Subject: Re: [PATCH v2 00/11] Request for Inclusion: kmem controller for memcg.
Posted by Ying Han on Fri, 17 Aug 2012 21:37:21 GMT
View Forum Message <> Reply to Message

On Thu, Aug 9, 2012 at 6:01 AM, Glauber Costa <glommer@parallels.com> wrote:
> Hi,
>
> This is the first part of the kernel memory controller for memcg. It has been
> discussed many times, and I consider this stable enough to be on tree. A follow
> up to this series are the patches to also track slab memory. They are not
> included here because I believe we could benefit from merging them separately
> for better testing coverage. If there are any issues preventing this to be
> merged, let me know. I'll be happy to address them.
>
> The slab patches are also mature in my self evaluation and could be merged not
> too long after this. For the reference, the last discussion about them happened
> at http://lwn.net/Articles/508087/
>
> A (throwaway) git tree with them is placed at:
>
> git://github.com/glommer/linux.git kmemcg-slab

I would like to make a kernel on the tree and run some perf tests on
it. However the kernel
doesn't boot due to "divide error: 0000 [#1] SMP".
https://lkml.org/lkml/2012/5/21/502

I believe the issue has been fixed (didn't look through) and can you
do a rebase on your tree?

--Ying

>
> 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.
>
> One example of such problematic pressure that can be prevented by this work is
> a fork bomb conducted in a shell. We prevent it by noting that processes use a

> limited amount of stack pages. Seen this way, a fork bomb is just a special
> case of resource abuse. If the offender is unable to grab more pages for the
> stack, no new processes can be created.
>
> There are also other things the general mechanism protects against. For
> example, using too much of pinned dentry and inode cache, by touching files an
> leaving them in memory forever.
>
> In fact, a simple:
>
> while true; do mkdir x; cd x; done
>
> can halt your system easily because the file system limits are hard to reach
> (big disks), but the kernel memory is not. Those are examples, but the list
> certainly don't stop here.
>
> An important use case for all that, is concerned with people offering hosting
> services through containers. In a physical box we can put a limit to some
> resources, like total number of processes or threads. But in an environment
> where each independent user gets its own piece of the machine, we don't want a
> potentially malicious user to destroy good users' services.
>
> This might be true for systemd as well, that now groups services inside
> cgroups. They generally want to put forward a set of guarantees that limits the
> running service in a variety of ways, so that if they become badly behaved,
> they won't interfere with the rest of the system.
>
> There is, of course, a cost for that. To attempt to mitigate that, static
> branches are used to make sure that even if the feature is compiled in with
> potentially a lot of memory cgroups deployed this code will only be enabled
> after the first user of this service configures any limit. Limits lower than
> the user limit effectively means there is a separate kernel memory limit that
> may be reached independently than the user limit. Values equal or greater than
> the user limit implies only that kernel memory is tracked. This provides a
> unified vision of "maximum memory", be it kernel or user memory. Because this
> is all default-off, existing deployments will see no change in behavior.
>
> Glauber Costa (9):
> memcg: change defines to an enum
> kmem accounting basic infrastructure
> Add a __GFP_KMEMCG flag
> memcg: kmem controller infrastructure
> mm: Allocate kernel pages to the right memcg
> memcg: disable kmem code when not in use.
> memcg: propagate kmem limiting information to children
> memcg: allow a memcg with kmem charges to be destructed.
> protect architectures where THREAD_SIZE >= PAGE_SIZE against fork
> bombs

>
> Suleiman Souhlal (2):
> memcg: Make it possible to use the stock for more than one page.
> memcg: Reclaim when more than one page needed.
>
> include/linux/gfp.h | 10 +-
> include/linux/memcontrol.h | 82 ++++++++
> include/linux/thread_info.h | 2 +
> kernel/fork.c | 4 +-
> mm/memcontrol.c | 443 +++---
> mm/page_alloc.c | 38 ++++
> 6 files changed, 546 insertions(+), 33 deletions(-)
>
> --
> 1.7.11.2
>
> --
> 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: email@kvack.org
