Memory Resource Controller(Memcg) Implementation Memo

Last Updated: 2010/2

Base Kernel Version: based on 2.6.33-rc7-mm(candidate for 34).

Because VM is getting complex (one of reasons is memcg...), memcg's behavior is complex. This is a document for memcg's internal behavior. Please note that implementation details can be changed.

(*) Topics on API should be in Documentation/admin-guide/cgroup-v1/memory.rst)

0. How to record usage ?

2 objects are used.

page_cgroup ....an object per page.

Allocated at boot or memory hotplug. Freed at memory hot removal.

swap_cgroup ... an entry per swp_entry.

Allocated at swapon(). Freed at swapoff().

The page_cgroup has USED bit and double count against a page_cgroup never occurs. swap_cgroup is used only when a charged page is swapped-out.

1. Charge

a page/swp_entry may be charged (usage += PAGE_SIZE) at

mem_cgroup_try_charge()

2. Uncharge

a page/swp_entry may be uncharged (usage -= PAGE_SIZE) by

mem_cgroup_uncharge()
: Called when a page's refcount goes down to 0.

mem_cgroup_uncharge_swap()
: Called when swp_entry's refcnt goes down to 0. A charge against swap disappears.

3. charge-commit-cancel

Memcg pages are charged in two steps:

- mem_cgroup_try_charge()
- mem_cgroup_commit_charge() or mem_cgroup_cancel_charge()

At try_charge(), there are no flags to say "this page is charged". at this point, usage += PAGE_SIZE.

At commit(), the page is associated with the memcg.

At cancel(), simply usage -= PAGE_SIZE.

Under below explanation, we assume CONFIG_MEM_RES_CTRL_SWAP=y.

4. Anonymous

Anonymous page is newly allocated at

- page fault into MAP_ANONYMOUS mapping.
- Copy-On-Write.

4.1 Swap-in. At swap-in, the page is taken from swap-cache. There are 2 cases.

a. If the SwapCache is newly allocated and read, it has no charges.
b. If the SwapCache has been mapped by processes, it has been charged already.

4.2 Swap-out. At swap-out, typical state transition is below.

a. add to swap cache. (marked as SwapCache) swp_entry's refcnt += 1.
b. fully unmapped. swp_entry's refcnt += # of ptes.
c. write back to swap.
d. delete from swap cache. (remove from SwapCache) swp_entry's refcnt -= 1.

Finally, at task exit, (e) zap_pte() is called and swp_entry's refcnt -=1 -> 0.

5. Page Cache

Page Cache is charged at - add_to_page_cache_locked().

The logic is very clear. (About migration, see below)

Note:
__remove_from_page_cache() is called by remove_from_page_cache() and __remove_mapping().

6. Shmem(tmpfs) Page Cache

The best way to understand shmem's page state transition is to read mm/shmem.c.

But brief explanation of the behavior of memcg around shmem will be helpful to understand the logic.

Shmem's page (just leaf page, not direct/indirect block) can be on

- radix-tree of shmem's inode.
- SwapCache.
- Both on radix-tree and SwapCache. This happens at swap-in and swap-out,

It's charged when...

- A new page is added to shmem's radix-tree.
- A swp page is read. (move a charge from swap_cgroup to page_cgroup)

7. Page Migration

mem_cgroup_migrate()

8. LRU

Each memcg has its own vector of LRUs (inactive anon, active anon, inactive file, active file, unevictable) of pages from each node, each LRU handled under a single lru_lock for that memcg and node.

9. Typical Tests.

Tests for racy cases.

9.1 Small limit to memcg.

When you do test to do racy case, it's good test to set memcg's limit to be very small rather than GB. Many races found in the test under xKB or xxMB limits.

(Memory behavior under GB and Memory behavior under MB shows very different situation.)

9.2 Shmem

Historically, memcg's shmem handling was poor and we saw some amount of troubles here. This is because shmem is page-cache but can be SwapCache. Test with shmem/tmpfs is always good test.

9.3 Migration

For NUMA, migration is an another special case. To do easy test, cpuset is useful. Following is a sample script to do migration:

```
mount -t cgroup -o cpuset none /opt/cpuset

mkdir /opt/cpuset/01
echo 1 > /opt/cpuset/01/cpuset.cpus
echo 0 > /opt/cpuset/01/cpuset.mems
```

```
echo 1 > /opt/cpuset/01/cpuset.memory_migrate
mkdir /opt/cpuset/02
echo 1 > /opt/cpuset/02/cpuset.cpus
echo 1 > /opt/cpuset/02/cpuset.mems
echo 1 > /opt/cpuset/02/cpuset.memory_migrate
```

In above set, when you moves a task from 01 to 02, page migration to node 0 to node 1 will occur. Following is a script to migrate all under cpuset.:

```
--
move_task()
{
for pid in $1
do
        /bin/echo $pid >$2/tasks 2>/dev/null
        echo -n $pid
        echo -n " "
done
echo END
}

G1_TASK=`cat ${G1}/tasks`
G2_TASK=`cat ${G2}/tasks`
move_task "${G1_TASK}" ${G2} &
--
```

9.4 Memory hotplug

memory hotplug test is one of good test.

to offline memory, do following:

```
# echo offline > /sys/devices/system/memory/memoryXXX/state
```

(XXX is the place of memory)

This is an easy way to test page migration, too.

9.5 nested cgroups

Use tests like the following for testing nested cgroups:

```
mkdir /opt/cgroup/01/child_a
mkdir /opt/cgroup/01/child_b

set limit to 01.
add limit to 01/child_b
run jobs under child_a and child_b
```

create/delete following groups at random while jobs are running:

```
/opt/cgroup/01/child_a/child_aa
/opt/cgroup/01/child_b/child_bb
/opt/cgroup/01/child_c
```

running new jobs in new group is also good.

9.6 Mount with other subsystems

Mounting with other subsystems is a good test because there is a race and lock dependency with other cgroup subsystems.

example:

```
# mount -t cgroup none /cgroup -o cpuset,memory,cpu,devices
```

and do task move, mkdir, rmdir etc...under this.

9.7 swapoff

Besides management of swap is one of complicated parts of memcg, call path of swap-in at swapoff is not same as usual swap-in path.. It's worth to be tested explicitly.

For example, test like following is good:

(Shell-A):

```
# mount -t cgroup none /cgroup -o memory
# mkdir /cgroup/test
```

```
# echo 40M > /cgroup/test/memory.limit_in_bytes
# echo 0 > /cgroup/test/tasks
```

Run malloc(100M) program under this. You'll see 60M of swaps.

(Shell-B):

```
# move all tasks in /cgroup/test to /cgroup
# /sbin/swapoff -a
# rmdir /cgroup/test
# kill malloc task.
```

Of course, tmpfs v.s. swapoff test should be tested, too.

9.8 OOM-Killer

Out-of-memory caused by memcg's limit will kill tasks under the memcg. When hierarchy is used, a task under hierarchy will be killed by the kernel.

In this case, panic_on_oom shouldn't be invoked and tasks in other groups shouldn't be killed.

It's not difficult to cause OOM under memcg as following.

Case A) when you can swapoff:

```
#swapoff -a
#echo 50M > /memory.limit_in_bytes
```

run 51M of malloc

Case B) when you use mem+swap limitation:

```
#echo 50M > memory.limit_in_bytes
#echo 50M > memory.memsw.limit_in_bytes
```

run 51M of malloc

9.9 Move charges at task migration

Charges associated with a task can be moved along with task migration.

(Shell-A):

```
#mkdir /cgroup/A
#echo $$ >/cgroup/A/tasks
```

run some programs which uses some amount of memory in /cgroup/A.

(Shell-B):

```
#mkdir /cgroup/B
#echo 1 >/cgroup/B/memory.move_charge_at_immigrate
#echo "pid of the program running in group A" >/cgroup/B/tasks
```

You can see charges have been moved by reading `*.usage_in_bytes` or memory.stat of both A and B.

See 8.2 of Documentation/admin-guide/cgroup-v1/memory.rst to see what value should be written to move_charge_at_immigrate.

9.10 Memory thresholds

Memory controller implements memory thresholds using cgroups notification API. You can use tools/cgroup/cgroup_event_listener.c to test it.

(Shell-A) Create cgroup and run event listener:

```
# mkdir /cgroup/A
# ./cgroup_event_listener /cgroup/A/memory.usage_in_bytes 5M
```

(Shell-B) Add task to cgroup and try to allocate and free memory:

```
# echo $$ >/cgroup/A/tasks
# a="$(dd if=/dev/zero bs=1M count=10)"
# a=
```

You will see message from cgroup_event_listener every time you cross the thresholds.

Use /cgroup/A/memory.memsw.usage_in_bytes to test memsw thresholds.

It's good idea to test root cgroup as well.