Subject: Re: [PATCH v5 00/14] kmem controller for memcg.
Posted by Glauber Costa on Fri, 19 Oct 2012 09:55:05 GMT
View Forum Message <> Reply to Message

On 10/18/2012 11:21 PM, Andrew Morton wrote:
> On Thu, 18 Oct 2012 20:51:05 +0400
> Glauber Costa <glommer@parallels.com> wrote:
>
>> On 10/18/2012 02:11 AM, Andrew Morton wrote:
>>> On Tue, 16 Oct 2012 14:16:37 +0400
>>> Glauber Costa <glommer@parallels.com> wrote:
>>>
>>>> ...
>>>>
>>>> A general explanation of what this is all about follows:
>>>>
>>>> The kernel memory limitation mechanism for memcg concerns itself with
>>>> disallowing potentially non-reclaimable allocations to happen in exaggerate
>>>> quantities by a particular set of processes (cgroup). Those allocations could
>>>> create pressure that affects the behavior of a different and unrelated set of
>>>> processes.
>>>>
>>>> Its basic working mechanism is to annotate some allocations with the
>>>> _GFP_KMEMCG flag. When this flag is set, the current process allocating will
>>>> have its memcg identified and charged against. When reaching a specific limit,
>>>> further allocations will be denied.
>>>
>>> The need to set _GFP_KMEMCG is rather unpleasing, and makes one wonder
>>> "why didn't it just track all allocations".
>>>
>> This was raised as well by Peter Zijlstra during the memcg summit.
>
> Firstly: please treat any question from a reviewer as an indication
> that information was missing from the changelog or from code comments.
> Ideally all such queries are addressed in later version of the patch
> and changelog.
>
This is in no opposition with me telling a bit that this has been raised
before! =)

>> The
>> answer I gave to him still stands: There is a cost associated with it.
>> We believe it comes down to a trade off situation. How much tracking a
>> particular kind of allocation help vs how much does it cost.
>>
>> The free path is specially more expensive, since it will always incur in
>> a page_cgroup lookup.
>

> OK. But that is a quantitative argument, without any quantities! Do
> we have even an estimate of what this cost will be? Perhaps it's the
> case that, if well implemented, that cost will be acceptable. How do
> we tell?
>

There are two ways:
1) Measuring on various workloads. The workload I measured particularly in here (link in the beginning of this e-mail), showed a 2 - 3 % penalty with the whole thing applied. Truth be told, this was mostly pin-pointed to the slab part, which gets most of its cost from a relay function, and not from the page allocation per-se. But for me, this is enough to tell that there is a cost high enough to bother some.

2) We can infer from past behavior of memcg. It always shown itself as quite an expensive beast. Making it suck faster is a completely separate endeavor. It seems only natural to me to reduce its reach even without specific number for each of the to-be-tracked candidates.

Moreover, there is the cost question, but cost is not *the only* question, as I underlined a few paragraphs below. It is not always obvious how to pinpoint a kernel page to a specific process, so this need to be analyzed on a case-by-case basis. The slab is the hardest one, and it is done. But even then...

If this is still not good enough, and you would like me to measure something else, just let me know.

>>> Does this mean that over time we can expect more sites to get the
>>> _GFP_KMEMCG tagging?
>>
>> We have being doing kernel memory limitation for OpenVZ for a lot of
>> times, using a quite different mechanism. What we do in this work (with
>> slab included), allows us to achieve feature parity with that. It means
>> it is good enough for production environments.
>
> That's really good info.
>
>> Whether or not more people will want other allocations to be tracked, I
>> can't predict. What I do can say is that stack + slab is a very
>> significant part of the memory one potentially cares about, and if
>> anyone else ever have the need for more, it will come down to a
>> trade-off calculation.
>
> OK.
>
>>> If so, are there any special implications, or do
>>> we just go in, do the one-line patch and expect everything to work?

>>
>> With the infrastructure in place, it shouldn't be hard. But it's not
>> necessarily a one-liner either. It depends on what are the pratical
>> considerations for having that specific kind of allocation tied to a
>> memcg. The slab, for instance, that follows this series, is far away
>> from a one-liner: it is in fact, a 19-patch patch series.
>>
>>
>>
>>>
>>> And how *accurate* is the proposed code? What percentage of kernel
>>> memory allocations are unaccounted, typical case and worst case?
>>
>> With both patchsets applied, all memory used for the stack and most of
>> the memory used for slab objects allocated in userspace process contexts
>> are accounted.
>>
>> I honestly don't know which percentage of the total kernel memory this
>> represents.
>
> It sounds like the coverage will be good. What's left over? Random
> get_free_pages() calls and interrupt-time slab allocations?
>

random get_free_pages, vmalloc, ptes. interrupt is left out on purpose,
because we can't cgroup-track something that doesn't have a process context.

> I suppose that there are situations in which network rx could consume
> significant amounts of unaccounted memory?
>

Not unaccounted. This is merged already =)
