really care - but prefer the later).

This way we can share it between the alocators (the refcnt can stay,
though).

>>
>>
>> Why CONFIG_SLAB? If this is in memcontrol.c, shouldn't have anything
>> slab-specific here...
>
> I'm not sure this code will compile with another slab allocator.
> I'll look into what I need to do get rid of these #ifdefs.

move whatever is slab-specific to a helper function in slab.c, and let's
try to keep the name stable between slab.c and slub.c (and possibly
slob.c and the next 4 to appear)

That follows the kmem_cache_create and friends work.