

---

Subject: [PATCH v2 01/13] memcg: Consolidate various flags into a single flags field.

Posted by [Suleiman Souhlal](#) on Fri, 09 Mar 2012 20:39:04 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Since there is an ever-increasing number of flags in the memcg struct, consolidate them into a single flags field.

The flags that we consolidate are:

- use\_hierarchy
- memsw\_is\_minimum
- oom\_kill\_disable

Signed-off-by: Suleiman Souhlal <suleiman@google.com>

---

mm/memcontrol.c | 112 ++++++++++++++++++++++++++++++++++++++-----  
1 files changed, 78 insertions(+), 34 deletions(-)

diff --git a/mm/memcontrol.c b/mm/memcontrol.c

index 5585dc3..37ad2cb 100644

--- a/mm/memcontrol.c

+++ b/mm/memcontrol.c

@@ -213,6 +213,15 @@ struct mem\_cgroup\_eventfd\_list {  
static void mem\_cgroup\_threshold(struct mem\_cgroup \*memcg);  
static void mem\_cgroup\_oom\_notify(struct mem\_cgroup \*memcg);

+enum memcg\_flags {

+ MEMCG\_USE\_HIERARCHY, /\*

+ \* Should the accounting and control be

+ \* hierarchical, per subtree?

+ \*/

+ MEMCG\_MEMSW\_IS\_MINIMUM, /\* Set when res.limit == memsw.limit \*/

+ MEMCG\_OOM\_KILL\_DISABLE, /\* OOM-Killer disable \*/

+};

+

/\*

\* The memory controller data structure. The memory controller controls both  
\* page cache and RSS per cgroup. We would eventually like to provide

@@ -245,10 +254,7 @@ struct mem\_cgroup {

atomic\_t numainfo\_events;

atomic\_t numainfo\_updating;

#endif

- /\*

- \* Should the accounting and control be hierarchical, per subtree?

- \*/

- bool use\_hierarchy;

+ unsigned long flags;

bool oom\_lock;

```

    atomic_t under_oom;
@@ -256,11 +262,6 @@ struct mem_cgroup {
    atomic_t refcnt;

    int swappiness;
- /* OOM-Killer disable */
- int oom_kill_disable;
-
- /* set when res.limit == memsw.limit */
- bool memsw_is_minimum;

    /* protect arrays of thresholds */
    struct mutex thresholds_lock;
@@ -371,6 +372,24 @@ enum charge_type {
static void mem_cgroup_get(struct mem_cgroup *memcg);
static void mem_cgroup_put(struct mem_cgroup *memcg);

+static inline bool
+mem_cgroup_test_flag(const struct mem_cgroup *memcg, enum memcg_flags flag)
+{
+ return test_bit(flag, &memcg->flags);
+}
+
+static inline void
+mem_cgroup_set_flag(struct mem_cgroup *memcg, enum memcg_flags flag)
+{
+ set_bit(flag, &memcg->flags);
+}
+
+static inline void
+mem_cgroup_clear_flag(struct mem_cgroup *memcg, enum memcg_flags flag)
+{
+ clear_bit(flag, &memcg->flags);
+}
+
    /* Writing them here to avoid exposing memcg's inner layout */
    #ifdef CONFIG_CGROUP_MEM_RES_CTLR_KMEM
    #include <net/sock.h>
@@ -876,7 +895,8 @@ struct mem_cgroup *mem_cgroup_iter(struct mem_cgroup *root,
    if (prev && prev != root)
        css_put(&prev->css);

- if (!root->use_hierarchy && root != root_mem_cgroup) {
+ if (!mem_cgroup_test_flag(root, MEMCG_USE_HIERARCHY) && root !=
+     root_mem_cgroup) {
    if (prev)
        return NULL;
    return root;

```

```

@@ -1126,8 +1146,8 @@ static bool mem_cgroup_same_or_subtree(const struct mem_cgroup
*root_memcg,
    struct mem_cgroup *memcg)
{
    if (root_memcg != memcg) {
- return (root_memcg->use_hierarchy &&
- css_is_ancestor(&memcg->css, &root_memcg->css));
+ return mem_cgroup_test_flag(root_memcg, MEMCG_USE_HIERARCHY) &&
+ css_is_ancestor(&memcg->css, &root_memcg->css);
    }

    return true;
}
@@ -1460,7 +1480,8 @@ static unsigned long mem_cgroup_reclaim(struct mem_cgroup
*memcg,

    if (flags & MEM_CGROUP_RECLAIM_NOSWAP)
        noswap = true;
- if (!(flags & MEM_CGROUP_RECLAIM_SHRINK) && memcg->memsw_is_minimum)
+ if (!(flags & MEM_CGROUP_RECLAIM_SHRINK) && mem_cgroup_test_flag(memcg,
+ MEMCG_MEMSW_IS_MINIMUM))
        noswap = true;

    for (loop = 0; loop < MEM_CGROUP_MAX_RECLAIM_LOOPS; loop++) {
}
@@ -1813,7 +1834,7 @@ bool mem_cgroup_handle_oom(struct mem_cgroup *memcg, gfp_t
mask)
    * under OOM is always welcomed, use TASK_KILLABLE here.
    */
    prepare_to_wait(&memcg_oom_waitq, &owait.wait, TASK_KILLABLE);
- if (!locked || memcg->oom_kill_disable)
+ if (!locked || mem_cgroup_test_flag(memcg, MEMCG_OOM_KILL_DISABLE))
        need_to_kill = false;
    if (locked)
        mem_cgroup_oom_notify(memcg);
}
@@ -3416,9 +3437,11 @@ static int mem_cgroup_resize_limit(struct mem_cgroup *memcg,
ret = res_counter_set_limit(&memcg->res, val);
    if (!ret) {
        if (memswlimit == val)
- memcg->memsw_is_minimum = true;
+ mem_cgroup_set_flag(memcg,
+ MEMCG_MEMSW_IS_MINIMUM);
        else
- memcg->memsw_is_minimum = false;
+ mem_cgroup_clear_flag(memcg,
+ MEMCG_MEMSW_IS_MINIMUM);
    }
    mutex_unlock(&set_limit_mutex);
}
@@ -3475,9 +3498,11 @@ static int mem_cgroup_resize_memsw_limit(struct mem_cgroup

```

```

*memcg,
    ret = res_counter_set_limit(&memcg->memsw, val);
    if (!ret) {
        if (memlimit == val)
- memcg->memsw_is_minimum = true;
+ mem_cgroup_set_flag(memcg,
+     MEMCG_MEMSW_IS_MINIMUM);
        else
- memcg->memsw_is_minimum = false;
+ mem_cgroup_clear_flag(memcg,
+     MEMCG_MEMSW_IS_MINIMUM);
    }
    mutex_unlock(&set_limit_mutex);

```

@@ -3745,7 +3770,8 @@ int mem\_cgroup\_force\_empty\_write(struct cgroup \*cont, unsigned int event)

```

static u64 mem_cgroup_hierarchy_read(struct cgroup *cont, struct cftype *cft)
{
- return mem_cgroup_from_cont(cont)->use_hierarchy;
+ return mem_cgroup_test_flag(mem_cgroup_from_cont(cont),
+     MEMCG_USE_HIERARCHY);
}

```

static int mem\_cgroup\_hierarchy\_write(struct cgroup \*cont, struct cftype \*cft,  
@@ -3768,10 +3794,14 @@ static int mem\_cgroup\_hierarchy\_write(struct cgroup \*cont, struct cftype \*cft,

```

    * For the root cgroup, parent_mem is NULL, we allow value to be
    * set if there are no children.
    */
- if ((!parent_memcg || !parent_memcg->use_hierarchy) &&
-     (val == 1 || val == 0)) {
+ if ((!parent_memcg || !mem_cgroup_test_flag(parent_memcg,
+     MEMCG_USE_HIERARCHY)) && (val == 1 || val == 0)) {
    if (list_empty(&cont->children))
- memcg->use_hierarchy = val;
+ if (val)
+ mem_cgroup_set_flag(memcg, MEMCG_USE_HIERARCHY);
+ else
+ mem_cgroup_clear_flag(memcg,
+     MEMCG_USE_HIERARCHY);
    else
        retval = -EBUSY;
    } else
@@ -3903,13 +3933,13 @@ static void memcg_get_hierarchical_limit(struct mem_cgroup
*memcg,
    min_limit = res_counter_read_u64(&memcg->res, RES_LIMIT);
    min_memsw_limit = res_counter_read_u64(&memcg->memsw, RES_LIMIT);

```

```

    cgroup = memcg->css.cgroup;
- if (!memcg->use_hierarchy)
+ if (!mem_cgroup_test_flag(memcg, MEMCG_USE_HIERARCHY))
    goto out;

    while (cgroup->parent) {
        cgroup = cgroup->parent;
        memcg = mem_cgroup_from_cont(cgroup);
- if (!memcg->use_hierarchy)
+ if (!mem_cgroup_test_flag(memcg, MEMCG_USE_HIERARCHY))
        break;
        tmp = res_counter_read_u64(&memcg->res, RES_LIMIT);
        min_limit = min(min_limit, tmp);
@@ -4206,8 +4236,9 @@ static int mem_cgroup_swappiness_write(struct cgroup *cgrp, struct
cftype *cft,
    cgroup_lock();

    /* If under hierarchy, only empty-root can set this value */
- if ((parent->use_hierarchy) ||
-     (memcg->use_hierarchy && !list_empty(&cgrp->children))) {
+ if (mem_cgroup_test_flag(parent, MEMCG_USE_HIERARCHY) ||
+     (mem_cgroup_test_flag(memcg, MEMCG_USE_HIERARCHY) &&
+      !list_empty(&cgrp->children))) {
        cgroup_unlock();
        return -EINVAL;
    }
@@ -4518,7 +4549,8 @@ static int mem_cgroup_oom_control_read(struct cgroup *cgrp,
{
    struct mem_cgroup *memcg = mem_cgroup_from_cont(cgrp);

- cb->fill(cb, "oom_kill_disable", memcg->oom_kill_disable);
+ cb->fill(cb, "oom_kill_disable", mem_cgroup_test_flag(memcg,
+ MEMCG_OOM_KILL_DISABLE));

    if (atomic_read(&memcg->under_oom))
        cb->fill(cb, "under_oom", 1);
@@ -4541,14 +4573,18 @@ static int mem_cgroup_oom_control_write(struct cgroup *cgrp,

    cgroup_lock();
    /* oom-kill-disable is a flag for subhierarchy. */
- if ((parent->use_hierarchy) ||
-     (memcg->use_hierarchy && !list_empty(&cgrp->children))) {
+ if (mem_cgroup_test_flag(parent, MEMCG_USE_HIERARCHY) ||
+     (mem_cgroup_test_flag(memcg, MEMCG_USE_HIERARCHY) &&
+      !list_empty(&cgrp->children))) {
        cgroup_unlock();
        return -EINVAL;
    }
}

```

```

- memcg->oom_kill_disable = val;
- if (!val)
+ if (val)
+ mem_cgroup_set_flag(memcg, MEMCG_OOM_KILL_DISABLE);
+ else {
+ mem_cgroup_clear_flag(memcg, MEMCG_OOM_KILL_DISABLE);
  memcg_oom_recover(memcg);
+ }
  cgroup_unlock();
  return 0;
}
@@ -4916,11 +4952,19 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
  hotcpu_notifier(memcg_cpu_hotplug_callback, 0);
} else {
  parent = mem_cgroup_from_cont(cont->parent);
- memcg->use_hierarchy = parent->use_hierarchy;
- memcg->oom_kill_disable = parent->oom_kill_disable;
+
+ if (mem_cgroup_test_flag(parent, MEMCG_USE_HIERARCHY))
+ mem_cgroup_set_flag(memcg, MEMCG_USE_HIERARCHY);
+ else
+ mem_cgroup_clear_flag(memcg, MEMCG_USE_HIERARCHY);
+
+ if (mem_cgroup_test_flag(parent, MEMCG_OOM_KILL_DISABLE))
+ mem_cgroup_set_flag(memcg, MEMCG_OOM_KILL_DISABLE);
+ else
+ mem_cgroup_clear_flag(memcg, MEMCG_OOM_KILL_DISABLE);
}

- if (parent && parent->use_hierarchy) {
+ if (parent && mem_cgroup_test_flag(parent, MEMCG_USE_HIERARCHY)) {
  res_counter_init(&memcg->res, &parent->res);
  res_counter_init(&memcg->memsw, &parent->memsw);
  /*

```

### 1.7.7.3