Subject: [PATCH 00/23] slab+slub accounting for memcg
Posted by Glauber Costa on Fri, 20 Apr 2012 21:48:57 GMT
View Forum Message <> Reply to Message

Hi,

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.

Glauber Costa (19):

slub: don't create a copy of the name string in kmem_cache_create
slub: always get the cache from its page in kfree
slab: rename gfpflags to allocflags
slab: use obj_size field of struct kmem_cache when not debugging
change defines to an enum
don't force return value checking in res_counter_charge_nofail
kmem slab accounting basic infrastructure
slab/slub: struct memcg_params
slub: consider a memcg parameter in kmem_create_cache
slab: pass memcg parameter to kmem_cache_create
slub: create duplicate cache
slub: provide kmalloc_no_account
slab: create duplicate cache
slab: provide kmalloc_no_account
kmem controller charge/uncharge infrastructure
slub: charge allocation to a memcg
slab: per-memcg accounting of slab caches
memcg: disable kmem code when not in use.
slub: create slabinfo file for memcg

Suleiman Souhlal (4):
memcg: Make it possible to use the stock for more than one page.
memcg: Reclaim when more than one page needed.
memcg: Track all the memcg children of a kmem_cache.
memcg: Per-memcg memory.kmem.slabinfo file.

include/linux/memcontrol.h | 87 ++++++
include/linux/res_counter.h | 2 +-
include/linux/slab.h | 26 ++
include/linux/slab_def.h | 77 ++++++-
include/linux/slub_def.h | 36 +++-
init/Kconfig | 2 +-
mm/memcontrol.c | 607 ++--
mm/slab.c | 390 ++++++++++++++++++++++++++++++++++++++

registering caches have to keep a copy themselves anyway, since code
needs to work on all allocators.

Having slab create a copy of it as well may very well be the right
thing to do: but at this point, the callers are already there

My motivation for it comes from the kmem slab cache controller for
memcg. Because we create duplicate caches, having a more consistent
behavior here really helps.

I am sending the patch, however, more to probe on your opinion about
it. If you guys agree, but don't want to merge it - since it is not
fixing anything, nor improving any situation etc, I am more than happy
to carry it in my series until it gets merged (fingers crossed).

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Christoph Lameter <cl@linux.com>
CC: Pekka Enberg <penberg@cs.helsinki.fi>
---
 mm/slub.c | 14 ++----------
 1 files changed, 2 insertions(+), 12 deletions(-)

diff --git a/mm/slub.c b/mm/slub.c
index ffe13fd..af8cee9 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -3925,7 +3925,6 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t
size,
   size_t align, unsigned long flags, void (*ctor)(void *))
 {
  struct kmem_cache *s;
- char *n;

  if (WARN_ON(!name))
   return NULL;
@@ -3949,26 +3948,20 @@ struct kmem_cache *kmem_cache_create(const char *name, size_t
size,
   return s;
  }

- n = kstrdup(name, GFP_KERNEL);
- if (!n)
- goto err;
-
  s = kmalloc(kmem_size, GFP_KERNEL);
  if (s) {
- if (kmem_cache_open(s, n,
+ if (kmem_cache_open(s, name,

```
     size, align, flags, ctor)) {
    list_add(&s->list, &slab_caches);
    up_write(&slub_lock);
    if (sysfs_slab_add(s)) {
     down_write(&slub_lock);
     list_del(&s->list);
-    kfree(n);
     kfree(s);
     goto err;
    }
    return s;
   }
-  kfree(n);
   kfree(s);
  }
 err:
@@ -5212,7 +5205,6 @@ static void kmem_cache_release(struct kobject *kobj)
 {
  struct kmem_cache *s = to_slab(kobj);

- kfree(s->name);
  kfree(s);
 }

@@ -5318,11 +5310,9 @@ static int sysfs_slab_add(struct kmem_cache *s)
   return err;
  }
  kobject_uevent(&s->kobj, KOBJ_ADD);
- if (!unmergeable) {
+ if (!unmergeable)
   /* Setup first alias */
   sysfs_slab_alias(s, s->name);
-  kfree(name);
- }
  return 0;
 }

--
1.7.7.6
```

Subject: [PATCH 02/23] slub: always get the cache from its page in kfree
Posted by Glauber Costa on Fri, 20 Apr 2012 21:48:59 GMT
View Forum Message <> Reply to Message

struct page already have this information. If we start chaining
caches, this information will always be more trustworthy than
whatever is passed into the function

Signed-off-by: Glauber Costa <glommer@parallels.com>
---
 mm/slub.c | 2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)

diff --git a/mm/slub.c b/mm/slub.c
index af8cee9..2652e7c 100644
--- a/mm/slub.c
+++ b/mm/slub.c
@@ -2600,7 +2600,7 @@ void kmem_cache_free(struct kmem_cache *s, void *x)

  page = virt_to_head_page(x);

- slab_free(s, page, x, _RET_IP_);
+ slab_free(page->slab, page, x, _RET_IP_);

  trace_kmem_cache_free(_RET_IP_, x);
 }
--
1.7.7.6

Subject: [PATCH 03/23] slab: rename gfpflags to allocflags
Posted by Glauber Costa on Fri, 20 Apr 2012 21:49:00 GMT
View Forum Message <> Reply to Message

A consistent name with slub saves us an acessor function.
In both caches, this field represents the same thing. We would
like to use it from the mem_cgroup code.

Signed-off-by: Glauber Costa <glommer@parallels.com>
---
 include/linux/slab_def.h | 2 +-
 mm/slab.c | 10 +++++-----
 2 files changed, 6 insertions(+), 6 deletions(-)

diff --git a/include/linux/slab_def.h b/include/linux/slab_def.h
index fbd1117..d41effe 100644
--- a/include/linux/slab_def.h
+++ b/include/linux/slab_def.h
@@ -39,7 +39,7 @@ struct kmem_cache {
  unsigned int gfporder;

  /* force GFP flags, e.g. GFP_DMA */
- gfp_t gfpflags;
+ gfp_t allocflags;

```
  size_t colour;   /* cache colouring range */
  unsigned int colour_off; /* colour offset */
diff --git a/mm/slab.c b/mm/slab.c
index e901a36..c6e5ab8 100644
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -1798,7 +1798,7 @@ static void *kmem_getpages(struct kmem_cache *cachep, gfp_t flags,
int nodeid)
  flags |= __GFP_COMP;
 #endif

- flags |= cachep->gfpflags;
+ flags |= cachep->allocflags;
  if (cachep->flags & SLAB_RECLAIM_ACCOUNT)
   flags |= __GFP_RECLAIMABLE;

@@ -2508,9 +2508,9 @@ kmem_cache_create (const char *name, size_t size, size_t align,
  cachep->colour = left_over / cachep->colour_off;
  cachep->slab_size = slab_size;
  cachep->flags = flags;
- cachep->gfpflags = 0;
+ cachep->allocflags = 0;
  if (CONFIG_ZONE_DMA_FLAG && (flags & SLAB_CACHE_DMA))
-  cachep->gfpflags |= GFP_DMA;
+  cachep->allocflags |= GFP_DMA;
  cachep->buffer_size = size;
  cachep->reciprocal_buffer_size = reciprocal_value(size);

@@ -2857,9 +2857,9 @@ static void kmem_flagcheck(struct kmem_cache *cachep, gfp_t flags)
 {
  if (CONFIG_ZONE_DMA_FLAG) {
   if (flags & GFP_DMA)
-   BUG_ON(!(cachep->gfpflags & GFP_DMA));
+   BUG_ON(!(cachep->allocflags & GFP_DMA));
   else
-   BUG_ON(cachep->gfpflags & GFP_DMA);
+   BUG_ON(cachep->allocflags & GFP_DMA);
  }
 }

--
1.7.7.6
```

Subject: [PATCH 04/23] memcg: Make it possible to use the stock for more than one page.
Posted by Glauber Costa on Fri, 20 Apr 2012 21:49:01 GMT

From: Suleiman Souhlal <ssouhlal@FreeBSD.org>

Signed-off-by: Suleiman Souhlal <suleiman@google.com>
---
 mm/memcontrol.c | 18 +++++++++---------
 1 files changed, 9 insertions(+), 9 deletions(-)

```
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 932a734..4b94b2d 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -1998,19 +1998,19 @@ static DEFINE_PER_CPU(struct memcg_stock_pcp,
memcg_stock);
 static DEFINE_MUTEX(percpu_charge_mutex);

 /*
- * Try to consume stocked charge on this cpu. If success, one page is consumed
- * from local stock and true is returned. If the stock is 0 or charges from a
- * cgroup which is not current target, returns false. This stock will be
- * refilled.
+ * Try to consume stocked charge on this cpu. If success, nr_pages pages are
+ * consumed from local stock and true is returned. If the stock is 0 or
+ * charges from a cgroup which is not current target, returns false.
+ * This stock will be refilled.
  */
-static bool consume_stock(struct mem_cgroup *memcg)
+static bool consume_stock(struct mem_cgroup *memcg, int nr_pages)
 {
  struct memcg_stock_pcp *stock;
  bool ret = true;

  stock = &get_cpu_var(memcg_stock);
- if (memcg == stock->cached && stock->nr_pages)
-  stock->nr_pages--;
+ if (memcg == stock->cached && stock->nr_pages >= nr_pages)
+  stock->nr_pages -= nr_pages;
  else /* need to call res_counter_charge */
   ret = false;
  put_cpu_var(memcg_stock);
@@ -2309,7 +2309,7 @@ again:
   VM_BUG_ON(css_is_removed(&memcg->css));
   if (mem_cgroup_is_root(memcg))
    goto done;
-  if (nr_pages == 1 && consume_stock(memcg))
+  if (consume_stock(memcg, nr_pages))
    goto done;
   css_get(&memcg->css);
```

```
  } else {
@@ -2334,7 +2334,7 @@ again:
    rcu_read_unlock();
    goto done;
   }
-  if (nr_pages == 1 && consume_stock(memcg)) {
+  if (consume_stock(memcg, nr_pages)) {
    /*
     * It seems dagerous to access memcg without css_get().
     * But considering how consume_stok works, it's not
--
1.7.7.6
```

Subject: [PATCH 05/23] memcg: Reclaim when more than one page needed.
Posted by Glauber Costa on Fri, 20 Apr 2012 21:49:02 GMT
View Forum Message <> Reply to Message

From: Suleiman Souhlal <ssouhlal@FreeBSD.org>

mem_cgroup_do_charge() was written before slab accounting, and expects
three cases: being called for 1 page, being called for a stock of 32 pages,
or being called for a hugepage. If we call for 2 pages (and several slabs
used in process creation are such, at least with the debug options I had),
it assumed it's being called for stock and just retried without reclaiming.

Fix that by passing down a minsize argument in addition to the csize.

And what to do about that (csize == PAGE_SIZE && ret) retry? If it's
needed at all (and presumably is since it's there, perhaps to handle
races), then it should be extended to more than PAGE_SIZE, yet how far?
And should there be a retry count limit, of what? For now retry up to
COSTLY_ORDER (as page_alloc.c does), stay safe with a cond_resched(),
and make sure not to do it if __GFP_NORETRY.

Signed-off-by: Suleiman Souhlal <suleiman@google.com>
---
 mm/memcontrol.c | 18 ++++++++++++-------
 1 files changed, 11 insertions(+), 7 deletions(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 4b94b2d..cbffc4c 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -2187,7 +2187,8 @@ enum {
 };

 static int mem_cgroup_do_charge(struct mem_cgroup *memcg, gfp_t gfp_mask,

```
-    unsigned int nr_pages, bool oom_check)
+    unsigned int nr_pages, unsigned int min_pages,
+    bool oom_check)
 {
  unsigned long csize = nr_pages * PAGE_SIZE;
  struct mem_cgroup *mem_over_limit;
@@ -2210,18 +2211,18 @@ static int mem_cgroup_do_charge(struct mem_cgroup *memcg,
gfp_t gfp_mask,
  } else
   mem_over_limit = mem_cgroup_from_res_counter(fail_res, res);
  /*
-  * nr_pages can be either a huge page (HPAGE_PMD_NR), a batch
-  * of regular pages (CHARGE_BATCH), or a single regular page (1).
-  *
   * Never reclaim on behalf of optional batching, retry with a
   * single page instead.
   */
- if (nr_pages == CHARGE_BATCH)
+ if (nr_pages > min_pages)
   return CHARGE_RETRY;

  if (!(gfp_mask & __GFP_WAIT))
   return CHARGE_WOULDBLOCK;

+ if (gfp_mask & __GFP_NORETRY)
+  return CHARGE_NOMEM;
+
  ret = mem_cgroup_reclaim(mem_over_limit, gfp_mask, flags);
  if (mem_cgroup_margin(mem_over_limit) >= nr_pages)
   return CHARGE_RETRY;
@@ -2234,8 +2235,10 @@ static int mem_cgroup_do_charge(struct mem_cgroup *memcg, gfp_t
gfp_mask,
   * unlikely to succeed so close to the limit, and we fall back
   * to regular pages anyway in case of failure.
   */
- if (nr_pages == 1 && ret)
+ if (nr_pages <= (PAGE_SIZE << PAGE_ALLOC_COSTLY_ORDER) && ret) {
+  cond_resched();
   return CHARGE_RETRY;
+ }

  /*
   * At task move, charge accounts can be doubly counted. So, it's
@@ -2369,7 +2372,8 @@ again:
    nr_oom_retries = MEM_CGROUP_RECLAIM_RETRIES;
   }

-  ret = mem_cgroup_do_charge(memcg, gfp_mask, batch, oom_check);
```

```
+  ret = mem_cgroup_do_charge(memcg, gfp_mask, batch, nr_pages,
+      oom_check);
   switch (ret) {
   case CHARGE_OK:
    break;
--
1.7.7.6
```

Subject: [PATCH 06/23] slab: use obj_size field of struct kmem_cache when not debugging
Posted by Glauber Costa on Fri, 20 Apr 2012 21:49:03 GMT
View Forum Message <> Reply to Message

The kmem controller needs to keep track of the object size of
a cache so it can later on create a per-memcg duplicate. Logic
to keep track of that already exists, but it is only enable while
debugging.

This patch makes it also available when the kmem controller code
is compiled in.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Christoph Lameter <cl@linux.com>
CC: Pekka Enberg <penberg@cs.helsinki.fi>
---
 include/linux/slab_def.h | 4 +++-
 mm/slab.c | 37 ++++++

```
--- a/mm/slab.c
+++ b/mm/slab.c
@@ -413,8 +413,28 @@ static void kmem_list3_init(struct kmem_list3 *parent)
 #define STATS_INC_FREEMISS(x) do { } while (0)
 #endif

-#if DEBUG
+#if defined(CONFIG_DEBUG_SLAB) || defined(CONFIG_CGROUP_MEM_RES_CTLR_KMEM)
+static int obj_size(struct kmem_cache *cachep)
+{
+ return cachep->obj_size;
+}
+static void set_obj_size(struct kmem_cache *cachep, int size)
+{
+ cachep->obj_size = size;
+}
+
+#else
+static int obj_size(struct kmem_cache *cachep)
+{
+ return cachep->buffer_size;
+}
+
+static void set_obj_size(struct kmem_cache *cachep, int size)
+{
+}
+#endif

+#if DEBUG
 /*
  * memory layout of objects:
  * 0  : objp
@@ -433,11 +453,6 @@ static int obj_offset(struct kmem_cache *cachep)
  return cachep->obj_offset;
 }

-static int obj_size(struct kmem_cache *cachep)
-{
- return cachep->obj_size;
-}
-
 static unsigned long long *dbg_redzone1(struct kmem_cache *cachep, void *objp)
 {
  BUG_ON(!(cachep->flags & SLAB_RED_ZONE));
@@ -465,7 +480,6 @@ static void **dbg_userword(struct kmem_cache *cachep, void *objp)
 #else

 #define obj_offset(x)   0
```

```
-#define obj_size(cachep)  (cachep->buffer_size)
 #define dbg_redzone1(cachep, objp) ({BUG(); (unsigned long long *)NULL;})
 #define dbg_redzone2(cachep, objp) ({BUG(); (unsigned long long *)NULL;})
 #define dbg_userword(cachep, objp) ({BUG(); (void **)NULL;})
@@ -1555,9 +1569,9 @@ void __init kmem_cache_init(void)
   */
  cache_cache.buffer_size = offsetof(struct kmem_cache, array[nr_cpu_ids]) +
       nr_node_ids * sizeof(struct kmem_list3 *);
-#if DEBUG
- cache_cache.obj_size = cache_cache.buffer_size;
-#endif
+
+ set_obj_size(&cache_cache, cache_cache.buffer_size);
+
  cache_cache.buffer_size = ALIGN(cache_cache.buffer_size,
      cache_line_size());
  cache_cache.reciprocal_buffer_size =
@@ -2418,8 +2432,9 @@ kmem_cache_create (const char *name, size_t size, size_t align,
   goto oops;

  cachep->nodelists = (struct kmem_list3 **)&cachep->array[nr_cpu_ids];
+
+ set_obj_size(cachep, size);
 #if DEBUG
- cachep->obj_size = size;

  /*
   * Both debugging options require word-alignment which is calculated
--
1.7.7.6
```

Subject: [PATCH 07/23] change defines to an enum
Posted by Glauber Costa on Fri, 20 Apr 2012 21:49:04 GMT
View Forum Message <> Reply to Message

This is just a cleanup patch for clarity of expression.
In earlier submissions, people asked it to be in a separate
patch, so here it is.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Michal Hocko <mhocko@suse.cz>
CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Johannes Weiner <hannes@cmpxchg.org>
---
 mm/memcontrol.c | 9 ++++++---
 1 files changed, 6 insertions(+), 3 deletions(-)

```
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index cbffc4c..2810228 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -374,9 +374,12 @@ enum charge_type {
 };

 /* for encoding cft->private value on file */
-#define _MEM   (0)
-#define _MEMSWAP  (1)
-#define _OOM_TYPE  (2)
+enum res_type {
+ _MEM,
+ _MEMSWAP,
+ _OOM_TYPE,
+};
+
 #define MEMFILE_PRIVATE(x, val) (((x) << 16) | (val))
 #define MEMFILE_TYPE(val) (((val) >> 16) & 0xffff)
 #define MEMFILE_ATTR(val) ((val) & 0xffff)
--
1.7.7.6
```

Subject: [PATCH 08/23] don't force return value checking in res_counter_charge_nofail

Posted by Glauber Costa on Fri, 20 Apr 2012 21:49:05 GMT

View Forum Message <> Reply to Message

Since we will succeed with the allocation no matter what, there
isn't the need to use __must_check with it. It can very well
be optional.

Signed-off-by: Glauber Costa <glommer@parallels.com>
CC: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
CC: Johannes Weiner <hannes@cmpxchg.org>
CC: Michal Hocko <mhocko@suse.cz>
---
 include/linux/res_counter.h | 2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)

```
diff --git a/include/linux/res_counter.h b/include/linux/res_counter.h
index da81af0..f7621cf 100644
--- a/include/linux/res_counter.h
+++ b/include/linux/res_counter.h
@@ -119,7 +119,7 @@ int __must_check res_counter_charge_locked(struct res_counter
*counter,
   unsigned long val);
```

```
 int __must_check res_counter_charge(struct res_counter *counter,
   unsigned long val, struct res_counter **limit_fail_at);
-int __must_check res_counter_charge_nofail(struct res_counter *counter,
+int res_counter_charge_nofail(struct res_counter *counter,
   unsigned long val, struct res_counter **limit_fail_at);

 /*
--
1.7.7.6
```

Subject: Re: [PATCH 00/23] slab+slub accounting for memcg
Posted by Glauber Costa on Fri, 20 Apr 2012 22:01:33 GMT
View Forum Message <> Reply to Message

On 04/20/2012 06:48 PM, Glauber Costa wrote:
> Hi,
>
> 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.
>
> Glauber Costa (19):
> slub: don't create a copy of the name string in kmem_cache_create
> slub: always get the cache from its page in kfree
> slab: rename gfpflags to allocflags
> slab: use obj_size field of struct kmem_cache when not debugging
> change defines to an enum
> don't force return value checking in res_counter_charge_nofail
> kmem slab accounting basic infrastructure
> slab/slub: struct memcg_params
> slub: consider a memcg parameter in kmem_create_cache
> slab: pass memcg parameter to kmem_cache_create
> slub: create duplicate cache
> slub: provide kmalloc_no_account
> slab: create duplicate cache
> slab: provide kmalloc_no_account
> kmem controller charge/uncharge infrastructure
> slub: charge allocation to a memcg
> slab: per-memcg accounting of slab caches
> memcg: disable kmem code when not in use.
> slub: create slabinfo file for memcg
>
> Suleiman Souhlal (4):
> memcg: Make it possible to use the stock for more than one page.
> memcg: Reclaim when more than one page needed.
> memcg: Track all the memcg children of a kmem_cache.
> memcg: Per-memcg memory.kmem.slabinfo file.
>
I am sorry.

My mail server seems to be going crazy in the middle of the submission,
and the whole patchset is not going through (and a part of it got
duplicated)

I'll post the whole series later, when I figure out what's wrong.

Subject: Re: [PATCH 00/23] slab+slub accounting for memcg
Posted by Suleiman Souhlal on Mon, 30 Apr 2012 21:43:28 GMT
View Forum Message <> Reply to Message

On Fri, Apr 20, 2012 at 2:48 PM, Glauber Costa <glommer@parallels.com> wrote:
> Hi,
>
> 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.

Thanks a lot for doing this.

> 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.

I actually noticed that, at least for slab, the cmpxchg could never
fail due to kmem_cache_create() already making sure that duplicate
caches could not be created at the same time, while holding
cache_mutex_mutex.

I do like your simplification though.

>
> 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.

We will have to be careful for cache destruction.
I found several races between allocation and destruction, in my patchset.

I think we should consider doing the uncharging of kmem when
destroying a memcg in mem_cgroup_destroy() instead of in
pre_destroy(), because it's still possible that there are threads in
the cgroup while pre_destroy() is being called (or for threads to be
moved into the cgroup).

-- Suleiman

Subject: Re: [PATCH 00/23] slab+slub accounting for memcg
Posted by Glauber Costa on Wed, 02 May 2012 15:14:37 GMT
View Forum Message <> Reply to Message

On 04/30/2012 06:43 PM, Suleiman Souhlal wrote:
>> 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.
> We will have to be careful for cache destruction.
> I found several races between allocation and destruction, in my patchset.
>
> I think we should consider doing the uncharging of kmem when
> destroying a memcg in mem_cgroup_destroy() instead of in
> pre_destroy(), because it's still possible that there are threads in
> the cgroup while pre_destroy() is being called (or for threads to be
> moved into the cgroup).

I found some problems here as well.
I am trying to work ontop of what Kamezawa posted for pre_destroy()
rework. I have one or two incorrect uncharging issues to solve, that's
actually what is holding me for posting a new version.

expected soon
