Subject: [PATCH 00/10] memcg kmem limitation - slab.
Posted by Glauber Costa on Wed, 25 Jul 2012 14:38:11 GMT
View Forum Message <> Reply to Message

Hi,

This is the slab part of the kmem limitation mechanism in its last form. I would like to have comments on it to see if we can agree in its form. I consider it mature, since it doesn't change much in essence over the last forms. However, I would still prefer to defer merging it and merge the stack-only patchset first (even if inside the same merge window). That patchset contains most of the infrastructure needed here, and merging them separately would not only reduce the complexity for reviewers, but allow us a chance to have independent testing on them both. I would also likely benefit from some extra testing, to make sure the recent changes didn't introduce anything bad.

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 exagerate 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, there is 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 (10):
  slab/slub: struct memcg_params
  consider a memcg parameter in kmem_create_cache
  memcg: infrastructure to match an allocation to the right cache
  memcg: skip memcg kmem allocations in specified code regions
  slab: allow enable_cpu_cache to use preset values for its tunables
  sl[au]b: Allocate objects from memcg cache
  memcg: destroy memcg caches
  memcg/sl[au]b Track all the memcg children of a kmem_cache.
  slab: slab-specific propagation changes.
  memcg/sl[au]b: shrink dead caches

 include/linux/memcontrol.h | 54 ++++++
 include/linux/sched.h | 1 +
 include/linux/slab.h | 25 +++
 include/linux/slab_def.h | 4 +
 include/linux/slub_def.h | 21 ++-
 init/Kconfig | 2 +-
 mm/memcontrol.c | 414 ++-
 mm/slab.c | 57 +++++--
 mm/slab.h | 55 +++++-
 mm/slab_common.c | 69 +++++++-
 mm/slub.c | 25 ++-
 11 files changed, 702 insertions(+), 25 deletions(-)

--
1.7.10.4