Subject: Re: [PATCH v2 04/11] kmem accounting basic infrastructure
Posted by KAMEZAWA Hiroyuki on Fri, 10 Aug 2012 17:02:32 GMT
View Forum Message <> Reply to Message

(2012/08/09 22:01), Glauber Costa wrote:
> This patch adds the basic infrastructure for the accounting of the slab
> caches. To control that, the following files are created:
>
> * memory.kmem.usage_in_bytes
> * memory.kmem.limit_in_bytes
> * memory.kmem.failcnt
> * memory.kmem.max_usage_in_bytes
>
> They have the same meaning of their user memory counterparts. They
> reflect the state of the "kmem" res_counter.
>
> The code is not enabled until a limit is set. This can be tested by the
> flag "kmem_accounted". This means that after the patch is applied, no
> behavioral changes exists for whoever is still using memcg to control
> their memory usage.
>
> We always account to both user and kernel resource_counters. This
> effectively means that an independent kernel limit is in place when the
> limit is set to a lower value than the user memory. A equal or higher
> value means that the user limit will always hit first, meaning that kmem
> is effectively unlimited.
>
> People who want to track kernel memory but not limit it, can set this
> limit to a very high number (like RESOURCE_MAX - 1page - that no one
> will ever hit, or equal to the user memory)
>
> Signed-off-by: Glauber Costa <glommer@parallels.com>
> CC: Michal Hocko <mhocko@suse.cz>
> CC: Johannes Weiner <hannes@cmpxchg.org>
> Reviewed-by: Kamezawa Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

Could you add a patch for documentation of this new interface and a text
explaining the behavior of "kmem_accounting" ?

Hm, my concern is the difference of behavior between user page accounting and
kmem accounting...but this is how tcp-accounting is working.

Once you add Documentation, it's okay to add my Ack.

Thanks,
-Kame

```
> ---
>  mm/memcontrol.c | 69
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++-
>  1 file changed, 68 insertions(+), 1 deletion(-)
>
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index b0e29f4..54e93de 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -273,6 +273,10 @@ struct mem_cgroup {
>  };
>
>  /*
> +  * the counter to account for kernel memory usage.
> +  */
> + struct res_counter kmem;
> + /*
>  * Per cgroup active and inactive list, similar to the
>  * per zone LRU lists.
>  */
> @@ -287,6 +291,7 @@ struct mem_cgroup {
>  * Should the accounting and control be hierarchical, per subtree?
>  */
>  bool use_hierarchy;
> + bool kmem_accounted;
>
>  bool  oom_lock;
>  atomic_t under_oom;
> @@ -397,6 +402,7 @@ enum res_type {
>  _MEM,
>  _MEMSWAP,
>  _OOM_TYPE,
> + _KMEM,
>  };
>
>  #define MEMFILE_PRIVATE(x, val) ((x) << 16 | (val))
> @@ -1499,6 +1505,10 @@ done:
>  res_counter_read_u64(&memcg->memsw, RES_USAGE) >> 10,
>  res_counter_read_u64(&memcg->memsw, RES_LIMIT) >> 10,
>  res_counter_read_u64(&memcg->memsw, RES_FAILCNT));
> + printk(KERN_INFO "kmem: usage %llukB, limit %llukB, failcnt %llu\n",
> +  res_counter_read_u64(&memcg->kmem, RES_USAGE) >> 10,
> +  res_counter_read_u64(&memcg->kmem, RES_LIMIT) >> 10,
> +  res_counter_read_u64(&memcg->kmem, RES_FAILCNT));
>
>  mem_cgroup_print_oom_stat(memcg);
>  }
> @@ -4008,6 +4018,9 @@ static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype
```

```
*cft,
>     else
>      val = res_counter_read_u64(&memcg->memsw, name);
>     break;
> + case _KMEM:
> +  val = res_counter_read_u64(&memcg->kmem, name);
> +  break;
>    default:
>     BUG();
>    }
> @@ -4046,8 +4059,23 @@ static int mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
>      break;
>     if (type == _MEM)
>      ret = mem_cgroup_resize_limit(memcg, val);
> -  else
> +  else if (type == _MEMSWAP)
>      ret = mem_cgroup_resize_memsw_limit(memcg, val);
> +  else if (type == _KMEM) {
> +   ret = res_counter_set_limit(&memcg->kmem, val);
> +   if (ret)
> +    break;
> +   /*
> +    * Once enabled, can't be disabled. We could in theory
> +    * disable it if we haven't yet created any caches, or
> +    * if we can shrink them all to death.
> +    *
> +    * But it is not worth the trouble
> +    */
> +   if (!memcg->kmem_accounted && val != RESOURCE_MAX)
> +    memcg->kmem_accounted = true;
> +  } else
> +   return -EINVAL;
>     break;
>    case RES_SOFT_LIMIT:
>     ret = res_counter_memparse_write_strategy(buffer, &val);
> @@ -4113,12 +4141,16 @@ static int mem_cgroup_reset(struct cgroup *cont, unsigned int
event)
>    case RES_MAX_USAGE:
>     if (type == _MEM)
>      res_counter_reset_max(&memcg->res);
> +  else if (type == _KMEM)
> +   res_counter_reset_max(&memcg->kmem);
>     else
>      res_counter_reset_max(&memcg->memsw);
>     break;
>    case RES_FAILCNT:
>     if (type == _MEM)
>      res_counter_reset_failcnt(&memcg->res);
```

```
> +  else if (type == _KMEM)
> +   res_counter_reset_failcnt(&memcg->kmem);
>     else
>      res_counter_reset_failcnt(&memcg->memsw);
>     break;
> @@ -4672,6 +4704,33 @@ static int mem_cgroup_oom_control_write(struct cgroup *cgrp,
>    }
>
>    #ifdef CONFIG_MEMCG_KMEM
> +static struct cftype kmem_cgroup_files[] = {
> + {
> +  .name = "kmem.limit_in_bytes",
> +  .private = MEMFILE_PRIVATE(_KMEM, RES_LIMIT),
> +  .write_string = mem_cgroup_write,
> +  .read = mem_cgroup_read,
> + },
> + {
> +  .name = "kmem.usage_in_bytes",
> +  .private = MEMFILE_PRIVATE(_KMEM, RES_USAGE),
> +  .read = mem_cgroup_read,
> + },
> + {
> +  .name = "kmem.failcnt",
> +  .private = MEMFILE_PRIVATE(_KMEM, RES_FAILCNT),
> +  .trigger = mem_cgroup_reset,
> +  .read = mem_cgroup_read,
> + },
> + {
> +  .name = "kmem.max_usage_in_bytes",
> +  .private = MEMFILE_PRIVATE(_KMEM, RES_MAX_USAGE),
> +  .trigger = mem_cgroup_reset,
> +  .read = mem_cgroup_read,
> + },
> + {},
> +};
> +
>    static int memcg_init_kmem(struct mem_cgroup *memcg, struct cgroup_subsys *ss)
>    {
>     return mem_cgroup_sockets_init(memcg, ss);
> @@ -5015,6 +5074,12 @@ mem_cgroup_create(struct cgroup *cont)
>      int cpu;
>      enable_swap_cgroup();
>      parent = NULL;
> +
> +#ifdef CONFIG_MEMCG_KMEM
> +  WARN_ON(cgroup_add_cftypes(&mem_cgroup_subsys,
> +         kmem_cgroup_files));
> +#endif
```

```
> +
>     if (mem_cgroup_soft_limit_tree_init())
>      goto free_out;
>     root_mem_cgroup = memcg;
> @@ -5033,6 +5098,7 @@ mem_cgroup_create(struct cgroup *cont)
>    if (parent && parent->use_hierarchy) {
>     res_counter_init(&memcg->res, &parent->res);
>     res_counter_init(&memcg->memsw, &parent->memsw);
> +  res_counter_init(&memcg->kmem, &parent->kmem);
>     /*
>      * We increment refcnt of the parent to ensure that we can
>      * safely access it on res_counter_charge/uncharge.
> @@ -5043,6 +5109,7 @@ mem_cgroup_create(struct cgroup *cont)
>    } else {
>     res_counter_init(&memcg->res, NULL);
>     res_counter_init(&memcg->memsw, NULL);
> +  res_counter_init(&memcg->kmem, NULL);
>    }
>    memcg->last_scanned_node = MAX_NUMNODES;
>    INIT_LIST_HEAD(&memcg->oom_notify);
>
```