Subject: [PATCH -mm 0/5] swapcgroup (v3)
Posted by Daisuke Nishimura on Fri, 04 Jul 2008 06:15:36 GMT
View Forum Message <> Reply to Message

Hi.

This is new version of swapcgroup.

Major changes from previous version
- Rebased on 2.6.26-rc5-mm3.
  The new -mm has been released, but these patches
  can be applied on 2.6.26-rc8-mm1 too with only some offset warnings.
  I tested these patches on 2.6.26-rc5-mm3 with some fixes about memory,
  and it seems to work fine.
- (NEW) Implemented force_empty.
  Currently, it simply uncharges all the charges from the group.

Patches
- [1/5] add cgroup files
- [2/5] add a member to swap_info_struct
- [3/5] implement charge and uncharge
- [4/5] modify vm_swap_full()
- [5/5] implement force_empty

ToDo(in my thought. Feel free to add some others here.)
- need some documentation
  Add to memory.txt? or create a new documentation file?

- add option to disable only this feature
  I'm wondering if this option is needed.
  memcg has already the boot option to disable it.
  Is there any case where memory should be accounted but swap should not?

- hierarchy support
- move charges along with task
  Both of them need more discussion.

Thanks,
Daisuke Nishimura.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH -mm 1/5] swapcgroup (v3): add cgroup files

Even if limiting memory usage by memory cgroup,
swap space is shared, so resource isolation is not enough.
If one group uses up most of the swap space, it can affect
other groups anyway.

The purpose of swapcgroup is limiting swap usage per group
as memory cgroup limits the RSS memory usage.
It's now implemented as a add-on to memory cgroup.

This patch adds cgroup files(and a member to struct mem_cgroup)
for swapcgroup.

Files to be added by this patch are:

- memory.swap_usage_in_bytes
- memory.swap_max_usage_in_bytes
- memory.swap_limit_in_bytes
- memory.swap_failcnt

The meaning of those files are the same as memory cgroup.

Change log
v2->v3
- Rebased on 2.6.26-rc5-mm3.
v1->v2
- Rebased on 2.6.26-rc2-mm1.
- Implemented as a add-on to memory cgroup.

Signed-off-by: Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp>

---
 init/Kconfig | 7 +++++
 mm/memcontrol.c | 67 ++
 2 files changed, 74 insertions(+), 0 deletions(-)

diff --git a/init/Kconfig b/init/Kconfig
index 847931a..c604b6d 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -418,6 +418,13 @@ config CGROUP_MEMRLIMIT_CTLR
    memory RSS and Page Cache control. Virtual address space control
    is provided by this controller.

```
+config CGROUP_SWAP_RES_CTLR
+ bool "Swap Resource Controller for Control Groups"
+ depends on CGROUP_MEM_RES_CTLR && SWAP
+ help
+   Provides a swap resource controller that manages and limits swap usage.
+   Implemented as a add-on to Memory Resource Controller.
+
 config SYSFS_DEPRECATED
  bool

diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index b8fe33c..ddc842b 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -133,6 +133,12 @@ struct mem_cgroup {
   * statistics.
   */
  struct mem_cgroup_stat stat;
+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+ /*
+  * the counter to account for swap usage
+  */
+ struct res_counter swap_res;
+#endif
 };
 static struct mem_cgroup init_mem_cgroup;

@@ -953,6 +959,39 @@ static int mem_control_stat_show(struct cgroup *cont, struct cftype *cft,
  return 0;
 }

+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+static u64 swap_cgroup_read(struct cgroup *cont, struct cftype *cft)
+{
+ return res_counter_read_u64(&mem_cgroup_from_cont(cont)->swap_res,
+        cft->private);
+}
+
+static ssize_t swap_cgroup_write(struct cgroup *cont, struct cftype *cft,
+    struct file *file, const char __user *userbuf,
+    size_t nbytes, loff_t *ppos)
+{
+ return res_counter_write(&mem_cgroup_from_cont(cont)->swap_res,
+    cft->private, userbuf, nbytes, ppos,
+    mem_cgroup_write_strategy);
+}
+
+static int swap_cgroup_reset(struct cgroup *cont, unsigned int event)
```

```
+{
+ struct mem_cgroup *mem;
+
+ mem = mem_cgroup_from_cont(cont);
+ switch (event) {
+ case RES_MAX_USAGE:
+  res_counter_reset_max(&mem->swap_res);
+  break;
+ case RES_FAILCNT:
+  res_counter_reset_failcnt(&mem->swap_res);
+  break;
+ }
+ return 0;
+}
+#endif
+
 static struct cftype mem_cgroup_files[] = {
  {
   .name = "usage_in_bytes",
@@ -985,6 +1024,31 @@ static struct cftype mem_cgroup_files[] = {
   .name = "stat",
   .read_map = mem_control_stat_show,
  },
+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+ {
+  .name = "swap_usage_in_bytes",
+  .private = RES_USAGE,
+  .read_u64 = swap_cgroup_read,
+ },
+ {
+  .name = "swap_max_usage_in_bytes",
+  .private = RES_MAX_USAGE,
+  .trigger = swap_cgroup_reset,
+  .read_u64 = swap_cgroup_read,
+ },
+ {
+  .name = "swap_limit_in_bytes",
+  .private = RES_LIMIT,
+  .write = swap_cgroup_write,
+  .read_u64 = swap_cgroup_read,
+ },
+ {
+  .name = "swap_failcnt",
+  .private = RES_FAILCNT,
+  .trigger = swap_cgroup_reset,
+  .read_u64 = swap_cgroup_read,
+ },
+#endif
```

};

static int alloc_mem_cgroup_per_zone_info(struct mem_cgroup *mem, int node)
@@ -1063,6 +1127,9 @@ mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
}

res_counter_init(&mem->res);
+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+ res_counter_init(&mem->swap_res);
+#endif

for_each_node_state(node, N_POSSIBLE)
if (alloc_mem_cgroup_per_zone_info(mem, node))

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH -mm 2/5] swapcgroup (v3): add a member to swap_info_struct
Posted by Daisuke Nishimura on Fri, 04 Jul 2008 06:18:51 GMT
View Forum Message <> Reply to Message

This patch add a member to swap_info_struct for cgroup.

This member, array of pointers to mem_cgroup, is used to
remember to which cgroup each swap entries are charged.

The memory for this array of pointers is allocated on swapon,
and freed on swapoff.

Change log
v2->v3
- Rebased on 2.6.26-rc5-mm3
- add helper functions and removed #ifdef from sys_swapon()/sys_swapoff().
- add check on mem_cgroup_subsys.disabled
v1->v2
- Rebased on 2.6.26-rc2-mm1
- Implemented as a add-on to memory cgroup.

Signed-off-by: Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp>

---
include/linux/memcontrol.h | 20 ++++++++++++++++++-
include/linux/swap.h | 3 +++
mm/memcontrol.c | 36 ++++++++++++++++++++++++++++++++++++

```
 mm/swapfile.c             |   11 ++++++++++
 4 files changed, 69 insertions(+), 1 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index ee1b2fc..b6ff509 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -24,6 +24,7 @@ struct mem_cgroup;
 struct page_cgroup;
 struct page;
 struct mm_struct;
+struct swap_info_struct;

 #ifdef CONFIG_CGROUP_MEM_RES_CTLR

@@ -165,5 +166,22 @@ static inline long mem_cgroup_calc_reclaim(struct mem_cgroup *mem,
 }
 #endif /* CONFIG_CGROUP_MEM_CONT */

-#endif /* _LINUX_MEMCONTROL_H */
+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+extern struct mem_cgroup **swap_info_clear_memcg(struct swap_info_struct *p);
+extern int swap_info_alloc_memcg(struct swap_info_struct *p,
+    unsigned long maxpages);
+#else
+static inline
+struct mem_cgroup **swap_info_clear_memcg(struct swap_info_struct *p)
+{
+ return NULL;
+}

+static inline
+int swap_info_alloc_memcg(struct swap_info_struct *p, unsigned long maxpages)
+{
+ return 0;
+}
+#endif
+
+#endif /* _LINUX_MEMCONTROL_H */
diff --git a/include/linux/swap.h b/include/linux/swap.h
index a3af95b..6e1b03d 100644
--- a/include/linux/swap.h
+++ b/include/linux/swap.h
@@ -142,6 +142,9 @@ struct swap_info_struct {
  struct swap_extent *curr_swap_extent;
  unsigned old_block_size;
  unsigned short * swap_map;
+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
```

```
+ struct mem_cgroup **memcg;
+#endif
  unsigned int lowest_bit;
  unsigned int highest_bit;
  unsigned int cluster_next;
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index ddc842b..81bb7fa 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -1209,3 +1209,39 @@ struct cgroup_subsys mem_cgroup_subsys = {
  .attach = mem_cgroup_move_task,
  .early_init = 0,
 };
+
+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+/* called with swap_lock held */
+struct mem_cgroup **swap_info_clear_memcg(struct swap_info_struct *p)
+{
+ struct mem_cgroup **mem;
+
+ /* just clear p->memcg, without checking mem_cgroup_subsys.disabled */
+ mem = p->memcg;
+ p->memcg = NULL;
+
+ return mem;
+}
+
+/* called without swap_lock held */
+int swap_info_alloc_memcg(struct swap_info_struct *p, unsigned long maxpages)
+{
+ int ret = 0;
+
+ if (mem_cgroup_subsys.disabled)
+  goto out;
+
+ p->memcg = vmalloc(maxpages * sizeof(struct mem_cgroup *));
+ if (!p->memcg) {
+  /* make swapon fail */
+  printk(KERN_ERR "Unable to allocate memory for memcg\n");
+  ret = -ENOMEM;
+  goto out;
+ }
+ memset(p->memcg, 0, maxpages * sizeof(struct mem_cgroup *));
+
+out:
+ return ret;
+}
+#endif
```

```
+
diff --git a/mm/swapfile.c b/mm/swapfile.c
index bf7d13d..312c573 100644
--- a/mm/swapfile.c
+++ b/mm/swapfile.c
@@ -1228,6 +1228,7 @@ asmlinkage long sys_swapoff(const char __user * specialfile)
  unsigned short *swap_map;
  struct file *swap_file, *victim;
  struct address_space *mapping;
+ struct mem_cgroup **memcg = NULL;
  struct inode *inode;
  char * pathname;
  int i, type, prev;
@@ -1328,10 +1329,12 @@ asmlinkage long sys_swapoff(const char __user * specialfile)
  p->max = 0;
  swap_map = p->swap_map;
  p->swap_map = NULL;
+ memcg = swap_info_clear_memcg(p);
  p->flags = 0;
  spin_unlock(&swap_lock);
  mutex_unlock(&swapon_mutex);
  vfree(swap_map);
+ vfree(memcg);
  inode = mapping->host;
  if (S_ISBLK(inode->i_mode)) {
   struct block_device *bdev = I_BDEV(inode);
@@ -1475,6 +1478,7 @@ asmlinkage long sys_swapon(const char __user * specialfile, int
swap_flags)
  unsigned long maxpages = 1;
  int swapfilesize;
  unsigned short *swap_map;
+ struct mem_cgroup **memcg = NULL;
  struct page *page = NULL;
  struct inode *inode = NULL;
  int did_down = 0;
@@ -1498,6 +1502,7 @@ asmlinkage long sys_swapon(const char __user * specialfile, int
swap_flags)
  p->swap_file = NULL;
  p->old_block_size = 0;
  p->swap_map = NULL;
+ swap_info_clear_memcg(p);
  p->lowest_bit = 0;
  p->highest_bit = 0;
  p->cluster_nr = 0;
@@ -1670,6 +1675,10 @@ asmlinkage long sys_swapon(const char __user * specialfile, int
swap_flags)
     1 /* header page */;
   if (error)
```

```
    goto bad_swap;
+
+  error = swap_info_alloc_memcg(p, maxpages);
+  if (error)
+   goto bad_swap;
  }

  if (nr_good_pages) {
@@ -1729,11 +1738,13 @@ bad_swap_2:
  swap_map = p->swap_map;
  p->swap_file = NULL;
  p->swap_map = NULL;
+ memcg = swap_info_clear_memcg(p);
  p->flags = 0;
  if (!(swap_flags & SWAP_FLAG_PREFER))
   ++least_priority;
  spin_unlock(&swap_lock);
  vfree(swap_map);
+ vfree(memcg);
  if (swap_file)
   filp_close(swap_file, NULL);
 out:
```

Subject: [PATCH -mm 3/5] swapcgroup (v3): implement charge and uncharge
Posted by Daisuke Nishimura on Fri, 04 Jul 2008 06:20:14 GMT
View Forum Message <> Reply to Message

This patch implements charge and uncharge of swapcgroup.

- what will be charged ?
  charge the number of swap entries in bytes.

- when to charge/uncharge ?
  charge at get_swap_entry(), and uncharge at swap_entry_free().

- to what group charge the swap entry ?
  To determine to what mem_cgroup the swap entry should be charged,
  I changed the argument of get_swap_entry() from (void) to
  (struct page *). As a result, get_swap_entry() can determine
  to what mem_cgroup it should charge the swap entry
  by referring to page->page_cgroup->mem_cgroup.

- from what group uncharge the swap entry ?

I added in previous patch to swap_info_struct a member
'struct swap_cgroup **', array of pointer to which swap_cgroup
the swap entry is charged.

Change log
v2->v3
- Rebased on 2.6.26-rc5-mm3
- make swap_cgroup_charge() fail when !pc.
- add check on mem_cgroup_subsys.disabled
v1->v2
- Rebased on 2.6.26-rc2-mm1
- Implemented as a add-on to memory cgroup.

Signed-off-by: Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp>

---
include/linux/memcontrol.h | 17 +++++++++++++++
include/linux/swap.h | 4 +-
mm/memcontrol.c | 53 ++
mm/shmem.c | 2 +-
mm/swap_state.c | 2 +-
mm/swapfile.c | 13 ++++++++++-
6 files changed, 86 insertions(+), 5 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index b6ff509..f3c84f7 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -170,6 +170,11 @@ static inline long mem_cgroup_calc_reclaim(struct mem_cgroup *mem,
extern struct mem_cgroup **swap_info_clear_memcg(struct swap_info_struct *p);
extern int swap_info_alloc_memcg(struct swap_info_struct *p,
unsigned long maxpages);
+extern int swap_cgroup_charge(struct page *page,
+ struct swap_info_struct *si,
+ unsigned long offset);
+extern void swap_cgroup_uncharge(struct swap_info_struct *si,
+ unsigned long offset);
#else
static inline
struct mem_cgroup **swap_info_clear_memcg(struct swap_info_struct *p)
@@ -182,6 +187,18 @@ int swap_info_alloc_memcg(struct swap_info_struct *p, unsigned long
maxpages)
{
return 0;
}
+
+

```
+static inline int swap_cgroup_charge(struct page *page,
+   struct swap_info_struct *si,
+   unsigned long offset)
+{
+ return 0;
+}
+
+static inline void swap_cgroup_uncharge(struct swap_info_struct *si,
+    unsigned long offset)
+{
+}
 #endif

 #endif /* _LINUX_MEMCONTROL_H */
diff --git a/include/linux/swap.h b/include/linux/swap.h
index 6e1b03d..f80255b 100644
--- a/include/linux/swap.h
+++ b/include/linux/swap.h
@@ -298,7 +298,7 @@ extern struct page *swapin_readahead(swp_entry_t, gfp_t,
 /* linux/mm/swapfile.c */
 extern long total_swap_pages;
 extern void si_swapinfo(struct sysinfo *);
-extern swp_entry_t get_swap_page(void);
+extern swp_entry_t get_swap_page(struct page *);
 extern swp_entry_t get_swap_page_of_type(int);
 extern int swap_duplicate(swp_entry_t);
 extern int valid_swaphandles(swp_entry_t, unsigned long *);
@@ -405,7 +405,7 @@ static inline int remove_exclusive_swap_page_ref(struct page *page)
  return 0;
 }

-static inline swp_entry_t get_swap_page(void)
+static inline swp_entry_t get_swap_page(struct page *page)
 {
  swp_entry_t entry;
  entry.val = 0;
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 81bb7fa..d16d0a5 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -1243,5 +1243,58 @@ int swap_info_alloc_memcg(struct swap_info_struct *p, unsigned
long maxpages)
 out:
  return ret;
 }
+
+int swap_cgroup_charge(struct page *page,
+   struct swap_info_struct *si,
```

```
+   unsigned long offset)
+{
+ int ret;
+ struct page_cgroup *pc;
+ struct mem_cgroup *mem;
+
+ if (mem_cgroup_subsys.disabled)
+  return 0;
+
+ lock_page_cgroup(page);
+ pc = page_get_page_cgroup(page);
+ if (unlikely(!pc)) {
+  /* make get_swap_page fail */
+  unlock_page_cgroup(page);
+  return -ENOMEM;
+ } else {
+  mem = pc->mem_cgroup;
+  css_get(&mem->css);
+ }
+ unlock_page_cgroup(page);
+
+ ret = res_counter_charge(&mem->swap_res, PAGE_SIZE);
+ if (!ret)
+  si->memcg[offset] = mem;
+ else
+  css_put(&mem->css);
+
+ return ret;
+}
+
+void swap_cgroup_uncharge(struct swap_info_struct *si,
+    unsigned long offset)
+{
+ struct mem_cgroup *mem = si->memcg[offset];
+
+ if (mem_cgroup_subsys.disabled)
+  return;
+
+ /* "mem" would be NULL:
+  * 1. when get_swap_page() failed at charging swap_cgroup,
+  *    and called swap_entry_free().
+  * 2. when this swap entry had been assigned by
+  *    get_swap_page_of_type() (via SWSUSP?).
+  */
+ if (mem) {
+  res_counter_uncharge(&mem->swap_res, PAGE_SIZE);
+  si->memcg[offset] = NULL;
+  css_put(&mem->css);
```

```
+ }
+}
 #endif

diff --git a/mm/shmem.c b/mm/shmem.c
index bed732c..958e041 100644
--- a/mm/shmem.c
+++ b/mm/shmem.c
@@ -1029,7 +1029,7 @@ static int shmem_writepage(struct page *page, struct
writeback_control *wbc)
   * want to check if there's a redundant swappage to be discarded.
   */
  if (wbc->for_reclaim)
-  swap = get_swap_page();
+  swap = get_swap_page(page);
  else
   swap.val = 0;

diff --git a/mm/swap_state.c b/mm/swap_state.c
index ca43e64..ad5d85c 100644
--- a/mm/swap_state.c
+++ b/mm/swap_state.c
@@ -138,7 +138,7 @@ int add_to_swap(struct page * page, gfp_t gfp_mask)
  BUG_ON(!PageUptodate(page));

  for (;;) {
-  entry = get_swap_page();
+  entry = get_swap_page(page);
   if (!entry.val)
    return 0;

diff --git a/mm/swapfile.c b/mm/swapfile.c
index 312c573..57798c5 100644
--- a/mm/swapfile.c
+++ b/mm/swapfile.c
@@ -172,7 +172,9 @@ no_page:
  return 0;
 }

-swp_entry_t get_swap_page(void)
+/* get_swap_page() calls this */
+static int swap_entry_free(struct swap_info_struct *, unsigned long);
+swp_entry_t get_swap_page(struct page *page)
 {
  struct swap_info_struct *si;
  pgoff_t offset;
@@ -201,6 +203,14 @@ swp_entry_t get_swap_page(void)
   swap_list.next = next;
```

```
   offset = scan_swap_map(si);
   if (offset) {
+   /*
+    * This should be the first use of this swap entry.
+    * So, charge this swap entry here.
+    */
+   if (swap_cgroup_charge(page, si, offset)) {
+    swap_entry_free(si, offset);
+    goto noswap;
+   }
    spin_unlock(&swap_lock);
    return swp_entry(type, offset);
   }
@@ -285,6 +295,7 @@ static int swap_entry_free(struct swap_info_struct *p, unsigned long
offset)
     swap_list.next = p - swap_info;
    nr_swap_pages++;
    p->inuse_pages--;
+   swap_cgroup_uncharge(p, offset);
   }
  }
  return count;
```

Subject: [PATCH -mm 4/5] swapcgroup (v3): modify vm_swap_full()
Posted by Daisuke Nishimura on Fri, 04 Jul 2008 06:22:44 GMT
View Forum Message <> Reply to Message

This patch modifies vm_swap_full() to calculate
the rate of swap usage per cgroup.

The purpose of this change is to free freeable swap caches
(that is, swap entries) per cgroup, so that swap_cgroup_charge()
fails less frequently.

I tested whether this patch can reduce the swap usage or not,
by running qsbench (8 x 40M) 10 times in mem:128M/swap:256M group
and mem:256M/swap:128M group.
And I confirmed that this patch can keep swap usage to
(half of the limit < usage < the limit),
whereas without this patch, swap usage tends to keep near to the limit.

Change log

v2->v3
- Rebased on 2.6.26-rc5-mm3
- take into account global swap usage.
- change arg of vm_swap_full from page to memcg.
- add check on mem_cgroup_subsys.disabled
v1->v2
- new patch

Signed-off-by: Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp>

---
 include/linux/memcontrol.h | 12 ++++++++++
 include/linux/swap.h | 4 +++-
 mm/memcontrol.c | 18 ++++++++++++++++++
 mm/memory.c | 4 +++-
 mm/swapfile.c | 38 ++++++++++++++++++++++++++++++++++++++-
 mm/vmscan.c | 6 +++---
 6 files changed, 76 insertions(+), 6 deletions(-)

diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index f3c84f7..1e6eb0c 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -175,6 +175,8 @@ extern int swap_cgroup_charge(struct page *page,
    unsigned long offset);
 extern void swap_cgroup_uncharge(struct swap_info_struct *si,
     unsigned long offset);
+extern struct mem_cgroup *page_to_memcg(struct page *page);
+extern int swap_cgroup_vm_swap_full(struct mem_cgroup *memcg);
 #else
 static inline
 struct mem_cgroup **swap_info_clear_memcg(struct swap_info_struct *p)
@@ -199,6 +201,16 @@ static inline void swap_cgroup_uncharge(struct swap_info_struct *si,
     unsigned long offset)
 {
 }
+
+static inline struct mem_cgroup *page_to_memcg(struct page *page)
+{
+ return NULL;
+}
+
+static inline int swap_cgroup_vm_swap_full(struct mem_cgroup *memcg)
+{
+ return 0;
+}
 #endif

```
 #endif /* _LINUX_MEMCONTROL_H */
diff --git a/include/linux/swap.h b/include/linux/swap.h
index f80255b..2b95df9 100644
--- a/include/linux/swap.h
+++ b/include/linux/swap.h
@@ -161,7 +161,9 @@ struct swap_list_t {
 };

 /* Swap 50% full? Release swapcache more aggressively.. */
-#define vm_swap_full() (nr_swap_pages*2 < total_swap_pages)
+#define vm_swap_full(memcg) ((nr_swap_pages*2 < total_swap_pages) \
+    || swap_cgroup_vm_swap_full(memcg))
+

 /* linux/mm/page_alloc.c */
 extern unsigned long totalram_pages;
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index d16d0a5..160fca1 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -1296,5 +1296,23 @@ void swap_cgroup_uncharge(struct swap_info_struct *si,
   css_put(&mem->css);
  }
 }
+
+int swap_cgroup_vm_swap_full(struct mem_cgroup *memcg)
+{
+ u64 usage;
+ u64 limit;
+
+ if (memcg)
+  css_get(&memcg->css);
+ else
+  return 0;
+
+ usage = res_counter_read_u64(&memcg->swap_res, RES_USAGE);
+ limit = res_counter_read_u64(&memcg->swap_res, RES_LIMIT);
+
+ css_put(&memcg->css);
+
+ return usage * 2 > limit;
+}
 #endif

diff --git a/mm/memory.c b/mm/memory.c
index 536d748..afcf737 100644
--- a/mm/memory.c
```

```
+++ b/mm/memory.c
@@ -2230,7 +2230,9 @@ static int do_swap_page(struct mm_struct *mm, struct vm_area_struct
*vma,
  page_add_anon_rmap(page, vma, address);

  swap_free(entry);
- if (vm_swap_full() || (vma->vm_flags & VM_LOCKED) || PageMlocked(page))
+ if (vm_swap_full(page_to_memcg(page))
+  || (vma->vm_flags & VM_LOCKED)
+  || PageMlocked(page))
   remove_exclusive_swap_page(page);
  unlock_page(page);

diff --git a/mm/swapfile.c b/mm/swapfile.c
index 57798c5..6a863bc 100644
--- a/mm/swapfile.c
+++ b/mm/swapfile.c
@@ -28,6 +28,7 @@
 #include <linux/capability.h>
 #include <linux/syscalls.h>
 #include <linux/memcontrol.h>
+#include <linux/cgroup.h>

 #include <asm/pgtable.h>
 #include <asm/tlbflush.h>
@@ -447,7 +448,8 @@ void free_swap_and_cache(swp_entry_t entry)
   /* Only cache user (+us), or swap space full? Free it! */
   /* Also recheck PageSwapCache after page is locked (above) */
   if (PageSwapCache(page) && !PageWriteback(page) &&
-      (one_user || vm_swap_full())) {
+      (one_user
+      || vm_swap_full(page_to_memcg(page)))) {
    delete_from_swap_cache(page);
    SetPageDirty(page);
   }
@@ -1889,3 +1891,37 @@ int valid_swaphandles(swp_entry_t entry, unsigned long *offset)
  *offset = ++toff;
  return nr_pages? ++nr_pages: 0;
 }
+
+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+/*
+ * returns mem_cgroup to which the swap cache page is charged as swap.
+ * should be called with the page locked.
+ */
+struct mem_cgroup *page_to_memcg(struct page *page)
+{
+ struct swap_info_struct *p;
```

```
+ struct mem_cgroup *mem = NULL;
+ swp_entry_t entry;
+
+ BUG_ON(!PageLocked(page));
+
+ if (mem_cgroup_subsys.disabled)
+  goto out;
+
+ if (!PageSwapCache(page))
+  goto out;
+
+ entry.val = page_private(page);
+ p = swap_info_get(entry);
+ if (!p)
+  goto out;
+
+ mem = p->memcg[swp_offset(entry)];
+
+ spin_unlock(&swap_lock);
+
+out:
+ return mem;
+}
+#endif
+
diff --git a/mm/vmscan.c b/mm/vmscan.c
index 9a5e423..ac45993 100644
--- a/mm/vmscan.c
+++ b/mm/vmscan.c
@@ -752,7 +752,7 @@ cull_mlocked:

 activate_locked:
   /* Not a candidate for swapping, so reclaim swap space. */
-  if (PageSwapCache(page) && vm_swap_full())
+  if (PageSwapCache(page) && vm_swap_full(page_to_memcg(page)))
    remove_exclusive_swap_page_ref(page);
   VM_BUG_ON(PageActive(page));
   SetPageActive(page);
@@ -1317,7 +1317,7 @@ static void shrink_active_list(unsigned long nr_pages, struct zone
*zone,
    __mod_zone_page_state(zone, NR_LRU_BASE + lru, pgmoved);
    pgmoved = 0;
    spin_unlock_irq(&zone->lru_lock);
-   if (vm_swap_full())
+   if (vm_swap_full(sc->mem_cgroup))
     pagevec_swap_free(&pvec);
    __pagevec_release(&pvec);
    spin_lock_irq(&zone->lru_lock);
```

```
@@ -1328,7 +1328,7 @@ static void shrink_active_list(unsigned long nr_pages, struct zone
*zone,
  __count_zone_vm_events(PGREFILL, zone, pgscanned);
  __count_vm_events(PGDEACTIVATE, pgdeactivate);
  spin_unlock_irq(&zone->lru_lock);
- if (vm_swap_full())
+ if (vm_swap_full(sc->mem_cgroup))
   pagevec_swap_free(&pvec);

  pagevec_release(&pvec);
```

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH -mm 5/5] swapcgroup (v3): implement force_empty
Posted by Daisuke Nishimura on Fri, 04 Jul 2008 06:24:23 GMT
View Forum Message <> Reply to Message

This patch implements force_empty of swapcgroup.

Currently, it simply uncharges all the charges from the group.

I think there can be other implementations.

What I thought are:
- move all the charges to its parent.
- unuse(swap in) all the swap charged to the group.

But in any case, I think before implementing this way,
hierarchy and move charges support are needed.

So I think this is enough for the first step.

Change log
v2->v3
- new patch

Signed-off-by: Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp>

```
---
 include/linux/memcontrol.h |    5 ++++
 mm/memcontrol.c            |   47 +++++++++++++++++++++++++++++++++--------
 mm/swapfile.c              |   52 ++++++++++++++++++++++++++++++++++++++++++++-
 3 files changed, 91 insertions(+), 13 deletions(-)
```

```
diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
index 1e6eb0c..8c4bad0 100644
--- a/include/linux/memcontrol.h
+++ b/include/linux/memcontrol.h
@@ -177,6 +177,7 @@ extern void swap_cgroup_uncharge(struct swap_info_struct *si,
     unsigned long offset);
 extern struct mem_cgroup *page_to_memcg(struct page *page);
 extern int swap_cgroup_vm_swap_full(struct mem_cgroup *memcg);
+extern void __swap_cgroup_force_empty(struct mem_cgroup *mem);
 #else
 static inline
 struct mem_cgroup **swap_info_clear_memcg(struct swap_info_struct *p)
@@ -211,6 +212,10 @@ static inline int swap_cgroup_vm_swap_full(struct mem_cgroup
*memcg)
 {
  return 0;
 }
+
+static inline void __swap_cgroup_force_empty(struct mem_cgroup *mem)
+{
+}
 #endif

 #endif /* _LINUX_MEMCONTROL_H */
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 160fca1..f93907c 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -826,6 +826,34 @@ static void mem_cgroup_force_empty_list(struct mem_cgroup *mem,
  spin_unlock_irqrestore(&mz->lru_lock, flags);
 }

+static inline int mem_cgroup_in_use(struct mem_cgroup *mem)
+{
+ return mem->res.usage > 0;
+}
+
+static inline int swap_cgroup_in_use(struct mem_cgroup *mem)
+{
+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+ return mem->swap_res.usage > 0;
+#else
+ return 0;
+#endif
+}
+
+static void __mem_cgroup_force_empty(struct mem_cgroup *mem)
```

```
+{
+ int node, zid;
+
+ for_each_node_state(node, N_POSSIBLE)
+  for (zid = 0; zid < MAX_NR_ZONES; zid++) {
+   struct mem_cgroup_per_zone *mz;
+   enum lru_list l;
+   mz = mem_cgroup_zoneinfo(mem, node, zid);
+   for_each_lru(l)
+    mem_cgroup_force_empty_list(mem, mz, l);
+  }
+}
+
 /*
  * make mem_cgroup's charge to be 0 if there is no task.
  * This enables deleting this mem_cgroup.
@@ -833,7 +861,6 @@ static void mem_cgroup_force_empty_list(struct mem_cgroup *mem,
 static int mem_cgroup_force_empty(struct mem_cgroup *mem)
 {
  int ret = -EBUSY;
- int node, zid;

  css_get(&mem->css);
  /*
@@ -841,18 +868,17 @@ static int mem_cgroup_force_empty(struct mem_cgroup *mem)
   * active_list <-> inactive_list while we don't take a lock.
   * So, we have to do loop here until all lists are empty.
   */
- while (mem->res.usage > 0) {
+ while (mem_cgroup_in_use(mem) || swap_cgroup_in_use(mem)) {
   if (atomic_read(&mem->css.cgroup->count) > 0)
    goto out;
-  for_each_node_state(node, N_POSSIBLE)
-   for (zid = 0; zid < MAX_NR_ZONES; zid++) {
-    struct mem_cgroup_per_zone *mz;
-    enum lru_list l;
-    mz = mem_cgroup_zoneinfo(mem, node, zid);
-    for_each_lru(l)
-     mem_cgroup_force_empty_list(mem, mz, l);
-   }
+
+  if (mem_cgroup_in_use(mem))
+   __mem_cgroup_force_empty(mem);
+
+  if (swap_cgroup_in_use(mem))
+   __swap_cgroup_force_empty(mem);
  }
+
```

```
  ret = 0;
 out:
  css_put(&mem->css);
@@ -1289,6 +1315,7 @@ void swap_cgroup_uncharge(struct swap_info_struct *si,
   *    and called swap_entry_free().
   * 2. when this swap entry had been assigned by
   *    get_swap_page_of_type() (via SWSUSP?).
+  * 3. force empty can make such entries.
   */
  if (mem) {
   res_counter_uncharge(&mem->swap_res, PAGE_SIZE);
diff --git a/mm/swapfile.c b/mm/swapfile.c
index 6a863bc..2263ae8 100644
--- a/mm/swapfile.c
+++ b/mm/swapfile.c
@@ -704,15 +704,30 @@ static int unuse_mm(struct mm_struct *mm,
 }

 /*
+ * return the mem_cgroup to which a entry is charged
+ */
+static inline struct mem_cgroup *entry_to_memcg(swp_entry_t entry)
+{
+#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
+ return swap_info[swp_type(entry)].memcg[swp_offset(entry)];
+#else
+ return NULL;
+#endif
+}
+
+/*
  * Scan swap_map from current position to next entry still in use.
+ * If mem is specified, the entry should be charged to it.
  * Recycle to start on reaching the end, returning 0 when empty.
  */
 static unsigned int find_next_to_unuse(struct swap_info_struct *si,
-     unsigned int prev)
+     unsigned int prev,
+     struct mem_cgroup *mem)
 {
  unsigned int max = si->max;
  unsigned int i = prev;
  int count;
+ int type = si - swap_info;

  /*
   * No need for swap_lock here: we're just looking
@@ -735,7 +750,8 @@ static unsigned int find_next_to_unuse(struct swap_info_struct *si,
```

```
    i = 1;
   }
   count = si->swap_map[i];
-  if (count && count != SWAP_MAP_BAD)
+  if (count && count != SWAP_MAP_BAD
+   && (!mem || mem == entry_to_memcg(swp_entry(type, i))))
    break;
  }
  return i;
@@ -782,7 +798,7 @@ static int try_to_unuse(unsigned int type)
   * one pass through swap_map is enough, but not necessarily:
   * there are races when an instance of an entry might be missed.
   */
- while ((i = find_next_to_unuse(si, i)) != 0) {
+ while ((i = find_next_to_unuse(si, i, NULL)) != 0) {
   if (signal_pending(current)) {
    retval = -EINTR;
    break;
@@ -1923,5 +1939,35 @@ struct mem_cgroup *page_to_memcg(struct page *page)
 out:
  return mem;
 }
+
+/*
+ * uncharge all the entries that are charged to the group.
+ */
+void __swap_cgroup_force_empty(struct mem_cgroup *mem)
+{
+ struct swap_info_struct *p;
+ int type;
+
+ spin_lock(&swap_lock);
+ for (type = swap_list.head; type >= 0; type = swap_info[type].next) {
+  p = swap_info + type;
+
+  if ((p->flags & SWP_ACTIVE) == SWP_ACTIVE) {
+   unsigned int i = 0;
+
+   spin_unlock(&swap_lock);
+   while ((i = find_next_to_unuse(p, i, mem)) != 0) {
+    spin_lock(&swap_lock);
+    if (p->swap_map[i] && p->memcg[i] == mem)
+     swap_cgroup_uncharge(p, i);
+    spin_unlock(&swap_lock);
+   }
+   spin_lock(&swap_lock);
+  }
+ }
+ spin_unlock(&swap_lock);
+}
```

```
+ spin_unlock(&swap_lock);
+
+ return;
+}
 #endif
```

Subject: Re: [PATCH -mm 5/5] swapcgroup (v3): implement force_empty
Posted by yamamoto on Fri, 04 Jul 2008 06:54:31 GMT
View Forum Message <> Reply to Message

hi,

```
> +/*
> + * uncharge all the entries that are charged to the group.
> + */
> +void __swap_cgroup_force_empty(struct mem_cgroup *mem)
> +{
> + struct swap_info_struct *p;
> + int type;
> +
> + spin_lock(&swap_lock);
> + for (type = swap_list.head; type >= 0; type = swap_info[type].next) {
> +  p = swap_info + type;
> +
> +  if ((p->flags & SWP_ACTIVE) == SWP_ACTIVE) {
> +   unsigned int i = 0;
> +
> +   spin_unlock(&swap_lock);
```

what prevents the device from being swapoff'ed while you drop swap_lock?

YAMAMOTO Takashi

```
> +   while ((i = find_next_to_unuse(p, i, mem)) != 0) {
> +    spin_lock(&swap_lock);
> +    if (p->swap_map[i] && p->memcg[i] == mem)
> +     swap_cgroup_uncharge(p, i);
> +    spin_unlock(&swap_lock);
> +   }
> +   spin_lock(&swap_lock);
> +  }
> + }
```

> + spin_unlock(&swap_lock);
> +
> + return;
> +}
> #endif

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH -mm 5/5] swapcgroup (v3): implement force_empty
Posted by Daisuke Nishimura on Fri, 04 Jul 2008 07:26:29 GMT
View Forum Message <> Reply to Message

Hi, Yamamoto-san.

Thank you for your comment.

On Fri, 4 Jul 2008 15:54:31 +0900 (JST), yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
> hi,
>
> > +/*
> > + * uncharge all the entries that are charged to the group.
> > + */
> > +void __swap_cgroup_force_empty(struct mem_cgroup *mem)
> > +{
> > + struct swap_info_struct *p;
> > + int type;
> > +
> > + spin_lock(&swap_lock);
> > + for (type = swap_list.head; type >= 0; type = swap_info[type].next) {
> > + p = swap_info + type;
> > +
> > + if ((p->flags & SWP_ACTIVE) == SWP_ACTIVE) {
> > +   unsigned int i = 0;
> > +
> > +   spin_unlock(&swap_lock);
>
> what prevents the device from being swapoff'ed while you drop swap_lock?
>
Nothing.

After searching the entry to be uncharged(find_next_to_unuse below),
I recheck under swap_lock whether the entry is charged to the group.
Even if the device is swapoff'ed, swap_off must have uncharged the entry,
so I don't think it's needed anyway.

> YAMAMOTO Takashi
>
> > + while ((i = find_next_to_unuse(p, i, mem)) != 0) {
> > + spin_lock(&swap_lock);
> > + if (p->swap_map[i] && p->memcg[i] == mem)
Ah, I think it should be added !p->swap_map to check the device has not
been swapoff'ed.

Thanks,
Daisuke Nishimura.

> > + swap_cgroup_uncharge(p, i);
> > + spin_unlock(&swap_lock);
> > + }
> > + spin_lock(&swap_lock);
> > + }
> > + }
> > + spin_unlock(&swap_lock);
> > +
> > + return;
> > +}
> > #endif

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH -mm 5/5] swapcgroup (v3): implement force_empty
Posted by yamamoto on Fri, 04 Jul 2008 07:48:28 GMT
View Forum Message <> Reply to Message

> Hi, Yamamoto-san.
>
> Thank you for your comment.
>
> On Fri, 4 Jul 2008 15:54:31 +0900 (JST), yamamoto@valinux.co.jp (YAMAMOTO Takashi)
wrote:
> > hi,
> >
> > > +/*
> > > + * uncharge all the entries that are charged to the group.
> > > + */
> > > +void __swap_cgroup_force_empty(struct mem_cgroup *mem)
> > > +{
> > > + struct swap_info_struct *p;

> > > + int type;
> > > +
> > > + spin_lock(&swap_lock);
> > > + for (type = swap_list.head; type >= 0; type = swap_info[type].next) {
> > > + p = swap_info + type;
> > > +
> > > + if ((p->flags & SWP_ACTIVE) == SWP_ACTIVE) {
> > > +   unsigned int i = 0;
> > > +
> > > +   spin_unlock(&swap_lock);
> >
> > what prevents the device from being swapoff'ed while you drop swap_lock?
> >
> Nothing.
>
> After searching the entry to be uncharged(find_next_to_unuse below),
> I recheck under swap_lock whether the entry is charged to the group.
> Even if the device is swapoff'ed, swap_off must have uncharged the entry,
> so I don't think it's needed anyway.
>
> > YAMAMOTO Takashi
> >
> > > +   while ((i = find_next_to_unuse(p, i, mem)) != 0) {
> > > +   spin_lock(&swap_lock);
> > > +   if (p->swap_map[i] && p->memcg[i] == mem)
> Ah, I think it should be added !p->swap_map to check the device has not
> been swapoff'ed.

find_next_to_unuse seems to have fragile assumptions and
can dereference p->swap_map as well.

YAMAMOTO Takashi

>
>
> Thanks,
> Daisuke Nishimura.
>
> > > +   swap_cgroup_uncharge(p, i);
> > > +   spin_unlock(&swap_lock);
> > > +   }
> > > +   spin_lock(&swap_lock);
> > > + }
> > > + }
> > > + spin_unlock(&swap_lock);
> > > +
> > > + return;
> > > +}

> > > #endif

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH -mm 5/5] swapcgroup (v3): implement force_empty
Posted by Daisuke Nishimura on Fri, 04 Jul 2008 07:56:05 GMT
View Forum Message <> Reply to Message

On Fri, 4 Jul 2008 16:48:28 +0900 (JST), yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
> > Hi, Yamamoto-san.
> >
> > Thank you for your comment.
> >
> > On Fri, 4 Jul 2008 15:54:31 +0900 (JST), yamamoto@valinux.co.jp (YAMAMOTO Takashi) wrote:
> > > hi,
> > >
> > > > +/*
> > > > + * uncharge all the entries that are charged to the group.
> > > > + */
> > > > +void __swap_cgroup_force_empty(struct mem_cgroup *mem)
> > > > +{
> > > > + struct swap_info_struct *p;
> > > > + int type;
> > > > +
> > > > + spin_lock(&swap_lock);
> > > > + for (type = swap_list.head; type >= 0; type = swap_info[type].next) {
> > > > + p = swap_info + type;
> > > > +
> > > > + if ((p->flags & SWP_ACTIVE) == SWP_ACTIVE) {
> > > > + unsigned int i = 0;
> > > > +
> > > > + spin_unlock(&swap_lock);
> > >
> > > what prevents the device from being swapoff'ed while you drop swap_lock?
> > >
> > Nothing.
> >
> > After searching the entry to be uncharged(find_next_to_unuse below),
> > I recheck under swap_lock whether the entry is charged to the group.
> > Even if the device is swapoff'ed, swap_off must have uncharged the entry,
> > so I don't think it's needed anyway.
> >
> > > YAMAMOTO Takashi

> > >
> > > > +   while ((i = find_next_to_unuse(p, i, mem)) != 0) {
> > > > +   spin_lock(&swap_lock);
> > > > +   if (p->swap_map[i] && p->memcg[i] == mem)
> > Ah, I think it should be added !p->swap_map to check the device has not
> > been swapoff'ed.
>
> find_next_to_unuse seems to have fragile assumptions and
> can dereference p->swap_map as well.
>
You're right.
Thank you for pointing it out!

I'll consider more.

Thanks,
Daisuke Nishimura.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH -mm 0/5] swapcgroup (v3)
Posted by KAMEZAWA Hiroyuki on Fri, 04 Jul 2008 09:38:25 GMT
View Forum Message <> Reply to Message

On Fri, 4 Jul 2008 15:15:36 +0900
Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp> wrote:

> Hi.
>
> This is new version of swapcgroup.
>
> Major changes from previous version
> - Rebased on 2.6.26-rc5-mm3.
>   The new -mm has been released, but these patches
>   can be applied on 2.6.26-rc8-mm1 too with only some offset warnings.
>   I tested these patches on 2.6.26-rc5-mm3 with some fixes about memory,
>   and it seems to work fine.
> - (NEW) Implemented force_empty.
>   Currently, it simply uncharges all the charges from the group.
>
> Patches
> - [1/5] add cgroup files
> - [2/5] add a member to swap_info_struct
> - [3/5] implement charge and uncharge

> - [4/5] modify vm_swap_full()
> - [5/5] implement force_empty
>
> ToDo(in my thought. Feel free to add some others here.)
> - need some documentation
> Add to memory.txt? or create a new documentation file?
>
Maybe new documentation file is better.

> - add option to disable only this feature
> I'm wondering if this option is needed.
> memcg has already the boot option to disable it.
> Is there any case where memory should be accounted but swap should not?

On x86-32, area for vmalloc() is very small and array of swap_info_struct[]
will use much amount of it. If vmalloc() area is too small, the kernel cannot
load modules.

Thanks,
-Kame

> - hierarchy support
> - move charges along with task
> Both of them need more discussion.
>
>
> Thanks,
> Daisuke Nishimura.
>

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH -mm 4/5] swapcgroup (v3): modify vm_swap_full()
Posted by KAMEZAWA Hiroyuki on Fri, 04 Jul 2008 09:58:45 GMT
View Forum Message <> Reply to Message

On Fri, 4 Jul 2008 15:22:44 +0900
Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp> wrote:

> /* Swap 50% full? Release swapcache more aggressively.. */
> -#define vm_swap_full() (nr_swap_pages*2 < total_swap_pages)
> +#define vm_swap_full(memcg) ((nr_swap_pages*2 < total_swap_pages) \

> + || swap_cgroup_vm_swap_full(memcg))
> +
>
Maybe nitpick but I like
==
  vm_swap_full(page) ((nr_swap_pages *2 < total_swap_pages)
    || swap_cgroup_vm_swap_full_page(page))
==
rather than vm_swap_full(memcg)

And could you change this to inline funcition ?

> /* linux/mm/page_alloc.c */
> extern unsigned long totalram_pages;
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index d16d0a5..160fca1 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -1296,5 +1296,23 @@ void swap_cgroup_uncharge(struct swap_info_struct *si,
>   css_put(&mem->css);
>   }
> }
> +
> +int swap_cgroup_vm_swap_full(struct mem_cgroup *memcg)
> +{
> + u64 usage;
> + u64 limit;
> +
> + if (memcg)
> + css_get(&memcg->css);
> + else
> + return 0;
> +
> + usage = res_counter_read_u64(&memcg->swap_res, RES_USAGE);
> + limit = res_counter_read_u64(&memcg->swap_res, RES_LIMIT);
> +
> + css_put(&memcg->css);
> +
> + return usage * 2 > limit;
> +}
> #endif
>

How about this under above my changes ?
==
int memcg_swap_full(struct mem_cgroup *mem)
{
 if (!mem)

```
  return 0;
 usage = res_counter_read_u64(&memcg->swap_res, RES_USAGE);
 limit = res_counter_read_u64(&memcg->swap_res, RES_LIMIT);

 return (usage *2 > limit);
}

int swap_cgroup_vm_swap_full_page(struct page *page)
{
 struct page_cgroup *pc;
 int ret = 0;

 if (mem_cgroup_subsys.disabled)
  return 0;

 if (!PageSwapCache(page))
  return 0;

 entry.val = page_private(page);
 p = swap_info_get(entry);
 if (!p)
  goto out;

 mem = p->memcg[swp_offset(entry)];
 /* becasue we get swap_lock here, access to memcg is safe.*/
 ret = memcg_swap_full(mem);
 spin_unlock(&swap_lock);
out:
 return ret;
}
==
```

Thanks,
-Kame

```
> diff --git a/mm/memory.c b/mm/memory.c
> index 536d748..afcf737 100644
> --- a/mm/memory.c
> +++ b/mm/memory.c
> @@ -2230,7 +2230,9 @@ static int do_swap_page(struct mm_struct *mm, struct
vm_area_struct *vma,
>   page_add_anon_rmap(page, vma, address);
>
>   swap_free(entry);
> - if (vm_swap_full() || (vma->vm_flags & VM_LOCKED) || PageMlocked(page))
> + if (vm_swap_full(page_to_memcg(page))
```

```
> +  || (vma->vm_flags & VM_LOCKED)
> +  || PageMlocked(page))
>     remove_exclusive_swap_page(page);
>    unlock_page(page);
>
> diff --git a/mm/swapfile.c b/mm/swapfile.c
> index 57798c5..6a863bc 100644
> --- a/mm/swapfile.c
> +++ b/mm/swapfile.c
> @@ -28,6 +28,7 @@
>   #include <linux/capability.h>
>   #include <linux/syscalls.h>
>   #include <linux/memcontrol.h>
> +#include <linux/cgroup.h>
>
>   #include <asm/pgtable.h>
>   #include <asm/tlbflush.h>
> @@ -447,7 +448,8 @@ void free_swap_and_cache(swp_entry_t entry)
>     /* Only cache user (+us), or swap space full? Free it! */
>     /* Also recheck PageSwapCache after page is locked (above) */
>     if (PageSwapCache(page) && !PageWriteback(page) &&
> -      (one_user || vm_swap_full())) {
> +      (one_user
> +      || vm_swap_full(page_to_memcg(page)))) {
>      delete_from_swap_cache(page);
>      SetPageDirty(page);
>     }
> @@ -1889,3 +1891,37 @@ int valid_swaphandles(swp_entry_t entry, unsigned long *offset)
>    *offset = ++toff;
>    return nr_pages? ++nr_pages: 0;
>   }
> +
> +#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
> +/*
> + * returns mem_cgroup to which the swap cache page is charged as swap.
> + * should be called with the page locked.
> + */
> +struct mem_cgroup *page_to_memcg(struct page *page)
> +{
> + struct swap_info_struct *p;
> + struct mem_cgroup *mem = NULL;
> + swp_entry_t entry;
> +
> + BUG_ON(!PageLocked(page));
> +
> + if (mem_cgroup_subsys.disabled)
> +  goto out;
> +
```

> + if (!PageSwapCache(page))
> + goto out;
> +
> + entry.val = page_private(page);
> + p = swap_info_get(entry);
> + if (!p)
> + goto out;
> +
> + mem = p->memcg[swp_offset(entry)];
> +
> + spin_unlock(&swap_lock);
> +
> +out:
> + return mem;
> +}

> +#endif
> +
> diff --git a/mm/vmscan.c b/mm/vmscan.c
> index 9a5e423..ac45993 100644
> --- a/mm/vmscan.c
> +++ b/mm/vmscan.c
> @@ -752,7 +752,7 @@ cull_mlocked:
>
> activate_locked:
>   /* Not a candidate for swapping, so reclaim swap space. */
> - if (PageSwapCache(page) && vm_swap_full())
> + if (PageSwapCache(page) && vm_swap_full(page_to_memcg(page)))
>   remove_exclusive_swap_page_ref(page);
>   VM_BUG_ON(PageActive(page));
>   SetPageActive(page);
> @@ -1317,7 +1317,7 @@ static void shrink_active_list(unsigned long nr_pages, struct zone
*zone,
>   __mod_zone_page_state(zone, NR_LRU_BASE + lru, pgmoved);
>   pgmoved = 0;
>   spin_unlock_irq(&zone->lru_lock);
> -   if (vm_swap_full())
> +   if (vm_swap_full(sc->mem_cgroup))
>   pagevec_swap_free(&pvec);
>   __pagevec_release(&pvec);
>   spin_lock_irq(&zone->lru_lock);
> @@ -1328,7 +1328,7 @@ static void shrink_active_list(unsigned long nr_pages, struct zone
*zone,
>   __count_zone_vm_events(PGREFILL, zone, pgscanned);
>   __count_vm_events(PGDEACTIVATE, pgdeactivate);
>   spin_unlock_irq(&zone->lru_lock);

> - if (vm_swap_full())
> + if (vm_swap_full(sc->mem_cgroup))
> pagevec_swap_free(&pvec);
>
> pagevec_release(&pvec);
>

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH -mm 5/5] swapcgroup (v3): implement force_empty
Posted by KAMEZAWA Hiroyuki on Fri, 04 Jul 2008 10:15:11 GMT
View Forum Message <> Reply to Message

On Fri, 4 Jul 2008 15:24:23 +0900
Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp> wrote:

> This patch implements force_empty of swapcgroup.
>
> Currently, it simply uncharges all the charges from the group.
>
> I think there can be other implementations.
>
> What I thought are:
> - move all the charges to its parent.
> - unuse(swap in) all the swap charged to the group.
>
3. move all swap back to memory (see swapoff.)

> But in any case, I think before implementing this way,
> hierarchy and move charges support are needed.
>
> So I think this is enough for the first step.
>

I don't think hierarchy/automatic-load-balancer for swap cg is necessary.
Hmm...but handling limit_change (at least, returns -EBUSY) will be necessary.
Do you consider a some magical way to move pages in swap back to memory ?

In general, I like this set but we can't change the limit on demand. (maybe)
(just putting it to TO-DO-List is okay to me.)

Thanks,

-Kame

>
> Change log
> v2->v3
> - new patch
>
>
> Signed-off-by: Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp>
>
> ---
> include/linux/memcontrol.h | 5 ++++
> mm/memcontrol.c | 47 +++++++++++++++++++++++++++++++--------
> mm/swapfile.c | 52 ++--
> 3 files changed, 91 insertions(+), 13 deletions(-)
>
> diff --git a/include/linux/memcontrol.h b/include/linux/memcontrol.h
> index 1e6eb0c..8c4bad0 100644
> --- a/include/linux/memcontrol.h
> +++ b/include/linux/memcontrol.h
> @@ -177,6 +177,7 @@ extern void swap_cgroup_uncharge(struct swap_info_struct *si,
>   unsigned long offset);
> extern struct mem_cgroup *page_to_memcg(struct page *page);
> extern int swap_cgroup_vm_swap_full(struct mem_cgroup *memcg);
> +extern void __swap_cgroup_force_empty(struct mem_cgroup *mem);
> #else
> static inline
> struct mem_cgroup **swap_info_clear_memcg(struct swap_info_struct *p)
> @@ -211,6 +212,10 @@ static inline int swap_cgroup_vm_swap_full(struct mem_cgroup
*memcg)
> {
>   return 0;
> }
> +
> +static inline void __swap_cgroup_force_empty(struct mem_cgroup *mem)
> +{
> +}
> #endif
>
> #endif /* _LINUX_MEMCONTROL_H */
> diff --git a/mm/memcontrol.c b/mm/memcontrol.c
> index 160fca1..f93907c 100644
> --- a/mm/memcontrol.c
> +++ b/mm/memcontrol.c
> @@ -826,6 +826,34 @@ static void mem_cgroup_force_empty_list(struct mem_cgroup *mem,
>   spin_unlock_irqrestore(&mz->lru_lock, flags);
> }

>
> +static inline int mem_cgroup_in_use(struct mem_cgroup *mem)
> +{
> + return mem->res.usage > 0;
> +}
> +
> +static inline int swap_cgroup_in_use(struct mem_cgroup *mem)
> +{
> +#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
> + return mem->swap_res.usage > 0;
> +#else
> + return 0;
> +#endif
> +}
> +
> +static void __mem_cgroup_force_empty(struct mem_cgroup *mem)
> +{
> + int node, zid;
> +
> + for_each_node_state(node, N_POSSIBLE)
> + for (zid = 0; zid < MAX_NR_ZONES; zid++) {
> +   struct mem_cgroup_per_zone *mz;
> +   enum lru_list l;
> +   mz = mem_cgroup_zoneinfo(mem, node, zid);
> +   for_each_lru(l)
> +   mem_cgroup_force_empty_list(mem, mz, l);
> + }
> +}
> +
> /*
>   * make mem_cgroup's charge to be 0 if there is no task.
>   * This enables deleting this mem_cgroup.
> @@ -833,7 +861,6 @@ static void mem_cgroup_force_empty_list(struct mem_cgroup *mem,
> static int mem_cgroup_force_empty(struct mem_cgroup *mem)
> {
>   int ret = -EBUSY;
> - int node, zid;
>
>   css_get(&mem->css);
>   /*
> @@ -841,18 +868,17 @@ static int mem_cgroup_force_empty(struct mem_cgroup *mem)
>   * active_list <-> inactive_list while we don't take a lock.
>   * So, we have to do loop here until all lists are empty.
>   */
> - while (mem->res.usage > 0) {
> + while (mem_cgroup_in_use(mem) || swap_cgroup_in_use(mem)) {
>   if (atomic_read(&mem->css.cgroup->count) > 0)
>   goto out;

```
> -  for_each_node_state(node, N_POSSIBLE)
> -   for (zid = 0; zid < MAX_NR_ZONES; zid++) {
> -    struct mem_cgroup_per_zone *mz;
> -    enum lru_list l;
> -    mz = mem_cgroup_zoneinfo(mem, node, zid);
> -    for_each_lru(l)
> -     mem_cgroup_force_empty_list(mem, mz, l);
> -   }
> +
> +  if (mem_cgroup_in_use(mem))
> +   __mem_cgroup_force_empty(mem);
> +
> +  if (swap_cgroup_in_use(mem))
> +   __swap_cgroup_force_empty(mem);
>   }
> +
>   ret = 0;
>  out:
>   css_put(&mem->css);
> @@ -1289,6 +1315,7 @@ void swap_cgroup_uncharge(struct swap_info_struct *si,
>    *    and called swap_entry_free().
>    * 2. when this swap entry had been assigned by
>    *    get_swap_page_of_type() (via SWSUSP?).
> +  * 3. force empty can make such entries.
>    */
>   if (mem) {
>    res_counter_uncharge(&mem->swap_res, PAGE_SIZE);
> diff --git a/mm/swapfile.c b/mm/swapfile.c
> index 6a863bc..2263ae8 100644
> --- a/mm/swapfile.c
> +++ b/mm/swapfile.c
> @@ -704,15 +704,30 @@ static int unuse_mm(struct mm_struct *mm,
>  }
>
>  /*
> + * return the mem_cgroup to which a entry is charged
> + */
> +static inline struct mem_cgroup *entry_to_memcg(swp_entry_t entry)
> +{
> +#ifdef CONFIG_CGROUP_SWAP_RES_CTLR
> + return swap_info[swp_type(entry)].memcg[swp_offset(entry)];
> +#else
> + return NULL;
> +#endif
> +}
> +
> +/*
>   * Scan swap_map from current position to next entry still in use.
```

```
> + * If mem is specified, the entry should be charged to it.
>   * Recycle to start on reaching the end, returning 0 when empty.
>   */
>  static unsigned int find_next_to_unuse(struct swap_info_struct *si,
> -     unsigned int prev)
> +     unsigned int prev,
> +     struct mem_cgroup *mem)
>  {
>   unsigned int max = si->max;
>   unsigned int i = prev;
>   int count;
> + int type = si - swap_info;
>
>   /*
>    * No need for swap_lock here: we're just looking
> @@ -735,7 +750,8 @@ static unsigned int find_next_to_unuse(struct swap_info_struct *si,
>     i = 1;
>    }
>    count = si->swap_map[i];
> -  if (count && count != SWAP_MAP_BAD)
> +  if (count && count != SWAP_MAP_BAD
> +   && (!mem || mem == entry_to_memcg(swp_entry(type, i))))
>     break;
>   }
>   return i;
> @@ -782,7 +798,7 @@ static int try_to_unuse(unsigned int type)
>    * one pass through swap_map is enough, but not necessarily:
>    * there are races when an instance of an entry might be missed.
>    */
> - while ((i = find_next_to_unuse(si, i)) != 0) {
> + while ((i = find_next_to_unuse(si, i, NULL)) != 0) {
>    if (signal_pending(current)) {
>     retval = -EINTR;
>     break;
> @@ -1923,5 +1939,35 @@ struct mem_cgroup *page_to_memcg(struct page *page)
>  out:
>   return mem;
>  }
> +
> +/*
> + * uncharge all the entries that are charged to the group.
> + */
> +void __swap_cgroup_force_empty(struct mem_cgroup *mem)
> +{
> + struct swap_info_struct *p;
> + int type;
> +
> + spin_lock(&swap_lock);
```

```
> + for (type = swap_list.head; type >= 0; type = swap_info[type].next) {
> +  p = swap_info + type;
> +
> +  if ((p->flags & SWP_ACTIVE) == SWP_ACTIVE) {
> +   unsigned int i = 0;
> +
> +   spin_unlock(&swap_lock);
> +   while ((i = find_next_to_unuse(p, i, mem)) != 0) {
> +    spin_lock(&swap_lock);
> +    if (p->swap_map[i] && p->memcg[i] == mem)
> +     swap_cgroup_uncharge(p, i);
> +    spin_unlock(&swap_lock);
> +   }
> +   spin_lock(&swap_lock);
> +  }
> + }
> + spin_unlock(&swap_lock);
> +
> + return;
> +}
>  #endif
>
>
```

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH -mm 4/5] swapcgroup (v3): modify vm_swap_full()
Posted by Daisuke Nishimura on Fri, 04 Jul 2008 10:40:59 GMT
View Forum Message <> Reply to Message

Hi, Kamezawa-san.

On Fri, 4 Jul 2008 18:58:45 +0900, KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> On Fri, 4 Jul 2008 15:22:44 +0900
> Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp> wrote:
>
> > /* Swap 50% full? Release swapcache more aggressively.. */
> > -#define vm_swap_full() (nr_swap_pages*2 < total_swap_pages)
> > +#define vm_swap_full(memcg) ((nr_swap_pages*2 < total_swap_pages) \
> > + || swap_cgroup_vm_swap_full(memcg))
> > +
> >
> Maybe nitpick but I like

> ==
>   vm_swap_full(page) ((nr_swap_pages *2 < total_swap_pages)
> || swap_cgroup_vm_swap_full_page(page))
> ==
> rather than vm_swap_full(memcg)
>
Well, I used "page" in v2, but Kosaki-san said vm_swap_full()
is not page-granularity operation so it should be changed.

And more,

> > @@ -1317,7 +1317,7 @@ static void shrink_active_list(unsigned long nr_pages, struct zone
*zone,
> >   __mod_zone_page_state(zone, NR_LRU_BASE + lru, pgmoved);
> >   pgmoved = 0;
> >   spin_unlock_irq(&zone->lru_lock);
> > -   if (vm_swap_full())
> > +   if (vm_swap_full(sc->mem_cgroup))
> >   pagevec_swap_free(&pvec);
> >   __pagevec_release(&pvec);
> >   spin_lock_irq(&zone->lru_lock);
> > @@ -1328,7 +1328,7 @@ static void shrink_active_list(unsigned long nr_pages, struct zone
*zone,
> >   __count_zone_vm_events(PGREFILL, zone, pgscanned);
> >   __count_vm_events(PGDEACTIVATE, pgdeactivate);
> >   spin_unlock_irq(&zone->lru_lock);
> > - if (vm_swap_full())
> > + if (vm_swap_full(sc->mem_cgroup))
> >   pagevec_swap_free(&pvec);
> >
> >   pagevec_release(&pvec);

"page" cannot be determined in those places.
I don't want to change pagevec_swap_free(), so I changed
the argument of vm_swap_full().

> And could you change this to inline funcition ?
>
Of course.
I think it would be better.

Thanks,
Daisuke Nishimura.

Subject: Re: [PATCH -mm 0/5] swapcgroup (v3)
Posted by Daisuke Nishimura on Fri, 04 Jul 2008 10:58:36 GMT
View Forum Message <> Reply to Message

On Fri, 4 Jul 2008 18:40:33 +0900, KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> On Fri, 4 Jul 2008 15:15:36 +0900
> Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp> wrote:
>
> > Hi.
> >
> > This is new version of swapcgroup.
> >
> > Major changes from previous version
> > - Rebased on 2.6.26-rc5-mm3.
> >   The new -mm has been released, but these patches
> >   can be applied on 2.6.26-rc8-mm1 too with only some offset warnings.
> >   I tested these patches on 2.6.26-rc5-mm3 with some fixes about memory,
> >   and it seems to work fine.
> > - (NEW) Implemented force_empty.
> >   Currently, it simply uncharges all the charges from the group.
> >
> > Patches
> > - [1/5] add cgroup files
> > - [2/5] add a member to swap_info_struct
> > - [3/5] implement charge and uncharge
> > - [4/5] modify vm_swap_full()
> > - [5/5] implement force_empty
> >
> > ToDo(in my thought. Feel free to add some others here.)
> > - need some documentation
> >   Add to memory.txt? or create a new documentation file?
> >
> Maybe new documentation file is better.
>
O.K.

> > - add option to disable only this feature
> >   I'm wondering if this option is needed.
> >   memcg has already the boot option to disable it.
> >   Is there any case where memory should be accounted but swap should not?
>
> On x86-32, area for vmalloc() is very small and array of swap_info_struct[]
> will use much amount of it. If vmalloc() area is too small, the kernel cannot
> load modules.
>
It would be critical.
I'll add an option.

Thanks,
Daisuke Nishimura.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH -mm 5/5] swapcgroup (v3): implement force_empty
Posted by Daisuke Nishimura on Fri, 04 Jul 2008 12:33:01 GMT
View Forum Message <> Reply to Message

On Fri, 4 Jul 2008 19:16:38 +0900, KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> On Fri, 4 Jul 2008 15:24:23 +0900
> Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp> wrote:
>
> > This patch implements force_empty of swapcgroup.
> >
> > Currently, it simply uncharges all the charges from the group.
> >
> > I think there can be other implementations.
> >
> > What I thought are:
> > - move all the charges to its parent.
> > - unuse(swap in) all the swap charged to the group.
> >
> 3. move all swap back to memory (see swapoff.)
>
>
Do you mean swapping in all the swap including used by
other groups?
It would be one choice anyway.

> > But in any case, I think before implementing this way,
> > hierarchy and move charges support are needed.
> >
> > So I think this is enough for the first step.
> >
>
> I don't think hierarchy/automatic-load-balancer for swap cg is necessary.
It's the problem of how the "hierarchy" would be, I think.
I'm saying "hierarchy" here just to mean "some kind of feature
where a parent includes their children".
I think "hierarchy" is needed if we implement the choice 1 above,
and I personally think it would be the best choice.

> Hmm...but handling limit_change (at least, returns -EBUSY) will be necessary.
I think so too.
But I'm not sure now it's good or bad to support shrinking at limit_change
about swap.
Shrinking swap means increasing the memory usage and that may cause
another swapout.

> Do you consider a some magical way to move pages in swap back to memory ?
>
In this patch, I modified the find_next_to_unuse() to find
the entry charged to a specific group.
It might be possible to modify try_to_unuse()(or define another function
based on try_to_unuse()) to reduce swap usage of a specified group
down to some threashold.
But, I think, one problem here is from which device the swaps
should be back to memory, or usage balance between swap devices.

> In general, I like this set but we can't change the limit on demand. (maybe)
> (just putting it to TO-DO-List is okay to me.)
>
I'm sorry but what do you mean by "change the limit on demand"?
Could you explain more?

Thanks,
Daisuke Nishimura.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH -mm 5/5] swapcgroup (v3): implement force_empty
Posted by KAMEZAWA Hiroyuki on Sat, 05 Jul 2008 04:27:31 GMT
View Forum Message <> Reply to Message

On Fri, 4 Jul 2008 21:33:01 +0900
Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp> wrote:

> On Fri, 4 Jul 2008 19:16:38 +0900, KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
wrote:
> > On Fri, 4 Jul 2008 15:24:23 +0900
> > Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp> wrote:
> >
> > > This patch implements force_empty of swapcgroup.
> > >
> > > Currently, it simply uncharges all the charges from the group.
> > >

> > > I think there can be other implementations.
> > >
> > > What I thought are:
> > > - move all the charges to its parent.
> > > - unuse(swap in) all the swap charged to the group.
> > >
> > 3. move all swap back to memory (see swapoff.)
> >
> >
> Do you mean swapping in all the swap including used by
> other groups?

swapping in all swap used by the group (not by all group)

> It would be one choice anyway.
>
> > > But in any case, I think before implementing this way,
> > > hierarchy and move charges support are needed.
> > >
> > > So I think this is enough for the first step.
> > >
> >
> > I don't think hierarchy/automatic-load-balancer for swap cg is necessary.
> It's the problem of how the "hierarchy" would be, I think.

yes.

> I'm saying "hierarchy" here just to mean "some kind of feature
> where a parent includes their children".
> I think "hierarchy" is needed if we implement the choice 1 above,
> and I personally think it would be the best choice.
>

> > Hmm...but handling limit_change (at least, returns -EBUSY) will be necessary.
> I think so too.
> But I'm not sure now it's good or bad to support shrinking at limit_change
> about swap.
> Shrinking swap means increasing the memory usage and that may cause
> another swapout.
yes. but who reduce the limit ? it's the admin or users.

At leaset, returning -EBUSY is necesary. You can use
res_counter: check limit change patch which I posted yesterday.

>
> > Do you consider a some magical way to move pages in swap back to memory ?

> >
> In this patch, I modified the find_next_to_unuse() to find
> the entry charged to a specific group.
> It might be possible to modify try_to_unuse()(or define another function
> based on try_to_unuse()) to reduce swap usage of a specified group
> down to some threashold.
> But, I think, one problem here is from which device the swaps
> should be back to memory, or usage balance between swap devices.
>
Ah, that's maybe difficult one.
As memcg has its own LRU, add MRU to swap .....is not a choice ;(

> > In general, I like this set but we can't change the limit on demand. (maybe)
> > (just putting it to TO-DO-List is okay to me.)
> >
> I'm sorry but what do you mean by "change the limit on demand"?
> Could you explain more?
>
In short, the administrator have to write the perfect plan to set
each group's swap limit beforehand because we cannot decrease used swap.

1st problem is that the user cannot reduce the usage of swap by hand.
(He can reduce by killing process or deleting shmem.)
Once the usage of swap of a group grows, other groups can't use much.

2nd problem is there is no entity who controls the total amount of swap.
The user/admin have to check the amount of free swap space by himself at planning
each group's swap limit more carefully than memcg.

So, I think rich-control of hierarchy will be of no use ;)
All things should be planned before the system starts.

In memcg, the amount of free memory is maintained by global LRU. It does much jobs
for us. But free swap space isn't. It's just used on demand.

If we can't decrease usage of swap by a group by hand, the problem which this
swap-space-controller want to fix will not be fixed at pleasant level.

Anyway, please return -EBUSY at setting limit < usage, at first :)
That's enough for me, now.

Thanks,
-Kame

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [PATCH -mm 0/5] swapcgroup (v3)
Posted by Balbir Singh on Sat, 05 Jul 2008 06:52:25 GMT
View Forum Message <> Reply to Message

Daisuke Nishimura wrote:
> Hi.
>
> This is new version of swapcgroup.
>
> Major changes from previous version
> - Rebased on 2.6.26-rc5-mm3.
> The new -mm has been released, but these patches
> can be applied on 2.6.26-rc8-mm1 too with only some offset warnings.
> I tested these patches on 2.6.26-rc5-mm3 with some fixes about memory,
> and it seems to work fine.
> - (NEW) Implemented force_empty.
> Currently, it simply uncharges all the charges from the group.
>
> Patches
> - [1/5] add cgroup files
> - [2/5] add a member to swap_info_struct
> - [3/5] implement charge and uncharge
> - [4/5] modify vm_swap_full()
> - [5/5] implement force_empty
>
> ToDo(in my thought. Feel free to add some others here.)
> - need some documentation
> Add to memory.txt? or create a new documentation file?
>

I think memory.txt is good. But then, we'll need to add a Table of Contents to
it, so that swap controller documentation can be located easily.

> - add option to disable only this feature
> I'm wondering if this option is needed.
> memcg has already the boot option to disable it.
> Is there any case where memory should be accounted but swap should not?
>

That depends on what use case you are trying to provide. Let's say I needed
backward compatibility with 2.6.25, then I would account for memory and leave
out swap (even though we have swap controller).

> - hierarchy support
> - move charges along with task

> Both of them need more discussion.
>

Yes, they do.

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

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH -mm 5/5] swapcgroup (v3): implement force_empty
Posted by Daisuke Nishimura on Mon, 07 Jul 2008 06:23:51 GMT
View Forum Message <> Reply to Message

On Sat, 5 Jul 2008 13:29:44 +0900, KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> On Fri, 4 Jul 2008 21:33:01 +0900
> Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp> wrote:
>
> > On Fri, 4 Jul 2008 19:16:38 +0900, KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
> > > On Fri, 4 Jul 2008 15:24:23 +0900
> > > Daisuke Nishimura <nishimura@mxp.nes.nec.co.jp> wrote:
> > >
> > > > This patch implements force_empty of swapcgroup.
> > > >
> > > > Currently, it simply uncharges all the charges from the group.
> > > >
> > > > I think there can be other implementations.
> > > >
> > > > What I thought are:
> > > > - move all the charges to its parent.
> > > > - unuse(swap in) all the swap charged to the group.
> > > >
> > > 3. move all swap back to memory (see swapoff.)
> > >
> > >
> > Do you mean swapping in all the swap including used by
> > other groups?
>
> swapping in all swap used by the group (not by all group)
>

O.K. I intended to say the same thing in 2.
I'll try it and I think some part of this implementation can be
used by shrinking support too.

(snip)

> > > Hmm...but handling limit_change (at least, returns -EBUSY) will be necessary.
> > I think so too.
> > But I'm not sure now it's good or bad to support shrinking at limit_change
> > about swap.
> > Shrinking swap means increasing the memory usage and that may cause
> > another swapout.
> yes. but who reduce the limit ? it's the admin or users.
>
> At leaset, returning -EBUSY is necesary. You can use
> res_counter: check limit change patch which I posted yesterday.
>
I saw your patch, and I agree that returning -EBUSY is the first step.

> > > Do you consider a some magical way to move pages in swap back to memory ?
> > >
> > In this patch, I modified the find_next_to_unuse() to find
> > the entry charged to a specific group.
> > It might be possible to modify try_to_unuse()(or define another function
> > based on try_to_unuse()) to reduce swap usage of a specified group
> > down to some threashold.
> > But, I think, one problem here is from which device the swaps
> > should be back to memory, or usage balance between swap devices.
> >
> Ah, that's maybe difficult one.
> As memcg has its own LRU, add MRU to swap .....is not a choice ;(
>
Swap devices are used in order of their priority,
so storing per device usage might be usefull for this porpose...
Anyway, I should consider more.

> > > In general, I like this set but we can't change the limit on demand. (maybe)
> > > (just putting it to TO-DO-List is okay to me.)
> > >
> > I'm sorry but what do you mean by "change the limit on demand"?
> > Could you explain more?
> >
> In short, the administrator have to write the perfect plan to set
> each group's swap limit beforehand because we cannot decrease used swap.
>
> 1st problem is that the user cannot reduce the usage of swap by hand.
> (He can reduce by killing process or deleting shmem.)
> Once the usage of swap of a group grows, other groups can't use much.

>
> 2nd problem is there is no entity who controls the total amount of swap.
> The user/admin have to check the amount of free swap space by himself at planning
> each group's swap limit more carefully than memcg.
>
> So, I think rich-control of hierarchy will be of no use ;)
> All things should be planned before the system starts.
>
> In memcg, the amount of free memory is maintained by global LRU. It does much jobs
> for us. But free swap space isn't. It's just used on demand.
>
> If we can't decrease usage of swap by a group by hand, the problem which this
> swap-space-controller want to fix will not be fixed at pleasant level.
>
Thank you for your explanation.
I see your point and agree that the shrinking support is desireble.
I'll add it to my ToDo.

> Anyway, please return -EBUSY at setting limit < usage, at first :)
> That's enough for me, now.
>
Yes, again.

Thanks,
Daisuke Nishimura.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH -mm 0/5] swapcgroup (v3)
Posted by Daisuke Nishimura on Mon, 07 Jul 2008 06:48:30 GMT
View Forum Message <> Reply to Message

Hi, Balbir-san.

On Sat, 05 Jul 2008 12:22:25 +0530, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> Daisuke Nishimura wrote:
> > Hi.
> >
> > This is new version of swapcgroup.
> >
> > Major changes from previous version
> > - Rebased on 2.6.26-rc5-mm3.
> > The new -mm has been released, but these patches

> > can be applied on 2.6.26-rc8-mm1 too with only some offset warnings.
> > I tested these patches on 2.6.26-rc5-mm3 with some fixes about memory,
> > and it seems to work fine.
> > - (NEW) Implemented force_empty.
> > Currently, it simply uncharges all the charges from the group.
> >
> > Patches
> > - [1/5] add cgroup files
> > - [2/5] add a member to swap_info_struct
> > - [3/5] implement charge and uncharge
> > - [4/5] modify vm_swap_full()
> > - [5/5] implement force_empty
> >
> > ToDo(in my thought. Feel free to add some others here.)
> > - need some documentation
> > Add to memory.txt? or create a new documentation file?
> >
>
> I think memory.txt is good. But then, we'll need to add a Table of Contents to
> it, so that swap controller documentation can be located easily.
>
I think memory.txt is a self-closed documentation,
so I don't want to change it, honestlly.

I'll write a documentation for swap as a new file first for review.

> > - add option to disable only this feature
> > I'm wondering if this option is needed.
> > memcg has already the boot option to disable it.
> > Is there any case where memory should be accounted but swap should not?
> >
>
> That depends on what use case you are trying to provide. Let's say I needed
> backward compatibility with 2.6.25, then I would account for memory and leave
> out swap (even though we have swap controller).
>
O.K. I'll add option.

Thanks,
Daisuke Nishimura.

Subject: Re: [PATCH -mm 1/5] swapcgroup (v3): add cgroup files
Posted by Dave Hansen on Thu, 10 Jul 2008 20:35:36 GMT
View Forum Message <> Reply to Message

On Fri, 2008-07-04 at 15:17 +0900, Daisuke Nishimura wrote:
> +config CGROUP_SWAP_RES_CTLR
> + bool "Swap Resource Controller for Control Groups"
> + depends on CGROUP_MEM_RES_CTLR && SWAP
> + help
> + Provides a swap resource controller that manages and limits swap usage.
> + Implemented as a add-on to Memory Resource Controller.

Could you make this just plain depend on 'CGROUP_MEM_RES_CTLR && SWAP'
and not make it configurable? I don't think the resource usage really
justifies yet another .config knob to tune and break. :)

-- Dave

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [PATCH -mm 1/5] swapcgroup (v3): add cgroup files
Posted by Daisuke Nishimura on Fri, 11 Jul 2008 11:02:28 GMT
View Forum Message <> Reply to Message

Hi, Dave-san.

On Thu, 10 Jul 2008 13:35:36 -0700, Dave Hansen <dave@linux.vnet.ibm.com> wrote:
> On Fri, 2008-07-04 at 15:17 +0900, Daisuke Nishimura wrote:
> > +config CGROUP_SWAP_RES_CTLR
> > + bool "Swap Resource Controller for Control Groups"
> > + depends on CGROUP_MEM_RES_CTLR && SWAP
> > + help
> > + Provides a swap resource controller that manages and limits swap usage.
> > + Implemented as a add-on to Memory Resource Controller.
>
> Could you make this just plain depend on 'CGROUP_MEM_RES_CTLR && SWAP'
> and not make it configurable? I don't think the resource usage really
> justifies yet another .config knob to tune and break. :)
>

I don't stick to using kernel config option.

As I said in my ToDo, I'm going to implement another method
(boot option or something) to disable(or enable?) this feature,

so I can make this config not configurable after it.

Thanks,
Daisuke Nishimura.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers