Subject: Re: [PATCH v2 04/11] kmem accounting basic infrastructure
Posted by Ying Han on Wed, 15 Aug 2012 19:50:55 GMT
View Forum Message <> Reply to Message

On Tue, Aug 14, 2012 at 9:21 AM, Michal Hocko <mhocko@suse.cz> wrote:
> On Thu 09-08-12 17:01:12, Glauber Costa wrote:
>> This patch adds the basic infrastructure for the accounting of the slab
>> caches. To control that, the following files are created:
>>
>> * memory.kmem.usage_in_bytes
>> * memory.kmem.limit_in_bytes
>> * memory.kmem.failcnt
>> * memory.kmem.max_usage_in_bytes
>>
>> They have the same meaning of their user memory counterparts. They
>> reflect the state of the "kmem" res_counter.
>>
>> The code is not enabled until a limit is set. This can be tested by the
>> flag "kmem_accounted". This means that after the patch is applied, no
>> behavioral changes exists for whoever is still using memcg to control
>> their memory usage.
>>
>> We always account to both user and kernel resource_counters. This
>> effectively means that an independent kernel limit is in place when the
>> limit is set to a lower value than the user memory. A equal or higher
>> value means that the user limit will always hit first, meaning that kmem
>> is effectively unlimited.
>
> Well, it contributes to the user limit so it is not unlimited. It just
> falls under a different limit and it tends to contribute less. This can
> be quite confusing. I am still not sure whether we should mix the two
> things together. If somebody wants to limit the kernel memory he has to
> touch the other limit anyway. Do you have a strong reason to mix the
> user and kernel counters?

The reason to mix the two together is a compromise of the two use
cases we've heard by far. In google, we only need one limit which
limits u & k, and the reclaim kicks in when the total usage hits the
limit.

> My impression was that kernel allocation should simply fail while user
> allocations might reclaim as well. Why should we reclaim just because of
> the kernel allocation (which is unreclaimable from hard limit reclaim
> point of view)?

Some of kernel objects are reclaimable if we have per-memcg shrinker.

> I also think that the whole thing would get much simpler if those two

> are split. Anyway if this is really a must then this should be
> documented here.

What would be the use case you have in your end?

--Ying

> One nit bellow.
>
>> People who want to track kernel memory but not limit it, can set this
>> limit to a very high number (like RESOURCE_MAX - 1page - that no one
>> will ever hit, or equal to the user memory)
>>
>> Signed-off-by: Glauber Costa <glommer@parallels.com>
>> CC: Michal Hocko <mhocko@suse.cz>
>> CC: Johannes Weiner <hannes@cmpxchg.org>
>> Reviewed-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
>> ---
>> mm/memcontrol.c | 69
++-
>> 1 file changed, 68 insertions(+), 1 deletion(-)
>>
>> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
>> index b0e29f4..54e93de 100644
>> --- a/mm/memcontrol.c
>> +++ b/mm/memcontrol.c
> [...]
>> @@ -4046,8 +4059,23 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
>>   break;
>>   if (type == _MEM)
>>   ret = mem_cgroup_resize_limit(memcg, val);
>> -   else
>> +   else if (type == _MEMSWAP)
>>   ret = mem_cgroup_resize_memsw_limit(memcg, val);
>> +   else if (type == _KMEM) {
>> +   ret = res_counter_set_limit(&memcg->kmem, val);
>> +   if (ret)
>> +   break;
>> +   /*
>> +   * Once enabled, can't be disabled. We could in theory
>> +   * disable it if we haven't yet created any caches, or
>> +   * if we can shrink them all to death.
>> +   *
>> +   * But it is not worth the trouble
>> +   */
>> +   if (!memcg->kmem_accounted && val != RESOURCE_MAX)
>> +   memcg->kmem_accounted = true;
>> +   } else

```
>> +                       return -EINVAL;
>>                 break;
>
> This doesn't check for the hierachy so kmem_accounted might not be in
> sync with it's parents. mem_cgroup_create (below) needs to copy
> kmem_accounted down from the parent and the above needs to check if this
> is a similar dance like mem_cgroup_oom_control_write.
>
> [...]
>
>> @@ -5033,6 +5098,7 @@ mem_cgroup_create(struct cgroup *cont)
>>       if (parent && parent->use_hierarchy) {
>>                 res_counter_init(&memcg->res, &parent->res);
>>                 res_counter_init(&memcg->memsw, &parent->memsw);
>> +                res_counter_init(&memcg->kmem, &parent->kmem);
>>                 /*
>>                  * We increment refcnt of the parent to ensure that we can
>>                  * safely access it on res_counter_charge/uncharge.
>> @@ -5043,6 +5109,7 @@ mem_cgroup_create(struct cgroup *cont)
>>       } else {
>>                 res_counter_init(&memcg->res, NULL);
>>                 res_counter_init(&memcg->memsw, NULL);
>> +                res_counter_init(&memcg->kmem, NULL);
>>       }
>>       memcg->last_scanned_node = MAX_NUMNODES;
>>       INIT_LIST_HEAD(&memcg->oom_notify);
>> --
>> 1.7.11.2
>>
>> --
>> To unsubscribe from this list: send the line "unsubscribe cgroups" in
>> the body of a message to majordomo@vger.kernel.org
>> More majordomo info at  http://vger.kernel.org/majordomo-info.html
>
> --
> Michal Hocko
> SUSE Labs
>
> --
> To unsubscribe, send a message with 'unsubscribe linux-mm' in
> the body to majordomo@kvack.org.  For more info on Linux MM,
> see: http://www.linux-mm.org/ .
> Don't email: <a href=mailto:"dont@kvack.org"> email@kvack.org </a>
```