Subject: Re: [PATCH 23/29] memory controller memory accounting v7
Posted by Balbir Singh on Thu, 13 Sep 2007 09:49:01 GMT
View Forum Message <> Reply to Message

Peter Zijlstra wrote:

```
>> From: Balbir Singh <balbir@linux.vnet.ibm.com>
>
>>  void page_assign_page_cgroup(struct page *page, struct page_cgroup *pc)
>>  {
>> - page->page_cgroup = (unsigned long)pc;
>> + int locked;
>> +
>> + /*
>> +  * While resetting the page_cgroup we might not hold the
>> +  * page_cgroup lock. free_hot_cold_page() is an example
>> +  * of such a scenario
>> +  */
>> + if (pc)
>> +  VM_BUG_ON(!page_cgroup_locked(page));
>> + locked = (page->page_cgroup & PAGE_CGROUP_LOCK);
>> + page->page_cgroup = ((unsigned long)pc | locked);
>>  }
>
> This looks a bit odd, why not write:
>
>  locked = page_cgroup_locked(page);
>  if (pc)
>    VM_BUG_ON(!locked)
>
```

Sure, we could write it this way or

```
VM_BUG_ON(pc && !locked)
```

```
>> +/*
>> + * Charge the memory controller for page usage.
>> + * Return
>> + * 0 if the charge was successful
>> + * < 0 if the cgroup is over its limit
>> + */
>> +int mem_cgroup_charge(struct page *page, struct mm_struct *mm)
>> +{
>> + struct mem_cgroup *mem;
>> + struct page_cgroup *pc, *race_pc;
>> +
>> + /*
>> +  * Should page_cgroup's go to their own slab?
>> +  * One could optimize the performance of the charging routine
>> +  * by saving a bit in the page_flags and using it as a lock
>> +  * to see if the cgroup page already has a page_cgroup associated
>> +  * with it
>> +  */
>> + lock_page_cgroup(page);
>> + pc = page_get_page_cgroup(page);
>> + /*
>> +  * The page_cgroup exists and the page has already been accounted
>> +  */
>> + if (pc) {
>> +  atomic_inc(&pc->ref_cnt);
>> +  goto done;
>> + }
>> +
>> + unlock_page_cgroup(page);
>> +
>> + pc = kzalloc(sizeof(struct page_cgroup), GFP_KERNEL);
>> + if (pc == NULL)
>> +  goto err;
>> +
>> + rcu_read_lock();
>> + /*
>> +  * We always charge the cgroup the mm_struct belongs to
>> +  * the mm_struct's mem_cgroup changes on task migration if the
>> +  * thread group leader migrates. It's possible that mm is not
>> +  * set, if so charge the init_mm (happens for pagecache usage).
>> +  */
>> + if (!mm)
>> +  mm = &init_mm;
>> +
>> + mem = rcu_dereference(mm->mem_cgroup);
>> + /*
>> +  * For every charge from the cgroup, increment reference
>> +  * count
>> +  */
>> + css_get(&mem->css);
>> + rcu_read_unlock();
>> +
>> + /*
>> +  * If we created the page_cgroup, we should free it on exceeding
>> +  * the cgroup limit.
>> +  */
>> + if (res_counter_charge(&mem->res, 1)) {
>> +  css_put(&mem->css);
>> +  goto free_pc;
>> + }
>> +
>> + lock_page_cgroup(page);
>> + /*
>> +  * Check if somebody else beat us to allocating the page_cgroup
>> +  */
>> + race_pc = page_get_page_cgroup(page);
>> + if (race_pc) {
>> +  kfree(pc);
>> +  pc = race_pc;
>> +  atomic_inc(&pc->ref_cnt);
>
> This inc
>
>> +  res_counter_uncharge(&mem->res, 1);
>> +  css_put(&mem->css);
>> +  goto done;
>> + }
>> +
>> + atomic_set(&pc->ref_cnt, 1);
>
> combined with this set make me wonder...
>
```

I am not sure I understand this comment.

```
>> + pc->mem_cgroup = mem;
>> + pc->page = page;
>> + page_assign_page_cgroup(page, pc);
>> +
>> +done:
>> + unlock_page_cgroup(page);
>> + return 0;
>> +free_pc:
>> + kfree(pc);
>> + return -ENOMEM;
>> +err:
>> + unlock_page_cgroup(page);
>> + return -ENOMEM;
>> +}
>
>
>
>> @@ -2161,6 +2184,9 @@ static int do_anonymous_page(struct mm_s
>>   if (!page)
>>    goto oom;
>>
>> +  if (mem_cgroup_charge(page, mm))
>> +   goto oom_free_page;
>> +
>>   entry = mk_pte(page, vma->vm_page_prot);
>>   entry = maybe_mkwrite(pte_mkdirty(entry), vma);
>>
>
> whitespace damage
>
```

Yes, it's already been fixed in Andrew's tree. Paul could you
please pull in the those fixes as well? They are not in a -mm
tree, but you can find them on mm-commits.

--
 Warm Regards,
 Balbir Singh
 Linux Technology Center
 IBM, ISTL