Subject: [PATCH 0/3] A few fixes for '[PATCH 00/23] slab+slub accounting for memcg' series
Posted by Anton Vorontsov on Mon, 30 Apr 2012 09:59:19 GMT
View Forum Message <> Reply to Message

Hello Glauber,

On Fri, Apr 20, 2012 at 06:57:08PM -0300, Glauber Costa wrote:
> This is my current attempt at getting the kmem controller
> into a mergeable state. IMHO, all the important bits are there, and it should't
> change *that* much from now on. I am, however, expecting at least a couple more
> interactions before we sort all the edges out.
>
> This series works for both the slub and the slab. One of my main goals was to
> make sure that the interfaces we are creating actually makes sense for both
> allocators.
>
> I did some adaptations to the slab-specific patches, but the bulk of it
> comes from Suleiman's patches. I did the best to use his patches
> as-is where possible so to keep authorship information. When not possible,
> I tried to be fair and quote it in the commit message.
>
> In this series, all existing caches are created per-memcg after its first hit.
> The main reason is, during discussions in the memory summit we came into
> agreement that the fragmentation problems that could arise from creating all
> of them are mitigated by the typically small quantity of caches in the system
> (order of a few megabytes total for sparsely used caches).
> The lazy creation from Suleiman is kept, although a bit modified. For instance,
> I now use a locked scheme instead of cmpxcgh to make sure cache creation won't
> fail due to duplicates, which simplifies things by quite a bit.
>
> The slub is a bit more complex than what I came up with in my slub-only
> series. The reason is we did not need to use the cache-selection logic
> in the allocator itself - it was done by the cache users. But since now
> we are lazy creating all caches, this is simply no longer doable.
>
> I am leaving destruction of caches out of the series, although most
> of the infrastructure for that is here, since we did it in earlier
> series. This is basically because right now Kame is reworking it for
> user memcg, and I like the new proposed behavior a lot more. We all seemed
> to have agreed that reclaim is an interesting problem by itself, and
> is not included in this already too complicated series. Please note
> that this is still marked as experimental, so we have so room. A proper
> shrinker implementation is a hard requirement to take the kmem controller
> out of the experimental state.
>
> I am also not including documentation, but it should only be a matter
> of merging what we already wrote in earlier series plus some additions.

The patches look great, thanks a lot for your work!

I finally tried them, and after a few fixes the kmem accounting
seems to work fine with slab. The fixes will follow this email,
and if they're fine, feel free to fold them into your patches.

However, with slub I'm getting kernel hangs and various traces[1].
It seems that kernel memcg recurses when trying to call
memcg_create_cache_enqueue() -- it calls kmalloc_no_account()
which was introduced to not recurse into memcg, but looking
into 'slub: provide kmalloc_no_account' patch, I don't see
any difference between _no_account and ordinary kmalloc. Hm.

OK, slub apart... the accounting works with slab, which is great.

There's another, more generic question: is there any particular
reason why you don't want to account slab memory for root cgroup?

Personally I'm interested in kmem accounting because I use
memcg for lowmemory notifications. I'm installing events
on the root's memory.usage_in_bytes, and the thresholds values
are calculated like this:

  total_ram - wanted_threshold

So, if we want to get a notification when there's 64 MB memory
left on a 256 MB machine, we'd install an event on the 194 MB
mark (the good thing about usage_in_bytes, is that it does
account file caches, so the formula is simple).

Obviously, without kmem accounting the formula can be very
imprecise when kernel (e.g. hw drivers) itself start using a
lot of memory. With root's slab accounting the problem
would be solved, but for some reason you deliberately do not
want to account it for root cgroup. I suspect that there are
some performance concerns?..

Thanks,

[1]

BUG: unable to handle kernel paging request at ffffffffb2e80900
IP: [<ffffffff8105940c>] check_preempt_wakeup+0x3c/0x210
PGD 160d067 PUD 1611063 PMD 0
Thread overran stack, or stack corrupted
Oops: 0000 [#1] SMP
CPU 0

Pid: 943, comm: bash Not tainted 3.4.0-rc4+ #34 Bochs Bochs
RIP: 0010:[<ffffffff8105940c>] [<ffffffff8105940c>] check_preempt_wakeup+0x3c/0x210
RSP: 0018:ffff880006305ee8 EFLAGS: 00010006
RAX: 00000000000109c0 RBX: ffff8800071b4e20 RCX: ffff880006306000
RDX: 0000000000000000 RSI: 0000000006306028 RDI: ffff880007c109c0
RBP: ffff880006305f28 R08: 0000000000000000 R09: 0000000000000001
R10: 0000000000000000 R11: 0000000000000000 R12: ffff880007c109c0
R13: ffff88000644ddc0 R14: ffff8800071b4e68 R15: 0000000000000000
FS: 00007fad1244c700(0000) GS:ffff880007c00000(0000) knlGS:0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 0000000080050033
CR2: ffffffffb2e80900 CR3: 00000000063b8000 CR4: 00000000000006b0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 00000000ffff0ff0 DR7: 0000000000000400
Process bash (pid: 943, threadinfo ffff880006306000, task ffff88000644ddc0)
Stack:
 0000000000000000 ffff88000644de08 ffff880007c109c0 ffff880007c109c0
 ffff8800071b4e20 0000000000000000 0000000000000000 0000000000000000
 ffff880006305f48 ffffffff81053304 ffff880007c109c0 ffff880007c109c0
Call Trace:
Code: 76 48 41 55 41 54 49 89 fc 53 48 89 f3 48 83 ec 18 4c 8b af e0 07 00 00 49 8d 4d 48 48 89 4d c8 49 8b 4d 08 4c 3b 75 c8 8b 71 18 <48> 8b 34 f5 c0 07 65 81 48 8b bc 30 a8 00 00 00 8b 35 3a 3f 5c
RIP [<ffffffff8105940c>] check_preempt_wakeup+0x3c/0x210
 RSP <ffff880006305ee8>
CR2: ffffffffb2e80900
---[end trace 78fa9c86bebb1214]---

--
Anton Vorontsov
Email: cbouatmailru@gmail.com