Subject: [RFC][PATCH 0/7] Resource controllers based on process containers
Posted by xemul on Tue, 06 Mar 2007 14:42:52 GMT
View Forum Message <> Reply to Message

This patchset adds RSS, accounting and control and
limiting the number of tasks and files within container.

Based on top of Paul Menage's container subsystem v7

RSS controller includes per-container RSS accounter,
reclamation and OOM killer. It behaves like standalone
machine - when container runs out of resources it tries
to reclaim some pages and if it doesn't succeed in it
kills some task which mm_struct belongs to container in
question.

Num tasks and files containers are very simple and
self-descriptive from code.

As discussed before when a task moves from one container
to another no resources follow it - they keep holding the
container they were allocated in.

The difficulties met during using of Pauls' containers were:

1. Container fork hook is placed before new task
   changes. This makes impossible of handling fork
   properly. I.e. new mm_struct should have pointer
   to RSS container, but we don't have one at that
   early time.

2. Extended containers may register themselves too late.
   Kernel threads/helpers start forking, opening files
   and touching pages much earlier. This patchset
   workarounds this in not-so-cute manner and I'm waiting
   for Paul's comments on this issue.

Subject: [RFC][PATCH 1/7] Resource counters
Posted by xemul on Tue, 06 Mar 2007 14:47:25 GMT
View Forum Message <> Reply to Message

Introduce generic structures and routines for
resource accounting.

Each resource accounting container is supposed to
aggregate it, container_subsystem_state and its
resource-specific members within.

```
diff -upr linux-2.6.20.orig/include/linux/res_counter.h linux-2.6.20-0/include/linux/res_counter.h
--- linux-2.6.20.orig/include/linux/res_counter.h 2007-03-06 13:39:17.000000000 +0300
+++ linux-2.6.20-0/include/linux/res_counter.h 2007-03-06 13:33:28.000000000 +0300
@@ -0,0 +1,83 @@
+#ifndef __RES_COUNTER_H__
+#define __RES_COUNTER_H__
+/*
+ * resource counters
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+#include <linux/container.h>
+
+struct res_counter {
+ unsigned long usage;
+ unsigned long limit;
+ unsigned long failcnt;
+ spinlock_t lock;
+};
+
+enum {
+ RES_USAGE,
+ RES_LIMIT,
+ RES_FAILCNT,
+};
+
+ssize_t res_counter_read(struct res_counter *cnt, int member,
+  const char __user *buf, size_t nbytes, loff_t *pos);
+ssize_t res_counter_write(struct res_counter *cnt, int member,
+  const char __user *buf, size_t nbytes, loff_t *pos);
+
+static inline void res_counter_init(struct res_counter *cnt)
+{
+ spin_lock_init(&cnt->lock);
+ cnt->limit = (unsigned long)LONG_MAX;
+}
+
+static inline int res_counter_charge_locked(struct res_counter *cnt,
+  unsigned long val)
+{
+ if (cnt->usage <= cnt->limit - val) {
+  cnt->usage += val;
+  return 0;
```

```
+ }
+
+ cnt->failcnt++;
+ return -ENOMEM;
+}
+
+static inline int res_counter_charge(struct res_counter *cnt,
+  unsigned long val)
+{
+ int ret;
+ unsigned long flags;
+
+ spin_lock_irqsave(&cnt->lock, flags);
+ ret = res_counter_charge_locked(cnt, val);
+ spin_unlock_irqrestore(&cnt->lock, flags);
+ return ret;
+}
+
+static inline void res_counter_uncharge_locked(struct res_counter *cnt,
+  unsigned long val)
+{
+ if (unlikely(cnt->usage < val)) {
+  WARN_ON(1);
+  val = cnt->usage;
+ }
+
+ cnt->usage -= val;
+}
+
+static inline void res_counter_uncharge(struct res_counter *cnt,
+  unsigned long val)
+{
+ unsigned long flags;
+
+ spin_lock_irqsave(&cnt->lock, flags);
+ res_counter_uncharge_locked(cnt, val);
+ spin_unlock_irqrestore(&cnt->lock, flags);
+}
+
+#endif
diff -upr linux-2.6.20.orig/init/Kconfig linux-2.6.20-0/init/Kconfig
--- linux-2.6.20.orig/init/Kconfig 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/init/Kconfig 2007-03-06 13:33:28.000000000 +0300
@@ -265,6 +265,10 @@ config CPUSETS

    Say N if unsure.

+config RESOURCE_COUNTERS
```

```
+ bool
+ select CONTAINERS
+
 config SYSFS_DEPRECATED
  bool "Create deprecated sysfs files"
  default y
diff -upr linux-2.6.20.orig/kernel/Makefile linux-2.6.20-0/kernel/Makefile
--- linux-2.6.20.orig/kernel/Makefile 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/kernel/Makefile 2007-03-06 13:33:28.000000000 +0300
@@ -51,6 +51,7 @@ obj-$(CONFIG_RELAY) += relay.o
 obj-$(CONFIG_UTS_NS) += utsname.o
 obj-$(CONFIG_TASK_DELAY_ACCT) += delayacct.o
 obj-$(CONFIG_TASKSTATS) += taskstats.o tsacct.o
+obj-$(CONFIG_RESOURCE_COUNTERS) += res_counter.o

 ifneq ($(CONFIG_SCHED_NO_NO_OMIT_FRAME_POINTER),y)
 # According to Alan Modra <alan@linuxcare.com.au>, the -fno-omit-frame-pointer is
diff -upr linux-2.6.20.orig/kernel/res_counter.c linux-2.6.20-0/kernel/res_counter.c
--- linux-2.6.20.orig/kernel/res_counter.c 2007-03-06 13:39:17.000000000 +0300
+++ linux-2.6.20-0/kernel/res_counter.c 2007-03-06 13:33:28.000000000 +0300
@@ -0,0 +1,72 @@
+/*
+ * resource containers
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+#include <linux/parser.h>
+#include <linux/fs.h>
+#include <linux/res_counter.h>
+#include <asm/uaccess.h>
+
+static inline unsigned long *res_counter_member(struct res_counter *cnt, int member)
+{
+ switch (member) {
+ case RES_USAGE:
+  return &cnt->usage;
+ case RES_LIMIT:
+  return &cnt->limit;
+ case RES_FAILCNT:
+  return &cnt->failcnt;
+ };
+
+ BUG();
+ return NULL;
```

```
+}
+
+ssize_t res_counter_read(struct res_counter *cnt, int member,
+  const char __user *userbuf, size_t nbytes, loff_t *pos)
+{
+ unsigned long *val;
+ char buf[64], *s;
+
+ s = buf;
+ val = res_counter_member(cnt, member);
+ s += sprintf(s, "%lu\n", *val);
+ return simple_read_from_buffer((void __user *)userbuf, nbytes,
+   pos, buf, s - buf);
+}
+
+ssize_t res_counter_write(struct res_counter *cnt, int member,
+  const char __user *userbuf, size_t nbytes, loff_t *pos)
+{
+ int ret;
+ char *buf, *end;
+ unsigned long tmp, *val;
+
+ buf = kmalloc(nbytes + 1, GFP_KERNEL);
+ ret = -ENOMEM;
+ if (buf == NULL)
+  goto out;
+
+ buf[nbytes] = 0;
+ ret = -EFAULT;
+ if (copy_from_user(buf, userbuf, nbytes))
+  goto out_free;
+
+ ret = -EINVAL;
+ tmp = simple_strtoul(buf, &end, 10);
+ if (*end != '\0')
+  goto out_free;
+
+ val = res_counter_member(cnt, member);
+ *val = tmp;
+ ret = nbytes;
+out_free:
+ kfree(buf);
+out:
+ return ret;
+}
```

Subject: [RFC][PATCH 2/7] RSS controller core
Posted by xemul on Tue, 06 Mar 2007 14:53:26 GMT
View Forum Message <> Reply to Message

This includes setup of RSS container within generic
process containers, all the declarations used in RSS
accounting, and core code responsible for accounting.

```
diff -upr linux-2.6.20.orig/include/linux/rss_container.h linux-2.6.20-0/include/linux/rss_container.h
--- linux-2.6.20.orig/include/linux/rss_container.h 2007-03-06 13:39:17.000000000 +0300
+++ linux-2.6.20-0/include/linux/rss_container.h 2007-03-06 13:33:28.000000000 +0300
@@ -0,0 +1,68 @@
+#ifndef __RSS_CONTAINER_H__
+#define __RSS_CONTAINER_H__
+/*
+ * RSS container
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+struct page_container;
+struct rss_container;
+
+#ifdef CONFIG_RSS_CONTAINER
+int container_rss_prepare(struct page *, struct vm_area_struct *vma,
+  struct page_container **);
+
+void container_rss_add(struct page_container *);
+void container_rss_del(struct page_container *);
+void container_rss_release(struct page_container *);
+
+int mm_init_container(struct mm_struct *mm, struct task_struct *tsk);
+void mm_free_container(struct mm_struct *mm);
+
+unsigned long container_isolate_pages(unsigned long nr_to_scan,
+  struct rss_container *rss, struct list_head *dst,
+  int active, unsigned long *scanned);
+unsigned long container_nr_physpages(struct rss_container *rss);
+
+unsigned long container_try_to_free_pages(struct rss_container *);
+void container_out_of_memory(struct rss_container *);
+
+void container_rss_init_early(void);
+#else
+static inline int container_rss_prepare(struct page *pg,
+  struct vm_area_struct *vma, struct page_container **pc)
```

```
+{
+ *pc = NULL; /* to make gcc happy */
+ return 0;
+}
+
+static inline void container_rss_add(struct page_container *pc)
+{
+}
+
+static inline void container_rss_del(struct page_container *pc)
+{
+}
+
+static inline void container_rss_release(struct page_container *pc)
+{
+}
+
+static inline int mm_init_container(struct mm_struct *mm, struct task_struct *t)
+{
+ return 0;
+}
+
+static inline void mm_free_container(struct mm_struct *mm)
+{
+}
+
+static inline void container_rss_init_early(void)
+{
+}
+#endif
+#endif
diff -upr linux-2.6.20.orig/init/Kconfig linux-2.6.20-0/init/Kconfig
--- linux-2.6.20.orig/init/Kconfig 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/init/Kconfig 2007-03-06 13:33:28.000000000 +0300
@@ -265,6 +265,13 @@ config CPUSETS
  bool
  select CONTAINERS

+config RSS_CONTAINER
+ bool "RSS accounting container"
+ select RESOURCE_COUNTERS
+ help
+   Provides a simple Resource Controller for monitoring and
+   controlling the total Resident Set Size of the tasks in a container
+
 config SYSFS_DEPRECATED
  bool "Create deprecated sysfs files"
  default y
```

```
diff -upr linux-2.6.20.orig/mm/Makefile linux-2.6.20-0/mm/Makefile
--- linux-2.6.20.orig/mm/Makefile 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/mm/Makefile 2007-03-06 13:33:28.000000000 +0300
@@ -29,3 +29,5 @@ obj-$(CONFIG_MEMORY_HOTPLUG) += memory_h
 obj-$(CONFIG_FS_XIP) += filemap_xip.o
 obj-$(CONFIG_MIGRATION) += migrate.o
 obj-$(CONFIG_SMP) += allocpercpu.o
+
+obj-$(CONFIG_RSS_CONTAINER) += rss_container.o
diff -upr linux-2.6.20.orig/mm/rss_container.c linux-2.6.20-0/mm/rss_container.c
--- linux-2.6.20.orig/mm/rss_container.c 2007-03-06 13:39:17.000000000 +0300
+++ linux-2.6.20-0/mm/rss_container.c 2007-03-06 13:33:28.000000000 +0300
@@ -0,0 +1,307 @@
+/*
+ * RSS accounting container
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+#include <linux/list.h>
+#include <linux/sched.h>
+#include <linux/mm.h>
+#include <linux/res_counter.h>
+#include <linux/rss_container.h>
+
+static struct container_subsys rss_subsys;
+
+struct rss_container {
+ struct res_counter res;
+ struct list_head page_list;
+ struct container_subsys_state css;
+};
+
+struct page_container {
+ struct page *page;
+ struct rss_container *cnt;
+ struct list_head list;
+};
+
+static inline struct rss_container *rss_from_cont(struct container *cnt)
+{
+ return container_of(container_subsys_state(cnt, &rss_subsys),
+   struct rss_container, css);
+}
+
```

```
+int mm_init_container(struct mm_struct *mm, struct task_struct *tsk)
+{
+ struct rss_container *cnt;
+
+ cnt = rss_from_cont(task_container(tsk, &rss_subsys));
+ if (css_get(&cnt->css))
+  return -EBUSY;
+
+ mm->rss_container = cnt;
+ return 0;
+}
+
+void mm_free_container(struct mm_struct *mm)
+{
+ css_put(&mm->rss_container->css);
+}
+
+int container_rss_prepare(struct page *page, struct vm_area_struct *vma,
+  struct page_container **ppc)
+{
+ struct rss_container *rss;
+ struct page_container *pc;
+
+ rcu_read_lock();
+ rss = rcu_dereference(vma->vm_mm->rss_container);
+ css_get_current(&rss->css);
+ rcu_read_unlock();
+
+ pc = kmalloc(sizeof(struct page_container), GFP_KERNEL);
+ if (pc == NULL)
+  goto out_nomem;
+
+ while (res_counter_charge(&rss->res, 1)) {
+  if (container_try_to_free_pages(rss))
+   continue;
+
+  container_out_of_memory(rss);
+  if (test_thread_flag(TIF_MEMDIE))
+   goto out_charge;
+ }
+
+ pc->page = page;
+ pc->cnt = rss;
+ *ppc = pc;
+ return 0;
+
+out_charge:
+ kfree(pc);
```

```
+out_nomem:
+ css_put(&rss->css);
+ return -ENOMEM;
+}
+
+void container_rss_release(struct page_container *pc)
+{
+ struct rss_container *rss;
+
+ rss = pc->cnt;
+ res_counter_uncharge(&rss->res, 1);
+ css_put(&rss->css);
+ kfree(pc);
+}
+
+void container_rss_add(struct page_container *pc)
+{
+ struct page *pg;
+ struct rss_container *rss;
+
+ pg = pc->page;
+ rss = pc->cnt;
+
+ spin_lock(&rss->res.lock);
+ list_add(&pc->list, &rss->page_list);
+ spin_unlock(&rss->res.lock);
+
+ page_container(pg) = pc;
+}
+
+void container_rss_del(struct page_container *pc)
+{
+ struct page *page;
+ struct rss_container *rss;
+
+ page = pc->page;
+ rss = pc->cnt;
+
+ spin_lock(&rss->res.lock);
+ list_del(&pc->list);
+ res_counter_uncharge_locked(&rss->res, 1);
+ spin_unlock(&rss->res.lock);
+
+ css_put(&rss->css);
+ kfree(pc);
+}
+
+unsigned long container_isolate_pages(unsigned long nr_to_scan,
```

```
+  struct rss_container *rss, struct list_head *dst,
+  int active, unsigned long *scanned)
+{
+ unsigned long nr_taken = 0;
+ struct page *page;
+ struct page_container *pc;
+ unsigned long scan;
+ struct list_head *src;
+ LIST_HEAD(pc_list);
+ struct zone *z;
+
+ spin_lock_irq(&rss->res.lock);
+ src = &rss->page_list;
+
+ for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
+  pc = list_entry(src->prev, struct page_container, list);
+  page = pc->page;
+  z = page_zone(page);
+
+  list_move(&pc->list, &pc_list);
+
+  spin_lock(&z->lru_lock);
+  if (PageLRU(page)) {
+   if ((active && PageActive(page)) ||
+     (!active && !PageActive(page))) {
+    if (likely(get_page_unless_zero(page))) {
+     ClearPageLRU(page);
+     nr_taken++;
+     list_move(&page->lru, dst);
+    }
+   }
+  }
+  spin_unlock(&z->lru_lock);
+ }
+
+ list_splice(&pc_list, src);
+ spin_unlock_irq(&rss->res.lock);
+
+ *scanned = scan;
+ return nr_taken;
+}
+
+unsigned long container_nr_physpages(struct rss_container *rss)
+{
+ return rss->res.usage;
+}
+
+static void rss_move_task(struct container_subsys *ss,
```

```
+  struct container *cont,
+  struct container *old_cont,
+  struct task_struct *p)
+{
+ struct mm_struct *mm;
+ struct rss_container *rss, *old_rss;
+
+ mm = get_task_mm(p);
+ if (mm == NULL)
+  goto out;
+
+ rss = rss_from_cont(cont);
+ old_rss = rss_from_cont(old_cont);
+ if (old_rss != mm->rss_container)
+  goto out_put;
+
+ css_get_current(&rss->css);
+ rcu_assign_pointer(mm->rss_container, rss);
+ css_put(&old_rss->css);
+
+out_put:
+ mmput(mm);
+out:
+ return;
+}
+
+static int rss_create(struct container_subsys *ss, struct container *cont)
+{
+ struct rss_container *rss;
+
+ rss = kzalloc(sizeof(struct rss_container), GFP_KERNEL);
+ if (rss == NULL)
+  return -ENOMEM;
+
+ res_counter_init(&rss->res);
+ INIT_LIST_HEAD(&rss->page_list);
+ cont->subsys[rss_subsys.subsys_id] = &rss->css;
+ return 0;
+}
+
+static void rss_destroy(struct container_subsys *ss,
+  struct container *cont)
+{
+ kfree(rss_from_cont(cont));
+}
+
+
+static ssize_t rss_read(struct container *cont, struct cftype *cft,
```

```
+  struct file *file, char __user *userbuf,
+  size_t nbytes, loff_t *ppos)
+{
+ return res_counter_read(&rss_from_cont(cont)->res, cft->private,
+   userbuf, nbytes, ppos);
+}
+
+static ssize_t rss_write(struct container *cont, struct cftype *cft,
+  struct file *file, const char __user *userbuf,
+  size_t nbytes, loff_t *ppos)
+{
+ return res_counter_write(&rss_from_cont(cont)->res, cft->private,
+   userbuf, nbytes, ppos);
+}
+
+
+static struct cftype rss_usage = {
+ .name = "rss_usage",
+ .private = RES_USAGE,
+ .read = rss_read,
+};
+
+static struct cftype rss_limit = {
+ .name = "rss_limit",
+ .private = RES_LIMIT,
+ .read = rss_read,
+ .write = rss_write,
+};
+
+static struct cftype rss_failcnt = {
+ .name = "rss_failcnt",
+ .private = RES_FAILCNT,
+ .read = rss_read,
+};
+
+static int rss_populate(struct container_subsys *ss,
+  struct container *cont)
+{
+ int rc;
+
+ if ((rc = container_add_file(cont, &rss_usage)) < 0)
+  return rc;
+ if ((rc = container_add_file(cont, &rss_failcnt)) < 0)
+  return rc;
+ if ((rc = container_add_file(cont, &rss_limit)) < 0)
+  return rc;
+
+ return 0;
```

```
+}
+
+static struct rss_container init_rss_container;
+
+static __init int rss_create_early(struct container_subsys *ss,
+  struct container *cont)
+{
+ struct rss_container *rss;
+
+ rss = &init_rss_container;
+ res_counter_init(&rss->res);
+ INIT_LIST_HEAD(&rss->page_list);
+ cont->subsys[rss_subsys.subsys_id] = &rss->css;
+ ss->create = rss_create;
+ return 0;
+}
+
+static struct container_subsys rss_subsys = {
+ .name = "rss",
+ .create = rss_create_early,
+ .destroy = rss_destroy,
+ .populate = rss_populate,
+ .attach = rss_move_task,
+};
+
+void __init container_rss_init_early(void)
+{
+ container_register_subsys(&rss_subsys);
+ init_mm.rss_container = rss_from_cont(
+   task_container(&init_task, &rss_subsys));
+ css_get_current(&init_mm.rss_container->css);
+}
```

Subject: [RFC][PATCH 3/7] Data structures changes for RSS accounting
Posted by xemul on Tue, 06 Mar 2007 14:55:46 GMT
View Forum Message <> Reply to Message

Adds needed pointers to mm_struct and page struct,
places hooks to core code for mm_struct initialization
and hooks in container_init_early() to preinitialize
RSS accounting subsystem.

```
diff -upr linux-2.6.20.orig/include/linux/mm.h linux-2.6.20-0/include/linux/mm.h
--- linux-2.6.20.orig/include/linux/mm.h 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/include/linux/mm.h 2007-03-06 13:33:28.000000000 +0300
@@ -220,6 +220,12 @@ struct vm_operations_struct {
 struct mmu_gather;
```

```
 struct inode;

+#ifdef CONFIG_RSS_CONTAINER
+#define page_container(page) (page->rss_container)
+#else
+#define page_container(page) (NULL)
+#endif
+
 #define page_private(page)  ((page)->private)
 #define set_page_private(page, v) ((page)->private = (v))

diff -upr linux-2.6.20.orig/include/linux/mm_types.h linux-2.6.20-0/include/linux/mm_types.h
--- linux-2.6.20.orig/include/linux/mm_types.h 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/include/linux/mm_types.h 2007-03-06 13:33:28.000000000 +0300
@@ -62,6 +62,9 @@ struct page {
  void *virtual;   /* Kernel virtual address (NULL if
        not kmapped, ie. highmem) */
 #endif /* WANT_PAGE_VIRTUAL */
+#ifdef CONFIG_RSS_CONTAINER
+ struct page_container *rss_container;
+#endif
 };

 #endif /* _LINUX_MM_TYPES_H */
diff -upr linux-2.6.20.orig/include/linux/sched.h linux-2.6.20-0/include/linux/sched.h
--- linux-2.6.20.orig/include/linux/sched.h 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/include/linux/sched.h 2007-03-06 13:33:28.000000000 +0300
@@ -373,6 +373,9 @@ struct mm_struct {
  /* aio bits */
  rwlock_t  ioctx_list_lock;
  struct kioctx  *ioctx_list;
+#ifdef CONFIG_RSS_CONTAINER
+ struct rss_container *rss_container;
+#endif
 };

 struct sighand_struct {
diff -upr linux-2.6.20.orig/kernel/fork.c linux-2.6.20-0/kernel/fork.c
--- linux-2.6.20.orig/kernel/fork.c 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/kernel/fork.c 2007-03-06 13:33:28.000000000 +0300
@@ -57,6 +57,8 @@
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>

+#include <linux/rss_container.h>
+
 /*
  * Protected counters by write_lock_irq(&tasklist_lock)
```

```
 */
@@ -325,7 +328,7 @@ static inline void mm_free_pgd(struct mm

 #include <linux/init_task.h>

-static struct mm_struct * mm_init(struct mm_struct * mm)
+static struct mm_struct * mm_init(struct mm_struct *mm, struct task_struct *tsk)
 {
  atomic_set(&mm->mm_users, 1);
  atomic_set(&mm->mm_count, 1);
@@ -341,10 +344,18 @@ static struct mm_struct * mm_init(struct
  mm->free_area_cache = TASK_UNMAPPED_BASE;
  mm->cached_hole_size = ~0UL;

- if (likely(!mm_alloc_pgd(mm))) {
-  mm->def_flags = 0;
-  return mm;
- }
+ if (unlikely(mm_init_container(mm, tsk)))
+  goto out_cont;
+
+ if (unlikely(mm_alloc_pgd(mm)))
+  goto out_pgd;
+
+ mm->def_flags = 0;
+ return mm;
+
+out_pgd:
+ mm_free_container(mm);
+out_cont:
  free_mm(mm);
  return NULL;
 }
@@ -359,7 +370,7 @@ struct mm_struct * mm_alloc(void)
  mm = allocate_mm();
  if (mm) {
   memset(mm, 0, sizeof(*mm));
-  mm = mm_init(mm);
+  mm = mm_init(mm, current);
  }
  return mm;
 }
@@ -373,6 +384,7 @@ void fastcall __mmdrop(struct mm_struct
 {
  BUG_ON(mm == &init_mm);
  mm_free_pgd(mm);
+ mm_free_container(mm);
  destroy_context(mm);
```

```
  free_mm(mm);
 }
@@ -493,7 +505,7 @@ static struct mm_struct *dup_mm(struct t
  mm->token_priority = 0;
  mm->last_interval = 0;

- if (!mm_init(mm))
+ if (!mm_init(mm, tsk))
   goto fail_nomem;

  if (init_new_context(tsk, mm))
@@ -520,6 +532,7 @@ fail_nocontext:
   * because it calls destroy_context()
   */
  mm_free_pgd(mm);
+ mm_free_container(mm);
  free_mm(mm);
  return NULL;
 }
diff -upr linux-2.6.20.orig/kernel/container.c linux-2.6.20-0/kernel/container.c
--- linux-2.6.20.orig/kernel/container.c 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/kernel/container.c 2007-03-06 13:35:48.000000000 +0300
@@ -60,6 +60,8 @@
 #include <asm/atomic.h>
 #include <linux/mutex.h>

+#include <linux/rss_container.h>
+
 #define CONTAINER_SUPER_MAGIC  0x27e0eb

 static struct container_subsys *subsys[CONFIG_MAX_CONTAINER_SUBSYS];
@@ -1721,6 +1725,8 @@ int __init container_init_early(void)
  }
  init_task.containers = &init_container_group;

+ container_rss_init_early();
+
  return 0;
 }
```

Subject: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by xemul on Tue, 06 Mar 2007 14:57:51 GMT
View Forum Message <> Reply to Message

Pages are charged to their first touchers which are
determined using pages' mapcount manipulations in
rmap calls.

```
diff -upr linux-2.6.20.orig/fs/exec.c linux-2.6.20-0/fs/exec.c
--- linux-2.6.20.orig/fs/exec.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/fs/exec.c 2007-03-06 13:33:28.000000000 +0300
@@ -58,6 +58,8 @@
 #include <linux/kmod.h>
 #endif

+#include <linux/rss_container.h>
+
 int core_uses_pid;
 char core_pattern[128] = "core";
 int suid_dumpable = 0;
@@ -309,27 +311,34 @@ void install_arg_page(struct vm_area_str
  struct mm_struct *mm = vma->vm_mm;
  pte_t * pte;
  spinlock_t *ptl;
+ struct page_container *pcont;

  if (unlikely(anon_vma_prepare(vma)))
   goto out;

+ if (container_rss_prepare(page, vma, &pcont))
+  goto out;
+
  flush_dcache_page(page);
  pte = get_locked_pte(mm, address, &ptl);
  if (!pte)
-  goto out;
+  goto out_release;
  if (!pte_none(*pte)) {
   pte_unmap_unlock(pte, ptl);
-  goto out;
+  goto out_release;
  }
  inc_mm_counter(mm, anon_rss);
  lru_cache_add_active(page);
  set_pte_at(mm, address, pte, pte_mkdirty(pte_mkwrite(mk_pte(
      page, vma->vm_page_prot))));
- page_add_new_anon_rmap(page, vma, address);
+ page_add_new_anon_rmap(page, vma, address, pcont);
  pte_unmap_unlock(pte, ptl);

  /* no need for flush_tlb */
  return;
+
+out_release:
+ container_rss_release(pcont);
```

```
 out:
  __free_page(page);
  force_sig(SIGKILL, current);
diff -upr linux-2.6.20.orig/include/linux/rmap.h linux-2.6.20-0/include/linux/rmap.h
--- linux-2.6.20.orig/include/linux/rmap.h 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/include/linux/rmap.h 2007-03-06 13:33:28.000000000 +0300
@@ -69,9 +69,13 @@ void __anon_vma_link(struct vm_area_stru
 /*
  * rmap interfaces called when adding or removing pte of page
  */
-void page_add_anon_rmap(struct page *, struct vm_area_struct *, unsigned long);
-void page_add_new_anon_rmap(struct page *, struct vm_area_struct *, unsigned long);
-void page_add_file_rmap(struct page *);
+struct page_container;
+
+void page_add_anon_rmap(struct page *, struct vm_area_struct *,
+  unsigned long, struct page_container *);
+void page_add_new_anon_rmap(struct page *, struct vm_area_struct *,
+  unsigned long, struct page_container *);
+void page_add_file_rmap(struct page *, struct page_container *);
 void page_remove_rmap(struct page *, struct vm_area_struct *);

 /**
diff -upr linux-2.6.20.orig/mm/fremap.c linux-2.6.20-0/mm/fremap.c
--- linux-2.6.20.orig/mm/fremap.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/mm/fremap.c 2007-03-06 13:33:28.000000000 +0300
@@ -20,6 +20,8 @@
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>

+#include <linux/rss_container.h>
+
 static int zap_pte(struct mm_struct *mm, struct vm_area_struct *vma,
    unsigned long addr, pte_t *ptep)
 {
@@ -57,6 +59,10 @@ int install_page(struct mm_struct *mm, s
  pte_t *pte;
  pte_t pte_val;
  spinlock_t *ptl;
+ struct page_container *pcont;
+
+ if (container_rss_prepare(page, vma, &pcont))
+  goto out_release;

  pte = get_locked_pte(mm, addr, &ptl);
  if (!pte)
@@ -81,13 +87,16 @@ int install_page(struct mm_struct *mm, s
  flush_icache_page(vma, page);
```

```
  pte_val = mk_pte(page, prot);
  set_pte_at(mm, addr, pte, pte_val);
- page_add_file_rmap(page);
+ page_add_file_rmap(page, pcont);
  update_mmu_cache(vma, addr, pte_val);
  lazy_mmu_prot_update(pte_val);
  err = 0;
 unlock:
  pte_unmap_unlock(pte, ptl);
 out:
+ if (err != 0)
+  container_rss_release(pcont);
+out_release:
  return err;
 }
 EXPORT_SYMBOL(install_page);
diff -upr linux-2.6.20.orig/mm/memory.c linux-2.6.20-0/mm/memory.c
--- linux-2.6.20.orig/mm/memory.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/mm/memory.c 2007-03-06 13:33:28.000000000 +0300
@@ -60,6 +60,8 @@
 #include <linux/swapops.h>
 #include <linux/elf.h>

+#include <linux/rss_container.h>
+
 #ifndef CONFIG_NEED_MULTIPLE_NODES
 /* use the per-pgdat data instead for discontigmem - mbligh */
 unsigned long max_mapnr;
@@ -1126,7 +1128,7 @@ static int zeromap_pte_range(struct mm_s
    break;
   }
   page_cache_get(page);
-  page_add_file_rmap(page);
+  page_add_file_rmap(page, NULL);
   inc_mm_counter(mm, file_rss);
   set_pte_at(mm, addr, pte, zero_pte);
  } while (pte++, addr += PAGE_SIZE, addr != end);
@@ -1234,7 +1236,7 @@ static int insert_page(struct mm_struct
  /* Ok, finally just insert the thing.. */
  get_page(page);
  inc_mm_counter(mm, file_rss);
- page_add_file_rmap(page);
+ page_add_file_rmap(page, NULL);
  set_pte_at(mm, addr, pte, mk_pte(page, prot));

  retval = 0;
@@ -1495,6 +1497,7 @@ static int do_wp_page(struct mm_struct *
  pte_t entry;
```

```
  int reuse = 0, ret = VM_FAULT_MINOR;
  struct page *dirty_page = NULL;
+ struct page_container *pcont;

  old_page = vm_normal_page(vma, address, orig_pte);
  if (!old_page)
@@ -1580,6 +1583,9 @@ gotten:
   cow_user_page(new_page, old_page, address, vma);
  }

+ if (container_rss_prepare(new_page, vma, &pcont))
+  goto oom;
+
  /*
   * Re-check the pte - we dropped the lock
   */
@@ -1607,12 +1613,14 @@ gotten:
   set_pte_at(mm, address, page_table, entry);
   update_mmu_cache(vma, address, entry);
   lru_cache_add_active(new_page);
-  page_add_new_anon_rmap(new_page, vma, address);
+  page_add_new_anon_rmap(new_page, vma, address, pcont);

   /* Free the old page.. */
   new_page = old_page;
   ret |= VM_FAULT_WRITE;
- }
+ } else
+  container_rss_release(pcont);
+
  if (new_page)
   page_cache_release(new_page);
  if (old_page)
@@ -1988,6 +1996,7 @@ static int do_swap_page(struct mm_struct
  swp_entry_t entry;
  pte_t pte;
  int ret = VM_FAULT_MINOR;
+ struct page_container *pcont;

  if (!pte_unmap_same(mm, pmd, page_table, orig_pte))
   goto out;
@@ -2020,6 +2029,11 @@ static int do_swap_page(struct mm_struct
   count_vm_event(PGMAJFAULT);
  }

+ if (container_rss_prepare(page, vma, &pcont)) {
+  ret = VM_FAULT_OOM;
+  goto out;
```

```
+ }
+
  delayacct_clear_flag(DELAYACCT_PF_SWAPIN);
  mark_page_accessed(page);
  lock_page(page);
@@ -2033,6 +2047,7 @@ static int do_swap_page(struct mm_struct

  if (unlikely(!PageUptodate(page))) {
   ret = VM_FAULT_SIGBUS;
+  container_rss_release(pcont);
   goto out_nomap;
  }

@@ -2047,7 +2062,7 @@ static int do_swap_page(struct mm_struct

  flush_icache_page(vma, page);
  set_pte_at(mm, address, page_table, pte);
- page_add_anon_rmap(page, vma, address);
+ page_add_anon_rmap(page, vma, address, pcont);

  swap_free(entry);
  if (vm_swap_full())
@@ -2069,6 +2084,7 @@ unlock:
 out:
  return ret;
 out_nomap:
+ container_rss_release(pcont);
  pte_unmap_unlock(page_table, ptl);
  unlock_page(page);
  page_cache_release(page);
@@ -2087,6 +2103,7 @@ static int do_anonymous_page(struct mm_s
  struct page *page;
  spinlock_t *ptl;
  pte_t entry;
+ struct page_container *pcont;

  if (write_access) {
   /* Allocate our own private page. */
@@ -2098,15 +2115,19 @@ static int do_anonymous_page(struct mm_s
   if (!page)
    goto oom;

+  if (container_rss_prepare(page, vma, &pcont))
+   goto oom_release;
+
   entry = mk_pte(page, vma->vm_page_prot);
   entry = maybe_mkwrite(pte_mkdirty(entry), vma);
```

```
   page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
   if (!pte_none(*page_table))
-   goto release;
+   goto release_container;
+
   inc_mm_counter(mm, anon_rss);
   lru_cache_add_active(page);
-  page_add_new_anon_rmap(page, vma, address);
+  page_add_new_anon_rmap(page, vma, address, pcont);
  } else {
   /* Map the ZERO_PAGE - vm_page_prot is readonly */
   page = ZERO_PAGE(address);
@@ -2118,7 +2139,7 @@ static int do_anonymous_page(struct mm_s
   if (!pte_none(*page_table))
    goto release;
   inc_mm_counter(mm, file_rss);
-  page_add_file_rmap(page);
+  page_add_file_rmap(page, NULL);
  }

  set_pte_at(mm, address, page_table, entry);
@@ -2129,9 +2150,14 @@ static int do_anonymous_page(struct mm_s
 unlock:
  pte_unmap_unlock(page_table, ptl);
  return VM_FAULT_MINOR;
+release_container:
+ container_rss_release(pcont);
 release:
  page_cache_release(page);
  goto unlock;
+
+oom_release:
+ page_cache_release(page);
 oom:
  return VM_FAULT_OOM;
 }
@@ -2161,6 +2187,7 @@ static int do_no_page(struct mm_struct *
  int ret = VM_FAULT_MINOR;
  int anon = 0;
  struct page *dirty_page = NULL;
+ struct page_container *pcont;

  pte_unmap(page_table);
  BUG_ON(vma->vm_flags & VM_PFNMAP);
@@ -2218,6 +2245,9 @@ retry:
   }
  }
```

```
+ if (container_rss_prepare(new_page, vma, &pcont))
+  goto oom;
+
  page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
  /*
   * For a file-backed vma, someone could have truncated or otherwise
@@ -2226,6 +2256,7 @@ retry:
   */
  if (mapping && unlikely(sequence != mapping->truncate_count)) {
   pte_unmap_unlock(page_table, ptl);
+  container_rss_release(pcont);
   page_cache_release(new_page);
   cond_resched();
   sequence = mapping->truncate_count;
@@ -2253,10 +2284,10 @@ retry:
   if (anon) {
    inc_mm_counter(mm, anon_rss);
    lru_cache_add_active(new_page);
-   page_add_new_anon_rmap(new_page, vma, address);
+   page_add_new_anon_rmap(new_page, vma, address, pcont);
   } else {
    inc_mm_counter(mm, file_rss);
-   page_add_file_rmap(new_page);
+   page_add_file_rmap(new_page, pcont);
    if (write_access) {
     dirty_page = new_page;
     get_page(dirty_page);
@@ -2264,6 +2295,7 @@ retry:
   }
  } else {
   /* One of our sibling threads was faster, back out. */
+  container_rss_release(pcont);
   page_cache_release(new_page);
   goto unlock;
  }
diff -upr linux-2.6.20.orig/mm/migrate.c linux-2.6.20-0/mm/migrate.c
--- linux-2.6.20.orig/mm/migrate.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/mm/migrate.c 2007-03-06 13:33:28.000000000 +0300
@@ -134,6 +134,7 @@ static void remove_migration_pte(struct
  pte_t *ptep, pte;
  spinlock_t *ptl;
  unsigned long addr = page_address_in_vma(new, vma);
+ struct page_container *pcont;

  if (addr == -EFAULT)
   return;
@@ -157,6 +158,11 @@ static void remove_migration_pte(struct
   return;
```

```
   }

+ if (container_rss_prepare(new, vma, &pcont)) {
+  pte_unmap(ptep);
+  return;
+ }
+
   ptl = pte_lockptr(mm, pmd);
   spin_lock(ptl);
  pte = *ptep;
@@ -175,16 +181,19 @@ static void remove_migration_pte(struct
  set_pte_at(mm, addr, ptep, pte);

  if (PageAnon(new))
-  page_add_anon_rmap(new, vma, addr);
+  page_add_anon_rmap(new, vma, addr, pcont);
  else
-  page_add_file_rmap(new);
+  page_add_file_rmap(new, pcont);

  /* No need to invalidate - it was non-present before */
  update_mmu_cache(vma, addr, pte);
  lazy_mmu_prot_update(pte);
+ pte_unmap_unlock(ptep, ptl);
+ return;

 out:
  pte_unmap_unlock(ptep, ptl);
+ container_rss_release(pcont);
 }

 /*
diff -upr linux-2.6.20.orig/mm/rmap.c linux-2.6.20-0/mm/rmap.c
--- linux-2.6.20.orig/mm/rmap.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/mm/rmap.c 2007-03-06 13:33:28.000000000 +0300
@@ -51,6 +51,8 @@

 #include <asm/tlbflush.h>

+#include <linux/rss_container.h>
+
 struct kmem_cache *anon_vma_cachep;

 static inline void validate_anon_vma(struct vm_area_struct *find_vma)
@@ -526,14 +528,19 @@ static void __page_set_anon_rmap(struct
  * @page: the page to add the mapping to
  * @vma: the vm area in which the mapping is added
  * @address: the user virtual address mapped
```

```
+ * @pcont: the page beancounter to charge page with
  *
  * The caller needs to hold the pte lock.
  */
 void page_add_anon_rmap(struct page *page,
- struct vm_area_struct *vma, unsigned long address)
+ struct vm_area_struct *vma, unsigned long address,
+ struct page_container *pcont)
 {
- if (atomic_inc_and_test(&page->_mapcount))
+ if (atomic_inc_and_test(&page->_mapcount)) {
+  container_rss_add(pcont);
   __page_set_anon_rmap(page, vma, address);
+ } else
+  container_rss_release(pcont);
  /* else checking page index and mapping is racy */
 }

@@ -542,27 +549,35 @@ void page_add_anon_rmap(struct page *pag
  * @page: the page to add the mapping to
  * @vma: the vm area in which the mapping is added
  * @address: the user virtual address mapped
+ * @pcont: the page beancounter to charge page with
  *
  * Same as page_add_anon_rmap but must only be called on *new* pages.
  * This means the inc-and-test can be bypassed.
  */
 void page_add_new_anon_rmap(struct page *page,
- struct vm_area_struct *vma, unsigned long address)
+ struct vm_area_struct *vma, unsigned long address,
+ struct page_container *pcont)
 {
  atomic_set(&page->_mapcount, 0); /* elevate count by 1 (starts at -1) */
+ container_rss_add(pcont);
  __page_set_anon_rmap(page, vma, address);
 }

 /**
  * page_add_file_rmap - add pte mapping to a file page
- * @page: the page to add the mapping to
+ * @page: the page to add the mapping to
+ * @pcont: the page beancounter to charge page with
  *
  * The caller needs to hold the pte lock.
  */
-void page_add_file_rmap(struct page *page)
+void page_add_file_rmap(struct page *page, struct page_container *pcont)
 {
```

```
- if (atomic_inc_and_test(&page->_mapcount))
+ if (atomic_inc_and_test(&page->_mapcount)) {
+  if (pcont)
+   container_rss_add(pcont);
   __inc_zone_page_state(page, NR_FILE_MAPPED);
+ } else if (pcont)
+  container_rss_release(pcont);
 }

 /**
@@ -573,6 +588,9 @@ void page_add_file_rmap(struct page *pag
  */
 void page_remove_rmap(struct page *page, struct vm_area_struct *vma)
 {
+ struct page_container *pcont;
+
+ pcont = page_container(page);
  if (atomic_add_negative(-1, &page->_mapcount)) {
   if (unlikely(page_mapcount(page) < 0)) {
    printk (KERN_EMERG "Eeek! page_mapcount(page) went negative! (%d)\n",
page_mapcount(page));
@@ -588,6 +606,8 @@ void page_remove_rmap(struct page *page,
    BUG();
   }

+  if (pcont)
+   container_rss_del(pcont);
   /*
    * It would be tidy to reset the PageAnon mapping here,
    * but that might overwrite a racing page_add_anon_rmap
diff -upr linux-2.6.20.orig/mm/swapfile.c linux-2.6.20-0/mm/swapfile.c
--- linux-2.6.20.orig/mm/swapfile.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/mm/swapfile.c 2007-03-06 13:33:28.000000000 +0300
@@ -32,6 +32,8 @@
 #include <asm/tlbflush.h>
 #include <linux/swapops.h>

+#include <linux/rss_container.h>
+
 DEFINE_SPINLOCK(swap_lock);
 unsigned int nr_swapfiles;
 long total_swap_pages;
@@ -507,13 +509,14 @@ unsigned int count_swap_pages(int type,
  * force COW, vm_page_prot omits write permission from any private vma.
  */
 static void unuse_pte(struct vm_area_struct *vma, pte_t *pte,
-  unsigned long addr, swp_entry_t entry, struct page *page)
+  unsigned long addr, swp_entry_t entry, struct page *page,
```

```
+  struct page_container *pcont)
 {
  inc_mm_counter(vma->vm_mm, anon_rss);
  get_page(page);
  set_pte_at(vma->vm_mm, addr, pte,
      pte_mkold(mk_pte(page, vma->vm_page_prot)));
- page_add_anon_rmap(page, vma, addr);
+ page_add_anon_rmap(page, vma, addr, pcont);
  swap_free(entry);
  /*
   * Move the page to the active list so it is not
@@ -530,6 +533,10 @@ static int unuse_pte_range(struct vm_are
  pte_t *pte;
  spinlock_t *ptl;
  int found = 0;
+ struct page_container *pcont;
+
+ if (container_rss_prepare(page, vma, &pcont))
+  return 0;

  pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
  do {
@@ -538,12 +545,14 @@ static int unuse_pte_range(struct vm_are
    * Test inline before going to call unuse_pte.
    */
   if (unlikely(pte_same(*pte, swp_pte))) {
-   unuse_pte(vma, pte++, addr, entry, page);
+   unuse_pte(vma, pte++, addr, entry, page, pcont);
    found = 1;
    break;
   }
  } while (pte++, addr += PAGE_SIZE, addr != end);
  pte_unmap_unlock(pte - 1, ptl);
+ if (!found)
+  container_rss_release(pcont);
  return found;
 }
```

Subject: [RFC][PATCH 5/7] Per-container OOM killer and page reclamation
Posted by xemul on Tue, 06 Mar 2007 15:01:05 GMT

View Forum Message <> Reply to Message

```
* container_try_to_free_pages() walks containers
  page list and tries to shrink pages. This is based
  on try_to_free_pages() and Co code.
  Called from core code when no resource left at the
  moment of page touching.
```

```
* container_out_of_memory() selects a process to be
  killed which mm_struct belongs to container in question.
  Called from core code when no resources left and no
  pages were reclaimed.

diff -upr linux-2.6.20.orig/mm/oom_kill.c linux-2.6.20-0/mm/oom_kill.c
--- linux-2.6.20.orig/mm/oom_kill.c 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/mm/oom_kill.c 2007-03-06 13:33:28.000000000 +0300
@@ -24,6 +24,7 @@
 #include <linux/cpuset.h>
 #include <linux/module.h>
 #include <linux/notifier.h>
+#include <linux/rss_container.h>

 int sysctl_panic_on_oom;
 /* #define DEBUG */
@@ -47,7 +48,8 @@ int sysctl_panic_on_oom;
  *    of least surprise ... (be careful when you change it)
  */

-unsigned long badness(struct task_struct *p, unsigned long uptime)
+unsigned long badness(struct task_struct *p, unsigned long uptime,
+  struct rss_container *rss)
 {
  unsigned long points, cpu_time, run_time, s;
  struct mm_struct *mm;
@@ -60,6 +62,13 @@ unsigned long badness(struct task_struct
   return 0;
  }

+#ifdef CONFIG_RSS_CONTAINER
+ if (rss != NULL && mm->rss_container != rss) {
+  task_unlock(p);
+  return 0;
+ }
+#endif
+
  /*
   * The memory size of the process is the basis for the badness.
   */
@@ -200,7 +209,8 @@ static inline int constrained_alloc(stru
  *
  * (not docbooked, we don't want this one cluttering up the manual)
  */
-static struct task_struct *select_bad_process(unsigned long *ppoints)
+static struct task_struct *select_bad_process(unsigned long *ppoints,
+  struct rss_container *rss)
```

```
 {
  struct task_struct *g, *p;
  struct task_struct *chosen = NULL;
@@ -254,7 +264,7 @@ static struct task_struct *select_bad_pr
    if (p->oomkilladj == OOM_DISABLE)
     continue;

-  points = badness(p, uptime.tv_sec);
+   points = badness(p, uptime.tv_sec, rss);
    if (points > *ppoints || !chosen) {
     chosen = p;
     *ppoints = points;
@@ -435,7 +445,7 @@ retry:
     * Rambo mode: Shoot down a process and hope it solves whatever
     * issues we may have.
     */
-  p = select_bad_process(&points);
+   p = select_bad_process(&points, NULL);

    if (PTR_ERR(p) == -1UL)
     goto out;
@@ -464,3 +474,27 @@ out:
   if (!test_thread_flag(TIF_MEMDIE))
    schedule_timeout_uninterruptible(1);
 }
+
+#ifdef CONFIG_RSS_CONTAINER
+void container_out_of_memory(struct rss_container *rss)
+{
+ unsigned long points = 0;
+ struct task_struct *p;
+
+ container_lock();
+ read_lock(&tasklist_lock);
+retry:
+ p = select_bad_process(&points, rss);
+ if (PTR_ERR(p) == -1UL)
+   goto out;
+
+ if (!p)
+   p = current;
+
+ if (oom_kill_process(p, points, "Container out of memory"))
+   goto retry;
+out:
+ read_unlock(&tasklist_lock);
+ container_unlock();
+}
```

```
+#endif
diff -upr linux-2.6.20.orig/mm/vmscan.c linux-2.6.20-0/mm/vmscan.c
--- linux-2.6.20.orig/mm/vmscan.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/mm/vmscan.c 2007-03-06 13:33:28.000000000 +0300
@@ -45,6 +45,8 @@

 #include "internal.h"

+#include <linux/rss_container.h>
+
 struct scan_control {
  /* Incremented by the number of inactive pages that were scanned */
  unsigned long nr_scanned;
@@ -1097,6 +1099,194 @@ out:
  return ret;
 }

+#ifdef CONFIG_RSS_CONTAINER
+/*
+ * These are containers' inactive and active pages shrinkers.
+ * Thes works like shrink_inactive_list() and shrink_active_list()
+ *
+ * Two main differences is that container_isolate_pages() is used to isolate
+ * pages, and that reclaim_mapped is considered to be 1 as hitting BC
+ * limit implies we have to shrink _mapped_ pages
+ */
+static unsigned long container_shrink_pages_inactive(unsigned long max_scan,
+  struct rss_container *rss, struct scan_control *sc)
+{
+ LIST_HEAD(page_list);
+ unsigned long nr_scanned = 0;
+ unsigned long nr_reclaimed = 0;
+
+ do {
+  struct page *page;
+  unsigned long nr_taken;
+  unsigned long nr_scan;
+  struct zone *z;
+
+  nr_taken = container_isolate_pages(sc->swap_cluster_max, rss,
+    &page_list, 0, &nr_scan);
+
+  nr_scanned += nr_scan;
+  nr_reclaimed += shrink_page_list(&page_list, sc);
+  if (nr_taken == 0)
+   goto done;
+
+  while (!list_empty(&page_list)) {
```

```
+   page = lru_to_page(&page_list);
+   z = page_zone(page);
+
+   spin_lock_irq(&z->lru_lock);
+   VM_BUG_ON(PageLRU(page));
+   SetPageLRU(page);
+   list_del(&page->lru);
+   if (PageActive(page))
+    add_page_to_active_list(z, page);
+   else
+    add_page_to_inactive_list(z, page);
+   spin_unlock_irq(&z->lru_lock);
+
+   put_page(page);
+  }
+   } while (nr_scanned < max_scan);
+done:
+ return nr_reclaimed;
+}
+
+static void container_shrink_pages_active(unsigned long nr_pages,
+  struct rss_container *rss, struct scan_control *sc)
+{
+ LIST_HEAD(l_hold);
+ LIST_HEAD(l_inactive);
+ LIST_HEAD(l_active);
+ struct page *page;
+ unsigned long nr_scanned;
+ unsigned long nr_deactivated = 0;
+ struct zone *z;
+
+ container_isolate_pages(nr_pages, rss, &l_hold, 1, &nr_scanned);
+
+ while (!list_empty(&l_hold)) {
+  cond_resched();
+  page = lru_to_page(&l_hold);
+  list_del(&page->lru);
+  if (page_mapped(page)) {
+   if ((total_swap_pages == 0 && PageAnon(page)) ||
+       page_referenced(page, 0)) {
+    list_add(&page->lru, &l_active);
+    continue;
+   }
+  }
+  nr_deactivated++;
+  list_add(&page->lru, &l_inactive);
+ }
+
```

```
+ while (!list_empty(&l_inactive)) {
+  page = lru_to_page(&l_inactive);
+  z = page_zone(page);
+
+  spin_lock_irq(&z->lru_lock);
+  VM_BUG_ON(PageLRU(page));
+  SetPageLRU(page);
+  VM_BUG_ON(!PageActive(page));
+  ClearPageActive(page);
+
+  list_move(&page->lru, &z->inactive_list);
+  z->nr_inactive++;
+  spin_unlock_irq(&z->lru_lock);
+
+  put_page(page);
+ }
+
+ while (!list_empty(&l_active)) {
+  page = lru_to_page(&l_active);
+  z = page_zone(page);
+
+  spin_lock_irq(&z->lru_lock);
+  VM_BUG_ON(PageLRU(page));
+  SetPageLRU(page);
+  VM_BUG_ON(!PageActive(page));
+  list_move(&page->lru, &z->active_list);
+  z->nr_active++;
+  spin_unlock_irq(&z->lru_lock);
+
+  put_page(page);
+ }
+}
+
+/*
+ * This is a reworked shrink_zone() routine - it scans active pages firts,
+ * then inactive and returns the number of pages reclaimed
+ */
+static unsigned long container_shrink_pages(int priority,
+  struct rss_container *rss, struct scan_control *sc)
+{
+ unsigned long nr_pages;
+ unsigned long nr_to_scan;
+ unsigned long nr_reclaimed = 0;
+
+ nr_pages = (container_nr_physpages(rss) >> priority) + 1;
+ if (nr_pages < sc->swap_cluster_max)
+  nr_pages = 0;
+
```

```
+ while (nr_pages) {
+  nr_to_scan = min(nr_pages, (unsigned long)sc->swap_cluster_max);
+  nr_pages -= nr_to_scan;
+  container_shrink_pages_active(nr_to_scan, rss, sc);
+ }
+
+ nr_pages = (container_nr_physpages(rss) >> priority) + 1;
+ if (nr_pages < sc->swap_cluster_max)
+  nr_pages = 0;
+
+ while (nr_pages) {
+  nr_to_scan = min(nr_pages, (unsigned long)sc->swap_cluster_max);
+  nr_pages -= nr_to_scan;
+  nr_reclaimed += container_shrink_pages_inactive(nr_to_scan, rss, sc);
+ }
+
+ throttle_vm_writeout();
+ return nr_reclaimed;
+}
+
+/*
+ * This functions works like try_to_free_pages() - it tries
+ * to shrink bc's pages with increasing priority
+ */
+unsigned long container_try_to_free_pages(struct rss_container *rss)
+{
+ int priority;
+ int ret = 0;
+ unsigned long total_scanned = 0;
+ unsigned long nr_reclaimed = 0;
+ struct scan_control sc = {
+  .gfp_mask = GFP_KERNEL,
+  .may_writepage = !laptop_mode,
+  .swap_cluster_max = SWAP_CLUSTER_MAX,
+  .may_swap = 1,
+  .swappiness = vm_swappiness,
+ };
+
+ for (priority = DEF_PRIORITY; priority >= 0; priority--) {
+  sc.nr_scanned = 0;
+  nr_reclaimed += container_shrink_pages(priority, rss, &sc);
+  total_scanned += sc.nr_scanned;
+  if (nr_reclaimed > 1) {
+   ret = 1;
+   goto out;
+  }
+
+  if (total_scanned > sc.swap_cluster_max +
```

```
+     sc.swap_cluster_max / 2) {
+   wakeup_pdflush(laptop_mode ? 0 : total_scanned);
+   sc.may_writepage = 1;
+  }
+
+  if (sc.nr_scanned && priority < DEF_PRIORITY - 2)
+   congestion_wait(WRITE, HZ/10);
+ }
+out:
+ return ret;
+}
+#endif
+
 /*
  * For kswapd, balance_pgdat() will work across all this node's zones until
  * they are all at pages_high.
```

Subject: [RFC][PATCH 6/7] Account for the number of tasks within container
Posted by xemul on Tue, 06 Mar 2007 15:02:39 GMT
View Forum Message <> Reply to Message

Small and simple - each fork()/clone() is accounted
and rejected when limit is hit.

```
diff -upr linux-2.6.20.orig/include/linux/numproc_container.h
linux-2.6.20-0/include/linux/numproc_container.h
--- linux-2.6.20.orig/include/linux/numproc_container.h 2007-03-06 13:39:17.000000000 +0300
+++ linux-2.6.20-0/include/linux/numproc_container.h 2007-03-06 13:33:28.000000000 +0300
@@ -0,0 +1,32 @@
+#ifndef __NUMPROC_CONTAINER_H__
+#define __NUMPROC_CONTAINER_H__
+/*
+ * Numproc container
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+#ifdef CONFIG_PROCESS_CONTAINER
+int container_proc_charge(struct task_struct *tsk);
+void container_proc_uncharge(struct task_struct *tsk);
+
+void container_numproc_init_early(void);
+#else
+static inline int container_proc_charge(struct task_struct *tsk)
```

```
+{
+ return 0;
+}
+
+static inline void container_proc_uncharge(struct task_struct *tsk)
+{
+}
+
+static inline void container_numproc_init_early(void)
+{
+}
+#endif
+
+#endif
diff -upr linux-2.6.20.orig/include/linux/sched.h linux-2.6.20-0/include/linux/sched.h
--- linux-2.6.20.orig/include/linux/sched.h 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/include/linux/sched.h 2007-03-06 13:33:28.000000000 +0300
@@ -1052,6 +1055,9 @@ struct task_struct {
 #ifdef CONFIG_FAULT_INJECTION
  int make_it_fail;
 #endif
+#ifdef CONFIG_PROCESS_CONTAINER
+ struct numproc_container *numproc_cnt;
+#endif
 };

 static inline pid_t process_group(struct task_struct *tsk)
diff -upr linux-2.6.20.orig/init/Kconfig linux-2.6.20-0/init/Kconfig
--- linux-2.6.20.orig/init/Kconfig 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/init/Kconfig 2007-03-06 13:33:28.000000000 +0300
@@ -265,6 +265,12 @@ config CPUSETS
    Provides a simple Resource Controller for monitoring and
    controlling the total Resident Set Size of the tasks in a container

+config PROCESS_CONTAINER
+ bool "Numproc accounting container"
+ select RESOURCE_COUNTERS
+ help
+   Provides the-number-of-tasks accounting container
+
 config SYSFS_DEPRECATED
  bool "Create deprecated sysfs files"
  default y
diff -upr linux-2.6.20.orig/kernel/Makefile linux-2.6.20-0/kernel/Makefile
--- linux-2.6.20.orig/kernel/Makefile 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/kernel/Makefile 2007-03-06 13:33:28.000000000 +0300
@@ -51,6 +51,7 @@ obj-$(CONFIG_RELAY) += relay.o
 obj-$(CONFIG_TASK_DELAY_ACCT) += delayacct.o
```

```
 obj-$(CONFIG_TASKSTATS) += taskstats.o tsacct.o
 obj-$(CONFIG_RESOURCE_COUNTERS) += res_counter.o
+obj-$(CONFIG_PROCESS_CONTAINER) += numproc_container.o

 ifneq ($(CONFIG_SCHED_NO_NO_OMIT_FRAME_POINTER),y)
 # According to Alan Modra <alan@linuxcare.com.au>, the -fno-omit-frame-pointer is
diff -upr linux-2.6.20.orig/kernel/exit.c linux-2.6.20-0/kernel/exit.c
--- linux-2.6.20.orig/kernel/exit.c 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/kernel/exit.c 2007-03-06 13:33:28.000000000 +0300
@@ -48,6 +48,8 @@
 #include <asm/pgtable.h>
 #include <asm/mmu_context.h>

+#include <linux/numproc_container.h>
+
 extern void sem_exit (void);

 static void exit_mm(struct task_struct * tsk);
@@ -174,6 +176,7 @@ repeat:
  write_unlock_irq(&tasklist_lock);
  proc_flush_task(p);
  release_thread(p);
+ container_proc_uncharge(p);
  call_rcu(&p->rcu, delayed_put_task_struct);

  p = leader;
diff -upr linux-2.6.20.orig/kernel/fork.c linux-2.6.20-0/kernel/fork.c
--- linux-2.6.20.orig/kernel/fork.c 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/kernel/fork.c 2007-03-06 13:33:28.000000000 +0300
@@ -57,6 +57,7 @@
 #include <asm/tlbflush.h>

 #include <linux/rss_container.h>
+#include <linux/numproc_container.h>

 /*
  * Protected counters by write_lock_irq(&tasklist_lock)
@@ -986,6 +999,9 @@ static struct task_struct *copy_process(
  if (!p)
   goto fork_out;

+ if (container_proc_charge(p))
+  goto charge_out;
+
  rt_mutex_init_task(p);

 #ifdef CONFIG_TRACE_IRQFLAGS
@@ -1302,6 +1318,8 @@ bad_fork_cleanup_count:
```

```
  atomic_dec(&p->user->processes);
  free_uid(p->user);
 bad_fork_free:
+ container_proc_uncharge(p);
+charge_out:
  free_task(p);
 fork_out:
  return ERR_PTR(retval);
diff -upr linux-2.6.20.orig/kernel/numproc_container.c linux-2.6.20-0/kernel/numproc_container.c
--- linux-2.6.20.orig/kernel/numproc_container.c 2007-03-06 13:39:17.000000000 +0300
+++ linux-2.6.20-0/kernel/numproc_container.c 2007-03-06 13:33:28.000000000 +0300
@@ -0,0 +1,151 @@
+/*
+ * Numproc accounting container
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+#include <linux/list.h>
+#include <linux/sched.h>
+#include <linux/mm.h>
+#include <linux/res_counter.h>
+#include <linux/numproc_container.h>
+
+static struct container_subsys numproc_subsys;
+
+struct numproc_container {
+ struct res_counter res;
+ struct container_subsys_state css;
+};
+
+static inline struct numproc_container *numproc_from_cont(struct container *cnt)
+{
+ return container_of(container_subsys_state(cnt, &numproc_subsys),
+    struct numproc_container, css);
+}
+
+int container_proc_charge(struct task_struct *new)
+{
+ struct numproc_container *np;
+
+ rcu_read_lock();
+ np = numproc_from_cont(task_container(current, &numproc_subsys));
+ css_get_current(&np->css);
+ rcu_read_unlock();
```

```
+
+ if (res_counter_charge(&np->res, 1)) {
+  css_put(&np->css);
+  return -ENOMEM;
+ }
+
+ new->numproc_cnt = np;
+ return 0;
+}
+
+void container_proc_uncharge(struct task_struct *tsk)
+{
+ struct numproc_container *np;
+
+ np = tsk->numproc_cnt;
+ res_counter_uncharge(&np->res, 1);
+ css_put(&np->css);
+}
+
+static int numproc_create(struct container_subsys *ss, struct container *cont)
+{
+ struct numproc_container *np;
+
+ np = kzalloc(sizeof(struct numproc_container), GFP_KERNEL);
+ if (np == NULL)
+  return -ENOMEM;
+
+ res_counter_init(&np->res);
+ cont->subsys[numproc_subsys.subsys_id] = &np->css;
+ return 0;
+}
+
+static void numproc_destroy(struct container_subsys *ss,
+  struct container *cont)
+{
+ kfree(numproc_from_cont(cont));
+}
+
+
+static ssize_t numproc_read(struct container *cont, struct cftype *cft,
+  struct file *file, char __user *userbuf,
+  size_t nbytes, loff_t *ppos)
+{
+ return res_counter_read(&numproc_from_cont(cont)->res, cft->private,
+   userbuf, nbytes, ppos);
+}
+
+static ssize_t numproc_write(struct container *cont, struct cftype *cft,
```

```
+  struct file *file, const char __user *userbuf,
+  size_t nbytes, loff_t *ppos)
+{
+ return res_counter_write(&numproc_from_cont(cont)->res, cft->private,
+   userbuf, nbytes, ppos);
+}
+
+
+static struct cftype numproc_usage = {
+ .name = "numproc_usage",
+ .private = RES_USAGE,
+ .read = numproc_read,
+};
+
+static struct cftype numproc_limit = {
+ .name = "numproc_limit",
+ .private = RES_LIMIT,
+ .read = numproc_read,
+ .write = numproc_write,
+};
+
+static struct cftype numproc_failcnt = {
+ .name = "numproc_failcnt",
+ .private = RES_FAILCNT,
+ .read = numproc_read,
+};
+
+static int numproc_populate(struct container_subsys *ss,
+  struct container *cont)
+{
+ int rc;
+
+ if ((rc = container_add_file(cont, &numproc_usage)) < 0)
+  return rc;
+ if ((rc = container_add_file(cont, &numproc_failcnt)) < 0)
+  return rc;
+ if ((rc = container_add_file(cont, &numproc_limit)) < 0)
+  return rc;
+
+ return 0;
+}
+
+static struct numproc_container init_numproc_container;
+
+static __init int numproc_create_early(struct container_subsys *ss,
+  struct container *cont)
+{
+ struct numproc_container *np;
```

```
+
+ np = &init_numproc_container;
+ res_counter_init(&np->res);
+ cont->subsys[numproc_subsys.subsys_id] = &np->css;
+ ss->create = numproc_create;
+ return 0;
+}
+
+static struct container_subsys numproc_subsys = {
+ .name = "numproc",
+ .create = numproc_create_early,
+ .destroy = numproc_destroy,
+ .populate = numproc_populate,
+};
+
+void __init container_numproc_init_early(void)
+{
+ container_register_subsys(&numproc_subsys);
+}
diff -upr linux-2.6.20.orig/kernel/container.c linux-2.6.20-0/kernel/container.c
--- linux-2.6.20.orig/kernel/container.c 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/kernel/container.c 2007-03-06 13:35:48.000000000 +0300
@@ -60,6 +60,7 @@
 #include <linux/mutex.h>

 #include <linux/rss_container.h>
+#include <linux/numproc_container.h>

 #define CONTAINER_SUPER_MAGIC  0x27e0eb

@@ -1721,6 +1725,7 @@ int __init container_init_early(void)
  init_task.containers = &init_container_group;

  container_rss_init_early();
+ container_numproc_init_early();

  return 0;
 }
```

Subject: [RFC][PATCH 7/7] Account for the number of files opened within container
Posted by xemul on Tue, 06 Mar 2007 15:05:46 GMT
View Forum Message <> Reply to Message

Simple again - increment usage counter at file open and
decrement at file close. Reject opening if limit is hit.

diff -upr linux-2.6.20.orig/fs/Makefile linux-2.6.20-0/fs/Makefile

```
--- linux-2.6.20.orig/fs/Makefile 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/fs/Makefile 2007-03-06 13:33:28.000000000 +0300
@@ -19,6 +19,8 @@ else
 obj-y += no-block.o
 endif

+obj-$(CONFIG_FILES_CONTAINER) += numfiles_container.o
+
 obj-$(CONFIG_INOTIFY)  += inotify.o
 obj-$(CONFIG_INOTIFY_USER) += inotify_user.o
 obj-$(CONFIG_EPOLL)  += eventpoll.o
diff -upr linux-2.6.20.orig/fs/file_table.c linux-2.6.20-0/fs/file_table.c
--- linux-2.6.20.orig/fs/file_table.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/fs/file_table.c 2007-03-06 13:33:28.000000000 +0300
@@ -21,6 +21,7 @@
 #include <linux/fsnotify.h>
 #include <linux/sysctl.h>
 #include <linux/percpu_counter.h>
+#include <linux/numfiles_container.h>

 #include <asm/atomic.h>

@@ -42,6 +43,7 @@ static inline void file_free_rcu(struct

 static inline void file_free(struct file *f)
 {
+ container_file_uncharge(f);
  percpu_counter_dec(&nr_files);
  call_rcu(&f->f_u.fu_rcuhead, file_free_rcu);
 }
@@ -109,6 +111,10 @@ struct file *get_empty_filp(void)

  percpu_counter_inc(&nr_files);
  memset(f, 0, sizeof(*f));
+
+ if (container_file_charge(f))
+  goto fail_charge;
+
  if (security_file_alloc(f))
   goto fail_sec;

@@ -132,7 +138,10 @@ over:
  goto fail;

 fail_sec:
- file_free(f);
+ container_file_uncharge(f);
+fail_charge:
```

```
+ percpu_counter_dec(&nr_files);
+ kmem_cache_free(filp_cachep, f);
 fail:
  return NULL;
 }
diff -upr linux-2.6.20.orig/fs/numfiles_container.c linux-2.6.20-0/fs/numfiles_container.c
--- linux-2.6.20.orig/fs/numfiles_container.c 2007-03-06 13:39:17.000000000 +0300
+++ linux-2.6.20-0/fs/numfiles_container.c 2007-03-06 13:33:28.000000000 +0300
@@ -0,0 +1,152 @@
+/*
+ * Numfiles accounting container
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+#include <linux/list.h>
+#include <linux/sched.h>
+#include <linux/mm.h>
+#include <linux/res_counter.h>
+#include <linux/numfiles_container.h>
+
+static struct container_subsys numfiles_subsys;
+
+struct files_container {
+ struct res_counter res;
+ struct container_subsys_state css;
+};
+
+static inline struct files_container *numfiles_from_cont(struct container *cnt)
+{
+ return container_of(container_subsys_state(cnt, &numfiles_subsys),
+    struct files_container, css);
+}
+
+int container_file_charge(struct file *file)
+{
+ struct files_container *fc;
+
+ rcu_read_lock();
+ fc = numfiles_from_cont(task_container(current, &numfiles_subsys));
+ css_get_current(&fc->css);
+ rcu_read_unlock();
+
+ if (res_counter_charge(&fc->res, 1)) {
+  css_put(&fc->css);
```

```
+  return -ENOMEM;
+ }
+
+ file->f_cont = fc;
+ return 0;
+}
+
+void container_file_uncharge(struct file *file)
+{
+ struct files_container *fc;
+
+ fc = file->f_cont;
+ res_counter_uncharge(&fc->res, 1);
+ css_put(&fc->css);
+}
+
+static int numfiles_create(struct container_subsys *ss, struct container *cont)
+{
+ struct files_container *fc;
+
+ fc = kzalloc(sizeof(struct files_container), GFP_KERNEL);
+ if (fc == NULL)
+  return -ENOMEM;
+
+ res_counter_init(&fc->res);
+ cont->subsys[numfiles_subsys.subsys_id] = &fc->css;
+ return 0;
+}
+
+static void numfiles_destroy(struct container_subsys *ss,
+  struct container *cont)
+{
+ kfree(numfiles_from_cont(cont));
+}
+
+
+static ssize_t numfiles_read(struct container *cont, struct cftype *cft,
+  struct file *file, char __user *userbuf,
+  size_t nbytes, loff_t *ppos)
+{
+ return res_counter_read(&numfiles_from_cont(cont)->res, cft->private,
+   userbuf, nbytes, ppos);
+}
+
+static ssize_t numfiles_write(struct container *cont, struct cftype *cft,
+  struct file *file, const char __user *userbuf,
+  size_t nbytes, loff_t *ppos)
+{
```

```
+ return res_counter_write(&numfiles_from_cont(cont)->res, cft->private,
+   userbuf, nbytes, ppos);
+}
+
+
+static struct cftype numfiles_usage = {
+ .name = "numfiles_usage",
+ .private = RES_USAGE,
+ .read = numfiles_read,
+};
+
+static struct cftype numfiles_limit = {
+ .name = "numfiles_limit",
+ .private = RES_LIMIT,
+ .read = numfiles_read,
+ .write = numfiles_write,
+};
+
+static struct cftype numfiles_failcnt = {
+ .name = "numfiles_failcnt",
+ .private = RES_FAILCNT,
+ .read = numfiles_read,
+};
+
+static int numfiles_populate(struct container_subsys *ss,
+  struct container *cont)
+{
+ int rc;
+
+ if ((rc = container_add_file(cont, &numfiles_usage)) < 0)
+  return rc;
+ if ((rc = container_add_file(cont, &numfiles_failcnt)) < 0)
+  return rc;
+ if ((rc = container_add_file(cont, &numfiles_limit)) < 0)
+  return rc;
+
+ return 0;
+}
+
+static struct files_container init_files_container;
+
+static __init int numfiles_create_early(struct container_subsys *ss,
+  struct container *cont)
+{
+ struct files_container *np;
+
+ np = &init_files_container;
+ res_counter_init(&np->res);
```

```
+ cont->subsys[numfiles_subsys.subsys_id] = &np->css;
+ ss->create = numfiles_create;
+ return 0;
+}
+
+static struct container_subsys numfiles_subsys = {
+ .name = "numfiles",
+ .create = numfiles_create_early,
+ .destroy = numfiles_destroy,
+ .populate = numfiles_populate,
+};
+
+void __init container_numfiles_init_early(void)
+{
+ container_register_subsys(&numfiles_subsys);
+}
+
diff -upr linux-2.6.20.orig/include/linux/fs.h linux-2.6.20-0/include/linux/fs.h
--- linux-2.6.20.orig/include/linux/fs.h 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/include/linux/fs.h 2007-03-06 13:33:28.000000000 +0300
@@ -739,6 +739,9 @@ struct file {
  spinlock_t  f_ep_lock;
 #endif /* #ifdef CONFIG_EPOLL */
  struct address_space *f_mapping;
+#ifdef CONFIG_FILES_CONTAINER
+ struct files_container *f_cont;
+#endif
 };
 extern spinlock_t files_lock;
 #define file_list_lock() spin_lock(&files_lock);
diff -upr linux-2.6.20.orig/include/linux/numfiles_container.h
linux-2.6.20-0/include/linux/numfiles_container.h
--- linux-2.6.20.orig/include/linux/numfiles_container.h 2007-03-06 13:39:17.000000000 +0300
+++ linux-2.6.20-0/include/linux/numfiles_container.h 2007-03-06 13:33:28.000000000 +0300
@@ -0,0 +1,33 @@
+#ifndef __NUMFILES_CONTAINER_H__
+#define __NUMFILES_CONTAINER_H__
+/*
+ * Numfiles container
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+#ifdef CONFIG_FILES_CONTAINER
+int container_file_charge(struct file *file);
```

```
+void container_file_uncharge(struct file *file);
+
+void container_numfiles_init_early(void);
+#else
+static inline int container_file_charge(struct file *file)
+{
+ return 0;
+}
+
+static inline void container_file_uncharge(struct file *file)
+{
+}
+
+static inline void container_numfiles_init_early(void)
+{
+}
+#endif
+
+#endif
+
diff -upr linux-2.6.20.orig/init/Kconfig linux-2.6.20-0/init/Kconfig
--- linux-2.6.20.orig/init/Kconfig 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/init/Kconfig 2007-03-06 13:33:28.000000000 +0300
@@ -265,6 +265,12 @@ config CPUSETS
  help
    Provides the-number-of-tasks accounting container

+config FILES_CONTAINER
+ bool "Numfiles accounting container"
+ select RESOURCE_COUNTERS
+ help
+   Provides the-number-of-files accounting container
+
 config SYSFS_DEPRECATED
  bool "Create deprecated sysfs files"
  default y
diff -upr linux-2.6.20.orig/kernel/container.c linux-2.6.20-0/kernel/container.c
--- linux-2.6.20.orig/kernel/container.c 2007-03-06 13:33:28.000000000 +0300
+++ linux-2.6.20-0/kernel/container.c 2007-03-06 13:35:48.000000000 +0300
@@ -60,6 +60,7 @@

 #include <linux/rss_container.h>
 #include <linux/numproc_container.h>
+#include <linux/numfiles_container.h>

 #define CONTAINER_SUPER_MAGIC  0x27e0eb

@@ -1721,6 +1725,7 @@ int __init container_init_early(void)
```

```
  container_rss_init_early();
  container_numproc_init_early();
+ container_numfiles_init_early();

  return 0;
 }
```

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Andrew Morton on Tue, 06 Mar 2007 22:00:36 GMT
View Forum Message <> Reply to Message

On Tue, 06 Mar 2007 17:55:29 +0300
Pavel Emelianov <xemul@sw.ru> wrote:

> +struct rss_container {
> + struct res_counter res;
> + struct list_head page_list;
> + struct container_subsys_state css;
> +};
> +
> +struct page_container {
> + struct page *page;
> + struct rss_container *cnt;
> + struct list_head list;
> +};

ah. This looks good. I'll find a hunk of time to go through this work
and through Paul's patches. It'd be good to get both patchsets lined
up in -mm within a couple of weeks. But..

We need to decide whether we want to do per-container memory limitation via
these data structures, or whether we do it via a physical scan of some
software zone, possibly based on Mel's patches.

Subject: Re: [RFC][PATCH 6/7] Account for the number of tasks within container
Posted by Paul Menage on Wed, 07 Mar 2007 02:00:04 GMT
View Forum Message <> Reply to Message

Hi Pavel,

On 3/6/07, Pavel Emelianov <xemul@sw.ru> wrote:
> diff -upr linux-2.6.20.orig/include/linux/sched.h linux-2.6.20-0/include/linux/sched.h
> --- linux-2.6.20.orig/include/linux/sched.h 2007-03-06 13:33:28.000000000 +0300
> +++ linux-2.6.20-0/include/linux/sched.h 2007-03-06 13:33:28.000000000 +0300

> @@ -1052,6 +1055,9 @@ struct task_struct {
> #ifdef CONFIG_FAULT_INJECTION
> int make_it_fail;
> #endif
> +#ifdef CONFIG_PROCESS_CONTAINER
> + struct numproc_container *numproc_cnt;
> +#endif
> };

Why do you need a pointer added to task_struct? One of the main points
of the generic containers is to avoid every different subsystem and
resource controller having to add new pointers there.

> +
> + rcu_read_lock();
> + np = numproc_from_cont(task_container(current, &numproc_subsys));
> + css_get_current(&np->css);

There's no need to hold a reference here - by definition, the task's
container can't go away while the task is in it.

Also, shouldn't you have an attach() method to move the count from one
container to another when a task moves?

Paul

Subject: Re: [RFC][PATCH 0/7] Resource controllers based on process containers
Posted by Paul Menage on Wed, 07 Mar 2007 02:02:32 GMT
View Forum Message <> Reply to Message

On 3/6/07, Pavel Emelianov <xemul@sw.ru> wrote:
> 2. Extended containers may register themselves too late.
> Kernel threads/helpers start forking, opening files
> and touching pages much earlier. This patchset
> workarounds this in not-so-cute manner and I'm waiting
> for Paul's comments on this issue.
>

Can we not make sure that each subsystem registers itself before any
of its resources become usable? So the file counting subsystem should
register at some point before filp_open() becomes usable, and the
process counting subsystem should register before it's possible to
fork, etc.

Paul

Subject: Re: [RFC][PATCH 1/7] Resource counters
Posted by Balbir Singh on Wed, 07 Mar 2007 04:03:49 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Introduce generic structures and routines for
> resource accounting.
>
> Each resource accounting container is supposed to
> aggregate it, container_subsystem_state and its
> resource-specific members within.
>
>
> ---------- ----------
>
> diff -upr linux-2.6.20.orig/include/linux/res_counter.h linux-2.6.20-0/include/linux/res_counter.h
> --- linux-2.6.20.orig/include/linux/res_counter.h 2007-03-06 13:39:17.000000000 +0300
> +++ linux-2.6.20-0/include/linux/res_counter.h 2007-03-06 13:33:28.000000000 +0300
> @@ -0,0 +1,83 @@
> +#ifndef __RES_COUNTER_H__
> +#define __RES_COUNTER_H__
> +/*
> + * resource counters
> + *
> + * Copyright 2007 OpenVZ SWsoft Inc
> + *
> + * Author: Pavel Emelianov <xemul@openvz.org>
> + *
> + */
> +
> +#include <linux/container.h>
> +
> +struct res_counter {
> + unsigned long usage;
> + unsigned long limit;
> + unsigned long failcnt;
> + spinlock_t lock;
> +};
> +
> +enum {
> + RES_USAGE,
> + RES_LIMIT,
> + RES_FAILCNT,
> +};
> +
> +ssize_t res_counter_read(struct res_counter *cnt, int member,
> + const char __user *buf, size_t nbytes, loff_t *pos);
> +ssize_t res_counter_write(struct res_counter *cnt, int member,
> + const char __user *buf, size_t nbytes, loff_t *pos);

> +
> +static inline void res_counter_init(struct res_counter *cnt)
> +{
> + spin_lock_init(&cnt->lock);
> + cnt->limit = (unsigned long)LONG_MAX;
> +}
> +

Is there any way to indicate that there are no limits on this container.
LONG_MAX is quite huge, but still when the administrator wants to
configure a container to *un-limited usage*, it becomes hard for
the administrator.

> +static inline int res_counter_charge_locked(struct res_counter *cnt,
> + unsigned long val)
> +{
> + if (cnt->usage <= cnt->limit - val) {
> + cnt->usage += val;
> + return 0;
> + }
> +
> + cnt->failcnt++;
> + return -ENOMEM;
> +}
> +
> +static inline int res_counter_charge(struct res_counter *cnt,
> + unsigned long val)
> +{
> + int ret;
> + unsigned long flags;
> +
> + spin_lock_irqsave(&cnt->lock, flags);
> + ret = res_counter_charge_locked(cnt, val);
> + spin_unlock_irqrestore(&cnt->lock, flags);
> + return ret;
> +}
> +

Will atomic counters help here.

> +static inline void res_counter_uncharge_locked(struct res_counter *cnt,
> + unsigned long val)
> +{
> + if (unlikely(cnt->usage < val)) {
> + WARN_ON(1);
> + val = cnt->usage;
> + }
> +

> + cnt->usage -= val;
> +}
> +
> +static inline void res_counter_uncharge(struct res_counter *cnt,
> + unsigned long val)
> +{
> + unsigned long flags;
> +
> + spin_lock_irqsave(&cnt->lock, flags);
> + res_counter_uncharge_locked(cnt, val);
> + spin_unlock_irqrestore(&cnt->lock, flags);
> +}
> +
> +#endif
> diff -upr linux-2.6.20.orig/init/Kconfig linux-2.6.20-0/init/Kconfig
> --- linux-2.6.20.orig/init/Kconfig 2007-03-06 13:33:28.000000000 +0300
> +++ linux-2.6.20-0/init/Kconfig 2007-03-06 13:33:28.000000000 +0300
> @@ -265,6 +265,10 @@ config CPUSETS
>
>   Say N if unsure.
>
> +config RESOURCE_COUNTERS
> + bool
> + select CONTAINERS
> +
> config SYSFS_DEPRECATED
>   bool "Create deprecated sysfs files"
>   default y
> diff -upr linux-2.6.20.orig/kernel/Makefile linux-2.6.20-0/kernel/Makefile
> --- linux-2.6.20.orig/kernel/Makefile 2007-03-06 13:33:28.000000000 +0300
> +++ linux-2.6.20-0/kernel/Makefile 2007-03-06 13:33:28.000000000 +0300
> @@ -51,6 +51,7 @@ obj-$(CONFIG_RELAY) += relay.o
> obj-$(CONFIG_UTS_NS) += utsname.o
> obj-$(CONFIG_TASK_DELAY_ACCT) += delayacct.o
> obj-$(CONFIG_TASKSTATS) += taskstats.o tsacct.o
> +obj-$(CONFIG_RESOURCE_COUNTERS) += res_counter.o
>
> ifneq ($(CONFIG_SCHED_NO_NO_OMIT_FRAME_POINTER),y)
> # According to Alan Modra <alan@linuxcare.com.au>, the -fno-omit-frame-pointer is
> diff -upr linux-2.6.20.orig/kernel/res_counter.c linux-2.6.20-0/kernel/res_counter.c
> --- linux-2.6.20.orig/kernel/res_counter.c 2007-03-06 13:39:17.000000000 +0300
> +++ linux-2.6.20-0/kernel/res_counter.c 2007-03-06 13:33:28.000000000 +0300
> @@ -0,0 +1,72 @@
> +/*
> + * resource containers
> + *
> + * Copyright 2007 OpenVZ SWsoft Inc
> + *

```
> + * Author: Pavel Emelianov <xemul@openvz.org>
> + *
> + */
> +
> +#include <linux/parser.h>
> +#include <linux/fs.h>
> +#include <linux/res_counter.h>
> +#include <asm/uaccess.h>
> +
> +static inline unsigned long *res_counter_member(struct res_counter *cnt, int member)
> +{
> + switch (member) {
> + case RES_USAGE:
> +  return &cnt->usage;
> + case RES_LIMIT:
> +  return &cnt->limit;
> + case RES_FAILCNT:
> +  return &cnt->failcnt;
> + };
> +
> + BUG();
> + return NULL;
> +}
> +
> +ssize_t res_counter_read(struct res_counter *cnt, int member,
> +  const char __user *userbuf, size_t nbytes, loff_t *pos)
> +{
> + unsigned long *val;
> + char buf[64], *s;
> +
> + s = buf;
> + val = res_counter_member(cnt, member);
> + s += sprintf(s, "%lu\n", *val);
> + return simple_read_from_buffer((void __user *)userbuf, nbytes,
> +   pos, buf, s - buf);
> +}
> +
> +ssize_t res_counter_write(struct res_counter *cnt, int member,
> +  const char __user *userbuf, size_t nbytes, loff_t *pos)
> +{
> + int ret;
> + char *buf, *end;
> + unsigned long tmp, *val;
> +
> + buf = kmalloc(nbytes + 1, GFP_KERNEL);
> + ret = -ENOMEM;
> + if (buf == NULL)
> +  goto out;
```

> +
> + buf[nbytes] = 0;
> + ret = -EFAULT;
> + if (copy_from_user(buf, userbuf, nbytes))
> + goto out_free;
> +
> + ret = -EINVAL;
> + tmp = simple_strtoul(buf, &end, 10);
> + if (*end != '\0')
> + goto out_free;
> +
> + val = res_counter_member(cnt, member);
> + *val = tmp;
> + ret = nbytes;
> +out_free:
> + kfree(buf);
> +out:
> + return ret;
> +}
>

These bits look a little out of sync, with no users for these routines in
this patch. Won't you get a compiler warning, compiling this bit alone?

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

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Balbir Singh on Wed, 07 Mar 2007 05:37:56 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> This includes setup of RSS container within generic
> process containers, all the declarations used in RSS
> accounting, and core code responsible for accounting.
>
>
> ---------- ----------
>
> diff -upr linux-2.6.20.orig/include/linux/rss_container.h
linux-2.6.20-0/include/linux/rss_container.h
> --- linux-2.6.20.orig/include/linux/rss_container.h 2007-03-06 13:39:17.000000000 +0300
> +++ linux-2.6.20-0/include/linux/rss_container.h 2007-03-06 13:33:28.000000000 +0300

```
> @@ -0,0 +1,68 @@
> +#ifndef __RSS_CONTAINER_H__
> +#define __RSS_CONTAINER_H__
> +/*
> + * RSS container
> + *
> + * Copyright 2007 OpenVZ SWsoft Inc
> + *
> + * Author: Pavel Emelianov <xemul@openvz.org>
> + *
> + */
> +
> +struct page_container;
> +struct rss_container;
> +
> +#ifdef CONFIG_RSS_CONTAINER
> +int container_rss_prepare(struct page *, struct vm_area_struct *vma,
> +  struct page_container **);
> +
> +void container_rss_add(struct page_container *);
> +void container_rss_del(struct page_container *);
> +void container_rss_release(struct page_container *);
> +
> +int mm_init_container(struct mm_struct *mm, struct task_struct *tsk);
> +void mm_free_container(struct mm_struct *mm);
> +
> +unsigned long container_isolate_pages(unsigned long nr_to_scan,
> +  struct rss_container *rss, struct list_head *dst,
> +  int active, unsigned long *scanned);
> +unsigned long container_nr_physpages(struct rss_container *rss);
> +
> +unsigned long container_try_to_free_pages(struct rss_container *);
> +void container_out_of_memory(struct rss_container *);
> +
> +void container_rss_init_early(void);
> +#else
> +static inline int container_rss_prepare(struct page *pg,
> +  struct vm_area_struct *vma, struct page_container **pc)
> +{
> + *pc = NULL; /* to make gcc happy */
> + return 0;
> +}
> +
> +static inline void container_rss_add(struct page_container *pc)
> +{
> +}
> +
> +static inline void container_rss_del(struct page_container *pc)
```

> +{
> +}
> +
> +static inline void container_rss_release(struct page_container *pc)
> +{
> +}
> +
> +static inline int mm_init_container(struct mm_struct *mm, struct task_struct *t)
> +{
> + return 0;
> +}
> +
> +static inline void mm_free_container(struct mm_struct *mm)
> +{
> +}
> +
> +static inline void container_rss_init_early(void)
> +{
> +}
> +#endif
> +#endif
> diff -upr linux-2.6.20.orig/init/Kconfig linux-2.6.20-0/init/Kconfig
> --- linux-2.6.20.orig/init/Kconfig 2007-03-06 13:33:28.000000000 +0300
> +++ linux-2.6.20-0/init/Kconfig 2007-03-06 13:33:28.000000000 +0300
> @@ -265,6 +265,13 @@ config CPUSETS
> bool
> select CONTAINERS
>
> +config RSS_CONTAINER
> + bool "RSS accounting container"
> + select RESOURCE_COUNTERS
> + help
> + Provides a simple Resource Controller for monitoring and
> + controlling the total Resident Set Size of the tasks in a container
> +

The wording looks very familiar :-). It would be useful to add
"The reclaim logic is now container aware, when the container goes overlimit
the page reclaimer reclaims pages belonging to this container. If we are
unable to reclaim enough pages to satisfy the request, the process is
killed with an out of memory warning"

> config SYSFS_DEPRECATED
> bool "Create deprecated sysfs files"
> default y
> diff -upr linux-2.6.20.orig/mm/Makefile linux-2.6.20-0/mm/Makefile
> --- linux-2.6.20.orig/mm/Makefile 2007-02-04 21:44:54.000000000 +0300
> +++ linux-2.6.20-0/mm/Makefile 2007-03-06 13:33:28.000000000 +0300

> @@ -29,3 +29,5 @@ obj-$(CONFIG_MEMORY_HOTPLUG) += memory_h
> obj-$(CONFIG_FS_XIP) += filemap_xip.o
> obj-$(CONFIG_MIGRATION) += migrate.o
> obj-$(CONFIG_SMP) += allocpercpu.o
> +
> +obj-$(CONFIG_RSS_CONTAINER) += rss_container.o
> diff -upr linux-2.6.20.orig/mm/rss_container.c linux-2.6.20-0/mm/rss_container.c
> --- linux-2.6.20.orig/mm/rss_container.c 2007-03-06 13:39:17.000000000 +0300
> +++ linux-2.6.20-0/mm/rss_container.c 2007-03-06 13:33:28.000000000 +0300
> @@ -0,0 +1,307 @@
> +/*
> + * RSS accounting container
> + *
> + * Copyright 2007 OpenVZ SWsoft Inc
> + *
> + * Author: Pavel Emelianov <xemul@openvz.org>
> + *
> + */
> +
> +#include <linux/list.h>
> +#include <linux/sched.h>
> +#include <linux/mm.h>
> +#include <linux/res_counter.h>
> +#include <linux/rss_container.h>
> +
> +static struct container_subsys rss_subsys;
> +
> +struct rss_container {
> + struct res_counter res;
> + struct list_head page_list;
> + struct container_subsys_state css;
> +};
> +
> +struct page_container {
> + struct page *page;
> + struct rss_container *cnt;
> + struct list_head list;
> +};
> +

Yes, this is what I was planning to get to -- a per container LRU list.
But you have just one list, don't you need active and inactive lists?
When the global LRU is manipulated, shouldn't this list be updated as
well, so that reclaim will pick the best pages.

> +static inline struct rss_container *rss_from_cont(struct container *cnt)
> +{
> + return container_of(container_subsys_state(cnt, &rss_subsys),

```
> +   struct rss_container, css);
> +}
> +
> +int mm_init_container(struct mm_struct *mm, struct task_struct *tsk)
> +{
> + struct rss_container *cnt;
> +
> + cnt = rss_from_cont(task_container(tsk, &rss_subsys));
> + if (css_get(&cnt->css))
> +  return -EBUSY;
> +
> + mm->rss_container = cnt;
> + return 0;
> +}
> +
> +void mm_free_container(struct mm_struct *mm)
> +{
> + css_put(&mm->rss_container->css);
> +}
> +
> +int container_rss_prepare(struct page *page, struct vm_area_struct *vma,
> +  struct page_container **ppc)
> +{
> + struct rss_container *rss;
> + struct page_container *pc;
> +
> + rcu_read_lock();
> + rss = rcu_dereference(vma->vm_mm->rss_container);
> + css_get_current(&rss->css);
> + rcu_read_unlock();
> +
> + pc = kmalloc(sizeof(struct page_container), GFP_KERNEL);
> + if (pc == NULL)
> +  goto out_nomem;
> +
> + while (res_counter_charge(&rss->res, 1)) {
> +  if (container_try_to_free_pages(rss))
> +   continue;
> +
```

The return codes of the functions is a bit confusing, ideally
container_try_to_free_pages() should return 0 on success. Also
res_counter_charge() has a WARN_ON(1) if the limit is exceeded.
The system administrator can figure out the details from failcnt,
I suspect when the container is running close to it's limit,
dmesg will have too many WARNING messages.

How much memory do you try to reclaim in container_try_to_free_pages()?

With my patches, I was planning to export this knob to userspace with
a default value. This will help the administrator decide how much
of the working set/container LRU should be freed on reaching the limit.
I cannot find the definition of container_try_to_free_pages() in
this patch.

> + container_out_of_memory(rss);
> + if (test_thread_flag(TIF_MEMDIE))
> +   goto out_charge;
> + }
> +
> + pc->page = page;
> + pc->cnt = rss;
> + *ppc = pc;
> + return 0;
> +
> +out_charge:
> + kfree(pc);
> +out_nomem:
> + css_put(&rss->css);
> + return -ENOMEM;
> +}
> +
> +void container_rss_release(struct page_container *pc)
> +{
> + struct rss_container *rss;
> +
> + rss = pc->cnt;
> + res_counter_uncharge(&rss->res, 1);
> + css_put(&rss->css);
> + kfree(pc);
> +}
> +
> +void container_rss_add(struct page_container *pc)
> +{
> + struct page *pg;
> + struct rss_container *rss;
> +
> + pg = pc->page;
> + rss = pc->cnt;
> +
> + spin_lock(&rss->res.lock);
> + list_add(&pc->list, &rss->page_list);

This is not good, it won't give us LRU behaviour which is
useful for determining which pages to free.

> + spin_unlock(&rss->res.lock);
> +
> + page_container(pg) = pc;
> +}
> +
> +void container_rss_del(struct page_container *pc)
> +{
> + struct page *page;
> + struct rss_container *rss;
> +
> + page = pc->page;
> + rss = pc->cnt;
> +
> + spin_lock(&rss->res.lock);
> + list_del(&pc->list);
> + res_counter_uncharge_locked(&rss->res, 1);
> + spin_unlock(&rss->res.lock);
> +
> + css_put(&rss->css);
> + kfree(pc);
> +}
> +
> +unsigned long container_isolate_pages(unsigned long nr_to_scan,
> + struct rss_container *rss, struct list_head *dst,
> + int active, unsigned long *scanned)
> +{
> + unsigned long nr_taken = 0;
> + struct page *page;
> + struct page_container *pc;
> + unsigned long scan;
> + struct list_head *src;
> + LIST_HEAD(pc_list);
> + struct zone *z;
> +
> + spin_lock_irq(&rss->res.lock);
> + src = &rss->page_list;
> +

Which part of the working set are we pushing out, this looks like
we are using FIFO to determine which pages to reclaim. This needs
to be FIXED.

> + for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
> + pc = list_entry(src->prev, struct page_container, list);
> + page = pc->page;
> + z = page_zone(page);
> +

```
> +  list_move(&pc->list, &pc_list);
> +
> +  spin_lock(&z->lru_lock);
> +  if (PageLRU(page)) {
> +   if ((active && PageActive(page)) ||
> +     (!active && !PageActive(page))) {
> +    if (likely(get_page_unless_zero(page))) {
> +     ClearPageLRU(page);
> +     nr_taken++;
> +     list_move(&page->lru, dst);
> +    }
> +   }
> +  }
> +  spin_unlock(&z->lru_lock);
> + }
> +
> + list_splice(&pc_list, src);
```

This would lead to LRU churning, I would recommend using list_splice_tail() instead. Since this code has a lot in common with isolate_lru_pages, it would be nice to reuse the code in vmscan.c

NOTE: Code duplication is a back door for subtle bugs and solving the same issue twice :-)

```
> + spin_unlock_irq(&rss->res.lock);
> +
> + *scanned = scan;
> + return nr_taken;
> +}
> +
> +unsigned long container_nr_physpages(struct rss_container *rss)
> +{
> + return rss->res.usage;
> +}
> +
> +static void rss_move_task(struct container_subsys *ss,
> +  struct container *cont,
> +  struct container *old_cont,
> +  struct task_struct *p)
> +{
> + struct mm_struct *mm;
> + struct rss_container *rss, *old_rss;
> +
> + mm = get_task_mm(p);
> + if (mm == NULL)
> +  goto out;
> +
```

> + rss = rss_from_cont(cont);
> + old_rss = rss_from_cont(old_cont);
> + if (old_rss != mm->rss_container)
> + goto out_put;
> +
> + css_get_current(&rss->css);
> + rcu_assign_pointer(mm->rss_container, rss);
> + css_put(&old_rss->css);
> +

I see that the charges are not migrated. Is that good?
If a user could find a way of migrating his/her task from
one container to another, it could create an issue with
the user's task taking up a big chunk of the RSS limit.

Can we migrate any task or just the thread group leader.
In my patches, I allowed migration of just the thread
group leader. Imagine if you have several threads, no
matter which container they belong to, their mm gets
charged (usage will not show up in the container's usage).
This could confuse the system administrator.

> +out_put:
> + mmput(mm);
> +out:
> + return;
> +}
> +
> +static int rss_create(struct container_subsys *ss, struct container *cont)
> +{
> + struct rss_container *rss;
> +
> + rss = kzalloc(sizeof(struct rss_container), GFP_KERNEL);
> + if (rss == NULL)
> + return -ENOMEM;
> +
> + res_counter_init(&rss->res);
> + INIT_LIST_HEAD(&rss->page_list);
> + cont->subsys[rss_subsys.subsys_id] = &rss->css;
> + return 0;
> +}
> +
> +static void rss_destroy(struct container_subsys *ss,
> + struct container *cont)
> +{
> + kfree(rss_from_cont(cont));
> +}
> +

```
> +
> +static ssize_t rss_read(struct container *cont, struct cftype *cft,
> +  struct file *file, char __user *userbuf,
> +  size_t nbytes, loff_t *ppos)
> +{
> + return res_counter_read(&rss_from_cont(cont)->res, cft->private,
> +   userbuf, nbytes, ppos);
> +}
> +
> +static ssize_t rss_write(struct container *cont, struct cftype *cft,
> +  struct file *file, const char __user *userbuf,
> +  size_t nbytes, loff_t *ppos)
> +{
> + return res_counter_write(&rss_from_cont(cont)->res, cft->private,
> +   userbuf, nbytes, ppos);
> +}
> +
> +
> +static struct cftype rss_usage = {
> + .name = "rss_usage",
> + .private = RES_USAGE,
> + .read = rss_read,
> +};
> +
> +static struct cftype rss_limit = {
> + .name = "rss_limit",
> + .private = RES_LIMIT,
> + .read = rss_read,
> + .write = rss_write,
> +};
> +
> +static struct cftype rss_failcnt = {
> + .name = "rss_failcnt",
> + .private = RES_FAILCNT,
> + .read = rss_read,
> +};
> +
> +static int rss_populate(struct container_subsys *ss,
> +  struct container *cont)
> +{
> + int rc;
> +
> + if ((rc = container_add_file(cont, &rss_usage)) < 0)
> +  return rc;
> + if ((rc = container_add_file(cont, &rss_failcnt)) < 0)
> +  return rc;
> + if ((rc = container_add_file(cont, &rss_limit)) < 0)
> +  return rc;
```

```
> +
> + return 0;
> +}
> +
> +static struct rss_container init_rss_container;
> +
> +static __init int rss_create_early(struct container_subsys *ss,
> +  struct container *cont)
> +{
> + struct rss_container *rss;
> +
> + rss = &init_rss_container;
> + res_counter_init(&rss->res);
> + INIT_LIST_HEAD(&rss->page_list);
> + cont->subsys[rss_subsys.subsys_id] = &rss->css;
> + ss->create = rss_create;
> + return 0;
> +}
> +
> +static struct container_subsys rss_subsys = {
> + .name = "rss",
> + .create = rss_create_early,
> + .destroy = rss_destroy,
> + .populate = rss_populate,
> + .attach = rss_move_task,
> +};
> +
> +void __init container_rss_init_early(void)
> +{
> + container_register_subsys(&rss_subsys);
> + init_mm.rss_container = rss_from_cont(
> +   task_container(&init_task, &rss_subsys));
> + css_get_current(&init_mm.rss_container->css);
> +}
>
```

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

Subject: Re: [RFC][PATCH 0/7] Resource controllers based on process containers
Posted by Balbir Singh on Wed, 07 Mar 2007 06:52:02 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> This patchset adds RSS, accounting and control and
> limiting the number of tasks and files within container.
>
> Based on top of Paul Menage's container subsystem v7
>
> RSS controller includes per-container RSS accounter,
> reclamation and OOM killer. It behaves like standalone
> machine - when container runs out of resources it tries
> to reclaim some pages and if it doesn't succeed in it
> kills some task which mm_struct belongs to container in
> question.
>
> Num tasks and files containers are very simple and
> self-descriptive from code.
>
> As discussed before when a task moves from one container
> to another no resources follow it - they keep holding the
> container they were allocated in.
>

I have one problem with the patchset, I cannot compile
the patches individually and some of the code is hard
to read as it depends on functions from future patches.
Patch 2, 3 and 4 fail to compile without patch 5 applied.

Patch 1 failed to apply with a reject in kernel/Makefile
I applied it on top of 2.6.20 with all of Paul Menage's
patches (all 7).

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

Subject: Re: [RFC][PATCH 6/7] Account for the number of tasks within container
Posted by xemul on Wed, 07 Mar 2007 07:10:56 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> Hi Pavel,
>
> On 3/6/07, Pavel Emelianov <xemul@sw.ru> wrote:
>> diff -upr linux-2.6.20.orig/include/linux/sched.h

>> linux-2.6.20-0/include/linux/sched.h
>> --- linux-2.6.20.orig/include/linux/sched.h 2007-03-06
>> 13:33:28.000000000 +0300
>> +++ linux-2.6.20-0/include/linux/sched.h 2007-03-06
>> 13:33:28.000000000 +0300
>> @@ -1052,6 +1055,9 @@ struct task_struct {
>> #ifdef CONFIG_FAULT_INJECTION
>> int make_it_fail;
>> #endif
>> +#ifdef CONFIG_PROCESS_CONTAINER
>> + struct numproc_container *numproc_cnt;
>> +#endif
>> };
>
> Why do you need a pointer added to task_struct? One of the main points
> of the generic containers is to avoid every different subsystem and
> resource controller having to add new pointers there.
>
>> +
>> + rcu_read_lock();
>> + np = numproc_from_cont(task_container(current, &numproc_subsys));
>> + css_get_current(&np->css);
>
> There's no need to hold a reference here - by definition, the task's
> container can't go away while the task is in it.
>
> Also, shouldn't you have an attach() method to move the count from one
> container to another when a task moves?

The idea is:

Task may be "the entity that allocates the resources" and "the
entity that is a resource allocated".

When task is the first entity it may move across containers
(that is implemented in your patches). When task is a resource
it shouldn't move across containers like files or pages do.

More generally - allocated resources hold reference to original
container till they die. No resource migration is performed.

Did I express my idea cleanly?

> Paul
>

Subject: Re: [RFC][PATCH 1/7] Resource counters
Posted by xemul on Wed, 07 Mar 2007 07:17:00 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelianov wrote:
>> Introduce generic structures and routines for
>> resource accounting.
>>
>> Each resource accounting container is supposed to
>> aggregate it, container_subsystem_state and its
>> resource-specific members within.
>>
>>
>> ---------- ----------
>>
>> diff -upr linux-2.6.20.orig/include/linux/res_counter.h
>> linux-2.6.20-0/include/linux/res_counter.h
>> --- linux-2.6.20.orig/include/linux/res_counter.h   2007-03-06
>> 13:39:17.000000000 +0300
>> +++ linux-2.6.20-0/include/linux/res_counter.h   2007-03-06
>> 13:33:28.000000000 +0300
>> @@ -0,0 +1,83 @@
>> +#ifndef __RES_COUNTER_H__
>> +#define __RES_COUNTER_H__
>> +/*
>> + * resource counters
>> + *
>> + * Copyright 2007 OpenVZ SWsoft Inc
>> + *
>> + * Author: Pavel Emelianov <xemul@openvz.org>
>> + *
>> + */
>> +
>> +#include <linux/container.h>
>> +
>> +struct res_counter {
>> +   unsigned long usage;
>> +   unsigned long limit;
>> +   unsigned long failcnt;
>> +   spinlock_t lock;
>> +};
>> +
>> +enum {
>> +   RES_USAGE,
>> +   RES_LIMIT,
>> +   RES_FAILCNT,
>> +};
>> +

```
>> +ssize_t res_counter_read(struct res_counter *cnt, int member,
>> +        const char __user *buf, size_t nbytes, loff_t *pos);
>> +ssize_t res_counter_write(struct res_counter *cnt, int member,
>> +        const char __user *buf, size_t nbytes, loff_t *pos);
>> +
>> +static inline void res_counter_init(struct res_counter *cnt)
>> +{
>> +    spin_lock_init(&cnt->lock);
>> +    cnt->limit = (unsigned long)LONG_MAX;
>> +}
>> +
>
> Is there any way to indicate that there are no limits on this container.
```

Yes - LONG_MAX is essentially a "no limit" value as no
container will ever have such many files :)

```
> LONG_MAX is quite huge, but still when the administrator wants to
> configure a container to *un-limited usage*, it becomes hard for
> the administrator.
>
>> +static inline int res_counter_charge_locked(struct res_counter *cnt,
>> +        unsigned long val)
>> +{
>> +    if (cnt->usage <= cnt->limit - val) {
>> +        cnt->usage += val;
>> +        return 0;
>> +    }
>> +
>> +    cnt->failcnt++;
>> +    return -ENOMEM;
>> +}
>> +
>> +static inline int res_counter_charge(struct res_counter *cnt,
>> +        unsigned long val)
>> +{
>> +    int ret;
>> +    unsigned long flags;
>> +
>> +    spin_lock_irqsave(&cnt->lock, flags);
>> +    ret = res_counter_charge_locked(cnt, val);
>> +    spin_unlock_irqrestore(&cnt->lock, flags);
>> +    return ret;
>> +}
>> +
>
> Will atomic counters help here.
```

I'm afraid no. We have to atomically check for limit and alter
one of usage or failcnt depending on the checking result. Making
this with atomic_xxx ops will require at least two ops.

If we'll remove failcnt this would look like
  while (atomic_cmpxchg(...))
which is also not that good.

Moreover - in RSS accounting patches I perform page list
manipulations under this lock, so this also saves one atomic op.

>> +static inline void res_counter_uncharge_locked(struct res_counter *cnt,
>> +   unsigned long val)
>> +{
>> +   if (unlikely(cnt->usage < val)) {
>> +   WARN_ON(1);
>> +   val = cnt->usage;
>> +   }
>> +
>> +   cnt->usage -= val;
>> +}
>> +
>> +static inline void res_counter_uncharge(struct res_counter *cnt,
>> +   unsigned long val)
>> +{
>> +   unsigned long flags;
>> +
>> +   spin_lock_irqsave(&cnt->lock, flags);
>> +   res_counter_uncharge_locked(cnt, val);
>> +   spin_unlock_irqrestore(&cnt->lock, flags);
>> +}
>> +
>> +#endif
>> diff -upr linux-2.6.20.orig/init/Kconfig linux-2.6.20-0/init/Kconfig
>> --- linux-2.6.20.orig/init/Kconfig   2007-03-06 13:33:28.000000000 +0300
>> +++ linux-2.6.20-0/init/Kconfig   2007-03-06 13:33:28.000000000 +0300
>> @@ -265,6 +265,10 @@ config CPUSETS
>>
>>   Say N if unsure.
>>
>> +config RESOURCE_COUNTERS
>> +   bool
>> +   select CONTAINERS
>> +
>> config SYSFS_DEPRECATED
>>   bool "Create deprecated sysfs files"
>>   default y
>> diff -upr linux-2.6.20.orig/kernel/Makefile

```
>> linux-2.6.20-0/kernel/Makefile
>> --- linux-2.6.20.orig/kernel/Makefile    2007-03-06 13:33:28.000000000
>> +0300
>> +++ linux-2.6.20-0/kernel/Makefile    2007-03-06 13:33:28.000000000 +0300
>> @@ -51,6 +51,7 @@ obj-$(CONFIG_RELAY) += relay.o
>>  obj-$(CONFIG_UTS_NS) += utsname.o
>>  obj-$(CONFIG_TASK_DELAY_ACCT) += delayacct.o
>>  obj-$(CONFIG_TASKSTATS) += taskstats.o tsacct.o
>> +obj-$(CONFIG_RESOURCE_COUNTERS) += res_counter.o
>>
>>  ifneq ($(CONFIG_SCHED_NO_NO_OMIT_FRAME_POINTER),y)
>>  # According to Alan Modra <alan@linuxcare.com.au>, the
>> -fno-omit-frame-pointer is
>> diff -upr linux-2.6.20.orig/kernel/res_counter.c
>> linux-2.6.20-0/kernel/res_counter.c
>> --- linux-2.6.20.orig/kernel/res_counter.c    2007-03-06
>> 13:39:17.000000000 +0300
>> +++ linux-2.6.20-0/kernel/res_counter.c    2007-03-06
>> 13:33:28.000000000 +0300
>> @@ -0,0 +1,72 @@
>> +/*
>> + * resource containers
>> + *
>> + * Copyright 2007 OpenVZ SWsoft Inc
>> + *
>> + * Author: Pavel Emelianov <xemul@openvz.org>
>> + *
>> + */
>> +
>> +#include <linux/parser.h>
>> +#include <linux/fs.h>
>> +#include <linux/res_counter.h>
>> +#include <asm/uaccess.h>
>> +
>> +static inline unsigned long *res_counter_member(struct res_counter
>> *cnt, int member)
>> +{
>> +    switch (member) {
>> +    case RES_USAGE:
>> +        return &cnt->usage;
>> +    case RES_LIMIT:
>> +        return &cnt->limit;
>> +    case RES_FAILCNT:
>> +        return &cnt->failcnt;
>> +    };
>> +
>> +    BUG();
>> +    return NULL;
```

```
>> +}
>> +
>> +ssize_t res_counter_read(struct res_counter *cnt, int member,
>> +        const char __user *userbuf, size_t nbytes, loff_t *pos)
>> +{
>> +    unsigned long *val;
>> +    char buf[64], *s;
>> +
>> +    s = buf;
>> +    val = res_counter_member(cnt, member);
>> +    s += sprintf(s, "%lu\n", *val);
>> +    return simple_read_from_buffer((void __user *)userbuf, nbytes,
>> +            pos, buf, s - buf);
>> +}
>> +
>> +ssize_t res_counter_write(struct res_counter *cnt, int member,
>> +        const char __user *userbuf, size_t nbytes, loff_t *pos)
>> +{
>> +    int ret;
>> +    char *buf, *end;
>> +    unsigned long tmp, *val;
>> +
>> +    buf = kmalloc(nbytes + 1, GFP_KERNEL);
>> +    ret = -ENOMEM;
>> +    if (buf == NULL)
>> +        goto out;
>> +
>> +    buf[nbytes] = 0;
>> +    ret = -EFAULT;
>> +    if (copy_from_user(buf, userbuf, nbytes))
>> +        goto out_free;
>> +
>> +    ret = -EINVAL;
>> +    tmp = simple_strtoul(buf, &end, 10);
>> +    if (*end != '\0')
>> +        goto out_free;
>> +
>> +    val = res_counter_member(cnt, member);
>> +    *val = tmp;
>> +    ret = nbytes;
>> +out_free:
>> +    kfree(buf);
>> +out:
>> +    return ret;
>> +}
>>
>
>
```

> These bits look a little out of sync, with no users for these routines in
> this patch. Won't you get a compiler warning, compiling this bit alone?
>

Nope - when you have a non-static function without users in a
file no compiler warning produced.

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by xemul on Wed, 07 Mar 2007 07:25:25 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelianov wrote:
>> This includes setup of RSS container within generic
>> process containers, all the declarations used in RSS
>> accounting, and core code responsible for accounting.
>>
>>
>> ---------- ----------
>>
>> diff -upr linux-2.6.20.orig/include/linux/rss_container.h
>> linux-2.6.20-0/include/linux/rss_container.h
>> --- linux-2.6.20.orig/include/linux/rss_container.h   2007-03-06
>> 13:39:17.000000000 +0300
>> +++ linux-2.6.20-0/include/linux/rss_container.h   2007-03-06
>> 13:33:28.000000000 +0300
>> @@ -0,0 +1,68 @@
>> +#ifndef __RSS_CONTAINER_H__
>> +#define __RSS_CONTAINER_H__
>> +/*
>> + * RSS container
>> + *
>> + * Copyright 2007 OpenVZ SWsoft Inc
>> + *
>> + * Author: Pavel Emelianov <xemul@openvz.org>
>> + *
>> + */
>> +
>> +struct page_container;
>> +struct rss_container;
>> +
>> +#ifdef CONFIG_RSS_CONTAINER
>> +int container_rss_prepare(struct page *, struct vm_area_struct *vma,
>> +   struct page_container **);
>> +
>> +void container_rss_add(struct page_container *);
>> +void container_rss_del(struct page_container *);

```
>> +void container_rss_release(struct page_container *);
>> +
>> +int mm_init_container(struct mm_struct *mm, struct task_struct *tsk);
>> +void mm_free_container(struct mm_struct *mm);
>> +
>> +unsigned long container_isolate_pages(unsigned long nr_to_scan,
>> +        struct rss_container *rss, struct list_head *dst,
>> +        int active, unsigned long *scanned);
>> +unsigned long container_nr_physpages(struct rss_container *rss);
>> +
>> +unsigned long container_try_to_free_pages(struct rss_container *);
>> +void container_out_of_memory(struct rss_container *);
>> +
>> +void container_rss_init_early(void);
>> +#else
>> +static inline int container_rss_prepare(struct page *pg,
>> +        struct vm_area_struct *vma, struct page_container **pc)
>> +{
>> +    *pc = NULL; /* to make gcc happy */
>> +    return 0;
>> +}
>> +
>> +static inline void container_rss_add(struct page_container *pc)
>> +{
>> +}
>> +
>> +static inline void container_rss_del(struct page_container *pc)
>> +{
>> +}
>> +
>> +static inline void container_rss_release(struct page_container *pc)
>> +{
>> +}
>> +
>> +static inline int mm_init_container(struct mm_struct *mm, struct
>> task_struct *t)
>> +{
>> +    return 0;
>> +}
>> +
>> +static inline void mm_free_container(struct mm_struct *mm)
>> +{
>> +}
>> +
>> +static inline void container_rss_init_early(void)
>> +{
>> +}
>> +#endif
```

>> +#endif
>> diff -upr linux-2.6.20.orig/init/Kconfig linux-2.6.20-0/init/Kconfig
>> --- linux-2.6.20.orig/init/Kconfig   2007-03-06 13:33:28.000000000 +0300
>> +++ linux-2.6.20-0/init/Kconfig   2007-03-06 13:33:28.000000000 +0300
>> @@ -265,6 +265,13 @@ config CPUSETS
>>   bool
>>   select CONTAINERS
>>
>> +config RSS_CONTAINER
>> +   bool "RSS accounting container"
>> +   select RESOURCE_COUNTERS
>> +   help
>> +   Provides a simple Resource Controller for monitoring and
>> +   controlling the total Resident Set Size of the tasks in a
>> container
>> +
>
> The wording looks very familiar :-). It would be useful to add
> "The reclaim logic is now container aware, when the container goes
> overlimit
> the page reclaimer reclaims pages belonging to this container. If we are
> unable to reclaim enough pages to satisfy the request, the process is
> killed with an out of memory warning"

OK. Thanks.

>
>> config SYSFS_DEPRECATED
>>   bool "Create deprecated sysfs files"
>>   default y
>> diff -upr linux-2.6.20.orig/mm/Makefile linux-2.6.20-0/mm/Makefile
>> --- linux-2.6.20.orig/mm/Makefile   2007-02-04 21:44:54.000000000 +0300
>> +++ linux-2.6.20-0/mm/Makefile   2007-03-06 13:33:28.000000000 +0300
>> @@ -29,3 +29,5 @@ obj-$(CONFIG_MEMORY_HOTPLUG) += memory_h
>> obj-$(CONFIG_FS_XIP) += filemap_xip.o
>> obj-$(CONFIG_MIGRATION) += migrate.o
>> obj-$(CONFIG_SMP) += allocpercpu.o
>> +
>> +obj-$(CONFIG_RSS_CONTAINER) += rss_container.o
>> diff -upr linux-2.6.20.orig/mm/rss_container.c
>> linux-2.6.20-0/mm/rss_container.c
>> --- linux-2.6.20.orig/mm/rss_container.c   2007-03-06
>> 13:39:17.000000000 +0300
>> +++ linux-2.6.20-0/mm/rss_container.c   2007-03-06 13:33:28.000000000
>> +0300
>> @@ -0,0 +1,307 @@
>> +/*
>> + * RSS accounting container

```
>> + *
>> + * Copyright 2007 OpenVZ SWsoft Inc
>> + *
>> + * Author: Pavel Emelianov <xemul@openvz.org>
>> + *
>> + */
>> +
>> +#include <linux/list.h>
>> +#include <linux/sched.h>
>> +#include <linux/mm.h>
>> +#include <linux/res_counter.h>
>> +#include <linux/rss_container.h>
>> +
>> +static struct container_subsys rss_subsys;
>> +
>> +struct rss_container {
>> +    struct res_counter res;
>> +    struct list_head page_list;
>> +    struct container_subsys_state css;
>> +};
>> +
>> +struct page_container {
>> +    struct page *page;
>> +    struct rss_container *cnt;
>> +    struct list_head list;
>> +};
>> +
>
> Yes, this is what I was planning to get to -- a per container LRU list.
> But you have just one list, don't you need active and inactive lists?
> When the global LRU is manipulated, shouldn't this list be updated as
> well, so that reclaim will pick the best pages.
>
>> +static inline struct rss_container *rss_from_cont(struct container *cnt)
>> +{
>> +    return container_of(container_subsys_state(cnt, &rss_subsys),
>> +            struct rss_container, css);
>> +}
>> +
>> +int mm_init_container(struct mm_struct *mm, struct task_struct *tsk)
>> +{
>> +    struct rss_container *cnt;
>> +
>> +    cnt = rss_from_cont(task_container(tsk, &rss_subsys));
>> +    if (css_get(&cnt->css))
>> +        return -EBUSY;
>> +
>> +    mm->rss_container = cnt;
```

```
>> +    return 0;
>> +}
>> +
>> +void mm_free_container(struct mm_struct *mm)
>> +{
>> +    css_put(&mm->rss_container->css);
>> +}
>> +
>> +int container_rss_prepare(struct page *page, struct vm_area_struct *vma,
>> +        struct page_container **ppc)
>> +{
>> +    struct rss_container *rss;
>> +    struct page_container *pc;
>> +
>> +    rcu_read_lock();
>> +    rss = rcu_dereference(vma->vm_mm->rss_container);
>> +    css_get_current(&rss->css);
>> +    rcu_read_unlock();
>> +
>> +    pc = kmalloc(sizeof(struct page_container), GFP_KERNEL);
>> +    if (pc == NULL)
>> +        goto out_nomem;
>> +
>> +    while (res_counter_charge(&rss->res, 1)) {
>> +        if (container_try_to_free_pages(rss))
>> +            continue;
>> +
>
> The return codes of the functions is a bit confusing, ideally
> container_try_to_free_pages() should return 0 on success. Also
```

This returns exactly what try_to_free_pages() does.

> res_counter_charge() has a WARN_ON(1) if the limit is exceeded.

Nope - res_counter_uncharge() has - this is an absolutely
sane check that we haven't over-uncharged resources.

> The system administrator can figure out the details from failcnt,
> I suspect when the container is running close to it's limit,
> dmesg will have too many WARNING messages.
>
> How much memory do you try to reclaim in container_try_to_free_pages()?

At least one page. This is enough to make one page charge.
That's the difference from general try_to_free_pages() that
returns success if it freed swap_cluster_max pages at least.

> With my patches, I was planning to export this knob to userspace with
> a default value. This will help the administrator decide how much
> of the working set/container LRU should be freed on reaching the limit.
> I cannot find the definition of container_try_to_free_pages() in
> this patch.

This is in patch #5.
Sorry for such a bad split - I'll make it cleaner next time :)

>
>
>> +		container_out_of_memory(rss);
>> +		if (test_thread_flag(TIF_MEMDIE))
>> +			goto out_charge;
>> +	}
>> +
>> +	pc->page = page;
>> +	pc->cnt = rss;
>> +	*ppc = pc;
>> +	return 0;
>> +
>> +out_charge:
>> +	kfree(pc);
>> +out_nomem:
>> +	css_put(&rss->css);
>> +	return -ENOMEM;
>> +}
>> +
>> +void container_rss_release(struct page_container *pc)
>> +{
>> +	struct rss_container *rss;
>> +
>> +	rss = pc->cnt;
>> +	res_counter_uncharge(&rss->res, 1);
>> +	css_put(&rss->css);
>> +	kfree(pc);
>> +}
>> +
>> +void container_rss_add(struct page_container *pc)
>> +{
>> +	struct page *pg;
>> +	struct rss_container *rss;
>> +
>> +	pg = pc->page;
>> +	rss = pc->cnt;
>> +
>> +	spin_lock(&rss->res.lock);
>> +	list_add(&pc->list, &rss->page_list);

>
> This is not good, it won't give us LRU behaviour which is
> useful for determining which pages to free.

Why not - recently used pages are in the head of the list.
Active/incative state of the page is determined from it's flags.

The idea of this list is to decrease the number of pages scanned
during reclamation. LRU-ness is checked from global page state.

```
>> +    spin_unlock(&rss->res.lock);
>> +
>> +    page_container(pg) = pc;
>> +}
>> +
>> +void container_rss_del(struct page_container *pc)
>> +{
>> +    struct page *page;
>> +    struct rss_container *rss;
>> +
>> +    page = pc->page;
>> +    rss = pc->cnt;
>> +
>> +    spin_lock(&rss->res.lock);
>> +    list_del(&pc->list);
>> +    res_counter_uncharge_locked(&rss->res, 1);
>> +    spin_unlock(&rss->res.lock);
>> +
>> +    css_put(&rss->css);
>> +    kfree(pc);
>> +}
>> +
>> +unsigned long container_isolate_pages(unsigned long nr_to_scan,
>> +        struct rss_container *rss, struct list_head *dst,
>> +        int active, unsigned long *scanned)
>> +{
>> +    unsigned long nr_taken = 0;
>> +    struct page *page;
>> +    struct page_container *pc;
>> +    unsigned long scan;
>> +    struct list_head *src;
>> +    LIST_HEAD(pc_list);
>> +    struct zone *z;
>> +
>> +    spin_lock_irq(&rss->res.lock);
>> +    src = &rss->page_list;
>> +
>
```

> Which part of the working set are we pushing out, this looks like
> we are using FIFO to determine which pages to reclaim. This needs
> to be FIXED.

This algo works exactly like general try_to_free_pages() does.
It scans for active pages first, then for inactive shrinking them.

```
>> +    for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
>> +        pc = list_entry(src->prev, struct page_container, list);
>> +        page = pc->page;
>> +        z = page_zone(page);
>> +
>> +        list_move(&pc->list, &pc_list);
>> +
>> +        spin_lock(&z->lru_lock);
>> +        if (PageLRU(page)) {
>> +            if ((active && PageActive(page)) ||
>> +                    (!active && !PageActive(page))) {
>> +                if (likely(get_page_unless_zero(page))) {
>> +                    ClearPageLRU(page);
>> +                    nr_taken++;
>> +                    list_move(&page->lru, dst);
>> +                }
>> +            }
>> +        }
>> +        spin_unlock(&z->lru_lock);
>> +    }
>> +
>> +    list_splice(&pc_list, src);
>
> This would lead to LRU churning, I would recommend using list_splice_tail()
> instead. Since this code has a lot in common with isolate_lru_pages, it
> would be nice to reuse the code in vmscan.c
```

I'll look at it.

```
> NOTE: Code duplication is a back door for subtle bugs and solving the same
> issue twice :-)
>
>> +    spin_unlock_irq(&rss->res.lock);
>> +
>> +    *scanned = scan;
>> +    return nr_taken;
>> +}
>> +
>> +unsigned long container_nr_physpages(struct rss_container *rss)
>> +{
>> +    return rss->res.usage;
```

```
>> +}
>> +
>> +static void rss_move_task(struct container_subsys *ss,
>> +        struct container *cont,
>> +        struct container *old_cont,
>> +        struct task_struct *p)
>> +{
>> +    struct mm_struct *mm;
>> +    struct rss_container *rss, *old_rss;
>> +
>> +    mm = get_task_mm(p);
>> +    if (mm == NULL)
>> +        goto out;
>> +
>> +    rss = rss_from_cont(cont);
>> +    old_rss = rss_from_cont(old_cont);
>> +    if (old_rss != mm->rss_container)
>> +        goto out_put;
>> +
>> +    css_get_current(&rss->css);
>> +    rcu_assign_pointer(mm->rss_container, rss);
>> +    css_put(&old_rss->css);
>> +
>
> I see that the charges are not migrated. Is that good?
```

This is what we came to long time ago - no resource migration.

```
> If a user could find a way of migrating his/her task from
> one container to another, it could create an issue with
> the user's task taking up a big chunk of the RSS limit.
>
> Can we migrate any task or just the thread group leader.
> In my patches, I allowed migration of just the thread
> group leader. Imagine if you have several threads, no
> matter which container they belong to, their mm gets
> charged (usage will not show up in the container's usage).
> This could confuse the system administrator.
```

Anyway - page migration may be done later with a
separate patch.

```
>> +out_put:
>> +    mmput(mm);
>> +out:
>> +    return;
>> +}
>> +
```

```
>> +static int rss_create(struct container_subsys *ss, struct container
>> *cont)
>> +{
>> +    struct rss_container *rss;
>> +
>> +    rss = kzalloc(sizeof(struct rss_container), GFP_KERNEL);
>> +    if (rss == NULL)
>> +        return -ENOMEM;
>> +
>> +    res_counter_init(&rss->res);
>> +    INIT_LIST_HEAD(&rss->page_list);
>> +    cont->subsys[rss_subsys.subsys_id] = &rss->css;
>> +    return 0;
>> +}
>> +
>> +static void rss_destroy(struct container_subsys *ss,
>> +        struct container *cont)
>> +{
>> +    kfree(rss_from_cont(cont));
>> +}
>> +
>> +
>> +static ssize_t rss_read(struct container *cont, struct cftype *cft,
>> +        struct file *file, char __user *userbuf,
>> +        size_t nbytes, loff_t *ppos)
>> +{
>> +    return res_counter_read(&rss_from_cont(cont)->res, cft->private,
>> +            userbuf, nbytes, ppos);
>> +}
>> +
>> +static ssize_t rss_write(struct container *cont, struct cftype *cft,
>> +        struct file *file, const char __user *userbuf,
>> +        size_t nbytes, loff_t *ppos)
>> +{
>> +    return res_counter_write(&rss_from_cont(cont)->res, cft->private,
>> +            userbuf, nbytes, ppos);
>> +}
>> +
>> +
>> +static struct cftype rss_usage = {
>> +    .name = "rss_usage",
>> +    .private = RES_USAGE,
>> +    .read = rss_read,
>> +};
>> +
>> +static struct cftype rss_limit = {
>> +    .name = "rss_limit",
>> +    .private = RES_LIMIT,
```

```
>> +    .read = rss_read,
>> +    .write = rss_write,
>> +};
>> +
>> +static struct cftype rss_failcnt = {
>> +    .name = "rss_failcnt",
>> +    .private = RES_FAILCNT,
>> +    .read = rss_read,
>> +};
>> +
>> +static int rss_populate(struct container_subsys *ss,
>> +        struct container *cont)
>> +{
>> +    int rc;
>> +
>> +    if ((rc = container_add_file(cont, &rss_usage)) < 0)
>> +        return rc;
>> +    if ((rc = container_add_file(cont, &rss_failcnt)) < 0)
>> +        return rc;
>> +    if ((rc = container_add_file(cont, &rss_limit)) < 0)
>> +        return rc;
>> +
>> +    return 0;
>> +}
>> +
>> +static struct rss_container init_rss_container;
>> +
>> +static __init int rss_create_early(struct container_subsys *ss,
>> +        struct container *cont)
>> +{
>> +    struct rss_container *rss;
>> +
>> +    rss = &init_rss_container;
>> +    res_counter_init(&rss->res);
>> +    INIT_LIST_HEAD(&rss->page_list);
>> +    cont->subsys[rss_subsys.subsys_id] = &rss->css;
>> +    ss->create = rss_create;
>> +    return 0;
>> +}
>> +
>> +static struct container_subsys rss_subsys = {
>> +    .name = "rss",
>> +    .create = rss_create_early,
>> +    .destroy = rss_destroy,
>> +    .populate = rss_populate,
>> +    .attach = rss_move_task,
>> +};
>> +
```

>> +void __init container_rss_init_early(void)
>> +{
>> +   container_register_subsys(&rss_subsys);
>> +   init_mm.rss_container = rss_from_cont(
>> +   task_container(&init_task, &rss_subsys));
>> +   css_get_current(&init_mm.rss_container->css);
>> +}
>>
>
>

Subject: Re: [RFC][PATCH 0/7] Resource controllers based on process containers
Posted by xemul on Wed, 07 Mar 2007 07:27:52 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On 3/6/07, Pavel Emelianov <xemul@sw.ru> wrote:
>> 2. Extended containers may register themselves too late.
>>   Kernel threads/helpers start forking, opening files
>>   and touching pages much earlier. This patchset
>>   workarounds this in not-so-cute manner and I'm waiting
>>   for Paul's comments on this issue.
>>
>
> Can we not make sure that each subsystem registers itself before any
> of its resources become usable? So the file counting subsystem should

Actually all the subsystems I've sent became usable very early.
Much earlier that initcalls started. I didn't found where exactly
but I can make it if we really need it.

> register at some point before filp_open() becomes usable, and the
> process counting subsystem should register before it's possible to
> fork, etc.
>
> Paul
>

Subject: Re: [RFC][PATCH 0/7] Resource controllers based on process containers
Posted by xemul on Wed, 07 Mar 2007 07:30:00 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelianov wrote:
>> This patchset adds RSS, accounting and control and

>> limiting the number of tasks and files within container.
>>
>> Based on top of Paul Menage's container subsystem v7
>>
>> RSS controller includes per-container RSS accounter,
>> reclamation and OOM killer. It behaves like standalone
>> machine - when container runs out of resources it tries
>> to reclaim some pages and if it doesn't succeed in it
>> kills some task which mm_struct belongs to container in
>> question.
>>
>> Num tasks and files containers are very simple and
>> self-descriptive from code.
>>
>> As discussed before when a task moves from one container
>> to another no resources follow it - they keep holding the
>> container they were allocated in.
>>
>
> I have one problem with the patchset, I cannot compile
> the patches individually and some of the code is hard
> to read as it depends on functions from future patches.
> Patch 2, 3 and 4 fail to compile without patch 5 applied.
>
> Patch 1 failed to apply with a reject in kernel/Makefile
> I applied it on top of 2.6.20 with all of Paul Menage's
> patches (all 7).

This sounds weird for me :(I've taken a stock 2.6.20
and applied Paul's patches. This is what this patchset
is applicable for.

Subject: Re: [RFC][PATCH 0/7] Resource controllers based on process containers
Posted by dev on Wed, 07 Mar 2007 09:30:34 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Balbir Singh wrote:
>
>>Pavel Emelianov wrote:
>>
>>>This patchset adds RSS, accounting and control and
>>>limiting the number of tasks and files within container.
>>>
>>>Based on top of Paul Menage's container subsystem v7
>>>
>>>RSS controller includes per-container RSS accounter,

>>>reclamation and OOM killer. It behaves like standalone
>>>machine - when container runs out of resources it tries
>>>to reclaim some pages and if it doesn't succeed in it
>>>kills some task which mm_struct belongs to container in
>>>question.
>>>
>>>Num tasks and files containers are very simple and
>>>self-descriptive from code.
>>>
>>>As discussed before when a task moves from one container
>>>to another no resources follow it - they keep holding the
>>>container they were allocated in.
>>>
>>
>>I have one problem with the patchset, I cannot compile
>>the patches individually and some of the code is hard
>>to read as it depends on functions from future patches.
>>Patch 2, 3 and 4 fail to compile without patch 5 applied.
>>
>>Patch 1 failed to apply with a reject in kernel/Makefile
>>I applied it on top of 2.6.20 with all of Paul Menage's
>>patches (all 7).
maybe Paul's patch should be taken w/o subsystems examples
(CKRM, UBC), i.e. first 3 patches only?

Kirill

Subject: Re: [RFC][PATCH 6/7] Account for the number of tasks within container
Posted by Paul Menage on Thu, 08 Mar 2007 13:49:27 GMT
View Forum Message <> Reply to Message

On 3/6/07, Pavel Emelianov <xemul@sw.ru> wrote:
> The idea is:
>
> Task may be "the entity that allocates the resources" and "the
> entity that is a resource allocated".
>
> When task is the first entity it may move across containers
> (that is implemented in your patches). When task is a resource
> it shouldn't move across containers like files or pages do.
>
> More generally - allocated resources hold reference to original
> container till they die. No resource migration is performed.
>
> Did I express my idea cleanly?

Yes, but I disagree with the premise. The title of your patch is

"Account for the number of tasks within container", but that's not what the subsystem does, it accounts for the number of forks within the container that aren't directly accompanied by an exit.

Ideally, resources like files and pages would be able to follow tasks as well. The reason that files and pages aren't easily migrated from one container to another is that there could be sharing involved; figuring out the sharing can be expensive, and it's not clear what to do if two users are in different containers.

But in the case of a task count, there are no such issues with sharing, so it seems to me to be more sensible (and more efficient) to just limit the number of tasks in a container.

i.e. when moving a task into a container or forking a task within a container, increment the count; when moving a task out of a container or when it exits, decrement the count.

With your approach, if you were to set the task limit of an empty container A to 1, and then move a process P from B into A, P would be able to fork a new child, since the "task count" would be 0 (as P was being charged to B still). Surely the fact that there's 1 process in A should prevent P from forking?

Paul

Subject: Re: [RFC][PATCH 1/7] Resource counters
Posted by Herbert Poetzl on Fri, 09 Mar 2007 16:37:11 GMT
View Forum Message <> Reply to Message

On Wed, Mar 07, 2007 at 10:19:05AM +0300, Pavel Emelianov wrote:
> Balbir Singh wrote:
> > Pavel Emelianov wrote:
> >> Introduce generic structures and routines for
> >> resource accounting.
> >>
> >> Each resource accounting container is supposed to
> >> aggregate it, container_subsystem_state and its
> >> resource-specific members within.
> >>
> >>
> >> ----------
> >>
> >> diff -upr linux-2.6.20.orig/include/linux/res_counter.h
> >> linux-2.6.20-0/include/linux/res_counter.h
> >> --- linux-2.6.20.orig/include/linux/res_counter.h   2007-03-06
> >> 13:39:17.000000000 +0300

> >> +++ linux-2.6.20-0/include/linux/res_counter.h 2007-03-06
> >> 13:33:28.000000000 +0300
> >> @@ -0,0 +1,83 @@
> >> +#ifndef __RES_COUNTER_H__
> >> +#define __RES_COUNTER_H__
> >> +/*
> >> + * resource counters
> >> + *
> >> + * Copyright 2007 OpenVZ SWsoft Inc
> >> + *
> >> + * Author: Pavel Emelianov <xemul@openvz.org>
> >> + *
> >> + */
> >> +
> >> +#include <linux/container.h>
> >> +
> >> +struct res_counter {
> >> +   unsigned long usage;
> >> +   unsigned long limit;
> >> +   unsigned long failcnt;
> >> +   spinlock_t lock;
> >> +};
> >> +
> >> +enum {
> >> +   RES_USAGE,
> >> +   RES_LIMIT,
> >> +   RES_FAILCNT,
> >> +};
> >> +
> >> +ssize_t res_counter_read(struct res_counter *cnt, int member,
> >> +   const char __user *buf, size_t nbytes, loff_t *pos);
> >> +ssize_t res_counter_write(struct res_counter *cnt, int member,
> >> +   const char __user *buf, size_t nbytes, loff_t *pos);
> >> +
> >> +static inline void res_counter_init(struct res_counter *cnt)
> >> +{
> >> +   spin_lock_init(&cnt->lock);
> >> +   cnt->limit = (unsigned long)LONG_MAX;
> >> +}
> >> +
> >
> > Is there any way to indicate that there are no limits on this container.
>
> Yes - LONG_MAX is essentially a "no limit" value as no
> container will ever have such many files :)

-1 or ~0 is a viable choice for userspace to
communicate 'infinite' or 'unlimited'

> > LONG_MAX is quite huge, but still when the administrator wants to
> > configure a container to *un-limited usage*, it becomes hard for
> > the administrator.
> >
> >> +static inline int res_counter_charge_locked(struct res_counter *cnt,
> >> +   unsigned long val)
> >> +{
> >> +   if (cnt->usage <= cnt->limit - val) {
> >> +   cnt->usage += val;
> >> +   return 0;
> >> +   }
> >> +
> >> +   cnt->failcnt++;
> >> +   return -ENOMEM;
> >> +}
> >> +
> >> +static inline int res_counter_charge(struct res_counter *cnt,
> >> +   unsigned long val)
> >> +{
> >> +   int ret;
> >> +   unsigned long flags;
> >> +
> >> +   spin_lock_irqsave(&cnt->lock, flags);
> >> +   ret = res_counter_charge_locked(cnt, val);
> >> +   spin_unlock_irqrestore(&cnt->lock, flags);
> >> +   return ret;
> >> +}
> >> +
> >
> > Will atomic counters help here.
>
> I'm afraid no. We have to atomically check for limit and alter
> one of usage or failcnt depending on the checking result. Making
> this with atomic_xxx ops will require at least two ops.

Linux-VServer does the accounting with atomic counters,
so that works quite fine, just do the checks at the
beginning of whatever resource allocation and the
accounting once the resource is acquired ...

> If we'll remove failcnt this would look like
>   while (atomic_cmpxchg(...))
> which is also not that good.
>
> Moreover - in RSS accounting patches I perform page list
> manipulations under this lock, so this also saves one atomic op.

it still hasn't been shown that this kind of RSS limit
doesn't add big time overhead to normal operations
(inside and outside of such a resource container)

note that the 'usual' memory accounting is much more
lightweight and serves similar purposes ...

best,
Herbert

> >> +static inline void res_counter_uncharge_locked(struct res_counter *cnt,
> >> +   unsigned long val)
> >> +{
> >> +   if (unlikely(cnt->usage < val)) {
> >> +   WARN_ON(1);
> >> +   val = cnt->usage;
> >> +   }
> >> +
> >> +   cnt->usage -= val;
> >> +}
> >> +
> >> +static inline void res_counter_uncharge(struct res_counter *cnt,
> >> +   unsigned long val)
> >> +{
> >> +   unsigned long flags;
> >> +
> >> +   spin_lock_irqsave(&cnt->lock, flags);
> >> +   res_counter_uncharge_locked(cnt, val);
> >> +   spin_unlock_irqrestore(&cnt->lock, flags);
> >> +}
> >> +
> >> +#endif
> >> diff -upr linux-2.6.20.orig/init/Kconfig linux-2.6.20-0/init/Kconfig
> >> --- linux-2.6.20.orig/init/Kconfig   2007-03-06 13:33:28.000000000 +0300
> >> +++ linux-2.6.20-0/init/Kconfig   2007-03-06 13:33:28.000000000 +0300
> >> @@ -265,6 +265,10 @@ config CPUSETS
> >>
> >>   Say N if unsure.
> >>
> >> +config RESOURCE_COUNTERS
> >> +   bool
> >> +   select CONTAINERS
> >> +
> >> config SYSFS_DEPRECATED
> >>   bool "Create deprecated sysfs files"
> >>   default y
> >> diff -upr linux-2.6.20.orig/kernel/Makefile
> >> linux-2.6.20-0/kernel/Makefile

```
> >> --- linux-2.6.20.orig/kernel/Makefile    2007-03-06 13:33:28.000000000
> >> +0300
> >> +++ linux-2.6.20-0/kernel/Makefile    2007-03-06 13:33:28.000000000 +0300
> >> @@ -51,6 +51,7 @@ obj-$(CONFIG_RELAY) += relay.o
> >>  obj-$(CONFIG_UTS_NS) += utsname.o
> >>  obj-$(CONFIG_TASK_DELAY_ACCT) += delayacct.o
> >>  obj-$(CONFIG_TASKSTATS) += taskstats.o tsacct.o
> >> +obj-$(CONFIG_RESOURCE_COUNTERS) += res_counter.o
> >>
> >>  ifneq ($(CONFIG_SCHED_NO_NO_OMIT_FRAME_POINTER),y)
> >>  # According to Alan Modra <alan@linuxcare.com.au>, the
> >> -fno-omit-frame-pointer is
> >> diff -upr linux-2.6.20.orig/kernel/res_counter.c
> >> linux-2.6.20-0/kernel/res_counter.c
> >> --- linux-2.6.20.orig/kernel/res_counter.c    2007-03-06
> >> 13:39:17.000000000 +0300
> >> +++ linux-2.6.20-0/kernel/res_counter.c    2007-03-06
> >> 13:33:28.000000000 +0300
> >> @@ -0,0 +1,72 @@
> >> +/*
> >> + * resource containers
> >> + *
> >> + * Copyright 2007 OpenVZ SWsoft Inc
> >> + *
> >> + * Author: Pavel Emelianov <xemul@openvz.org>
> >> + *
> >> + */
> >> +
> >> +#include <linux/parser.h>
> >> +#include <linux/fs.h>
> >> +#include <linux/res_counter.h>
> >> +#include <asm/uaccess.h>
> >> +
> >> +static inline unsigned long *res_counter_member(struct res_counter
> >> *cnt, int member)
> >> +{
> >> +    switch (member) {
> >> +    case RES_USAGE:
> >> +        return &cnt->usage;
> >> +    case RES_LIMIT:
> >> +        return &cnt->limit;
> >> +    case RES_FAILCNT:
> >> +        return &cnt->failcnt;
> >> +    };
> >> +
> >> +    BUG();
> >> +    return NULL;
> >> +}
```

```
> >> +
> >> +ssize_t res_counter_read(struct res_counter *cnt, int member,
> >> +        const char __user *userbuf, size_t nbytes, loff_t *pos)
> >> +{
> >> +    unsigned long *val;
> >> +    char buf[64], *s;
> >> +
> >> +    s = buf;
> >> +    val = res_counter_member(cnt, member);
> >> +    s += sprintf(s, "%lu\n", *val);
> >> +    return simple_read_from_buffer((void __user *)userbuf, nbytes,
> >> +            pos, buf, s - buf);
> >> +}
> >> +
> >> +ssize_t res_counter_write(struct res_counter *cnt, int member,
> >> +        const char __user *userbuf, size_t nbytes, loff_t *pos)
> >> +{
> >> +    int ret;
> >> +    char *buf, *end;
> >> +    unsigned long tmp, *val;
> >> +
> >> +    buf = kmalloc(nbytes + 1, GFP_KERNEL);
> >> +    ret = -ENOMEM;
> >> +    if (buf == NULL)
> >> +        goto out;
> >> +
> >> +    buf[nbytes] = 0;
> >> +    ret = -EFAULT;
> >> +    if (copy_from_user(buf, userbuf, nbytes))
> >> +        goto out_free;
> >> +
> >> +    ret = -EINVAL;
> >> +    tmp = simple_strtoul(buf, &end, 10);
> >> +    if (*end != '\0')
> >> +        goto out_free;
> >> +
> >> +    val = res_counter_member(cnt, member);
> >> +    *val = tmp;
> >> +    ret = nbytes;
> >> +out_free:
> >> +    kfree(buf);
> >> +out:
> >> +    return ret;
> >> +}
> >>
> >
> >
> > These bits look a little out of sync, with no users for these routines in
```

> > this patch. Won't you get a compiler warning, compiling this bit alone?
> >
>
> Nope - when you have a non-static function without users in a
> file no compiler warning produced.
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Herbert Poetzl on Fri, 09 Mar 2007 16:48:55 GMT
View Forum Message <> Reply to Message

On Tue, Mar 06, 2007 at 02:00:36PM -0800, Andrew Morton wrote:
> On Tue, 06 Mar 2007 17:55:29 +0300
> Pavel Emelianov <xemul@sw.ru> wrote:
>
> > +struct rss_container {
> > + struct res_counter res;
> > + struct list_head page_list;
> > + struct container_subsys_state css;
> > +};
> > +
> > +struct page_container {
> > + struct page *page;
> > + struct rss_container *cnt;
> > + struct list_head list;
> > +};
>
> ah. This looks good. I'll find a hunk of time to go through this work
> and through Paul's patches. It'd be good to get both patchsets lined
> up in -mm within a couple of weeks. But..

doesn't look so good for me, mainly becaus of the
additional per page data and per page processing

on 4GB memory, with 100 guests, 50% shared for each
guest, this basically means ~1mio pages, 500k shared
and 1500k x sizeof(page_container) entries, which
roughly boils down to ~25MB of wasted memory ...

increase the amount of shared pages and it starts

getting worse, but maybe I'm missing something here

> We need to decide whether we want to do per-container memory
> limitation via these data structures, or whether we do it via a
> physical scan of some software zone, possibly based on Mel's patches.

why not do simple page accounting (as done currently
in Linux) and use that for the limits, without
keeping the reference from container to page?

best,
Herbert

> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 5/7] Per-container OOM killer and page reclamation
Posted by Balbir Singh on Fri, 09 Mar 2007 21:21:11 GMT
View Forum Message <> Reply to Message

Hi, Pavel,

Please find my patch to add LRU behaviour to your latest RSS controller.

Balbir Singh
Linux Technology Center
IBM, ISTL

Subject: Re: [RFC][PATCH 6/7] Account for the number of tasks within container
Posted by xemul on Sun, 11 Mar 2007 08:34:24 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> On 3/6/07, Pavel Emelianov <xemul@sw.ru> wrote:
>> The idea is:
>>
>> Task may be "the entity that allocates the resources" and "the
>> entity that is a resource allocated".
>>

>> When task is the first entity it may move across containers
>> (that is implemented in your patches). When task is a resource
>> it shouldn't move across containers like files or pages do.
>>
>> More generally - allocated resources hold reference to original
>> container till they die. No resource migration is performed.
>>
>> Did I express my idea cleanly?
>
> Yes, but I disagree with the premise. The title of your patch is
> "Account for the number of tasks within container", but that's not
> what the subsystem does, it accounts for the number of forks within
> the container that aren't directly accompanied by an exit.
>
> Ideally, resources like files and pages would be able to follow tasks
> as well. The reason that files and pages aren't easily migrated from
> one container to another is that there could be sharing involved;
> figuring out the sharing can be expensive, and it's not clear what to
> do if two users are in different containers.
>
> But in the case of a task count, there are no such issues with
> sharing, so it seems to me to be more sensible (and more efficient) to
> just limit the number of tasks in a container.
>
> i.e. when moving a task into a container or forking a task within a
> container, increment the count; when moving a task out of a container
> or when it exits, decrement the count.

Sounds reasonable.
I'll take this into account when I make the next iteration.
Thanks.

> With your approach, if you were to set the task limit of an empty
> container A to 1, and then move a process P from B into A, P would be
> able to fork a new child, since the "task count" would be 0 (as P was
> being charged to B still). Surely the fact that there's 1 process in A
> should prevent P from forking?
>
> Paul
>

Subject: Re: [RFC][PATCH 5/7] Per-container OOM killer and page reclamation
Posted by xemul on Sun, 11 Mar 2007 08:39:18 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Hi, Pavel,

>
> Please find my patch to add LRU behaviour to your latest RSS controller.

Thanks for participation and additional testing :)
I'll include this into next generation of patches.

> Balbir Singh
> Linux Technology Center
> IBM, ISTL
>
>
> ---------- ----------
>
> Add LRU behaviour to the RSS controller patches posted by Pavel Emelianov
>
> http://lkml.org/lkml/2007/3/6/198
>
> which was in turn similar to the RSS controller posted by me
>
> http://lkml.org/lkml/2007/2/26/8
>
> Pavel's patches have a per container list of pages, which helps reduce
> reclaim time of the RSS controller but the per container list of pages is
> in FIFO order. I've implemented active and inactive lists per container to
> help select the right set of pages to reclaim when the container is under
> memory pressure.
>
> I've tested these patches on a ppc64 machine and they work fine for
> the minimal testing I've done.
>
> Pavel would you please include these patches in your next iteration.
>
> Comments, suggestions and further improvements are as always welcome!
>
> Signed-off-by: <balbir@in.ibm.com>
> ---
>
> include/linux/rss_container.h | 1
> mm/rss_container.c | 47 ++----------
> mm/swap.c | 5 ++++
> mm/vmscan.c | 3 ++
> 4 files changed, 44 insertions(+), 12 deletions(-)
>
> diff -puN include/linux/rss_container.h~rss-container-lru2 include/linux/rss_container.h
> --- linux-2.6.20/include/linux/rss_container.h~rss-container-lru 2 2007-03-09
22:52:56.000000000 +0530
> +++ linux-2.6.20-balbir/include/linux/rss_container.h 2007-03-10 00:39:59.000000000 +0530
> @@ -19,6 +19,7 @@ int container_rss_prepare(struct page *,

> void container_rss_add(struct page_container *);
> void container_rss_del(struct page_container *);
> void container_rss_release(struct page_container *);
> +void container_rss_move_lists(struct page *pg, bool active);
>
> int mm_init_container(struct mm_struct *mm, struct task_struct *tsk);
> void mm_free_container(struct mm_struct *mm);
> diff -puN mm/rss_container.c~rss-container-lru2 mm/rss_container.c
> --- linux-2.6.20/mm/rss_container.c~rss-container-lru2 2007-03-09 22:52:56.000000000 +0530
> +++ linux-2.6.20-balbir/mm/rss_container.c 2007-03-10 02:42:54.000000000 +0530
> @@ -17,7 +17,8 @@ static struct container_subsys rss_subsy
>
> struct rss_container {
>   struct res_counter res;
> - struct list_head page_list;
> + struct list_head inactive_list;
> + struct list_head active_list;
>   struct container_subsys_state css;
> };
>
> @@ -96,6 +97,26 @@ void container_rss_release(struct page_c
>   kfree(pc);
> }
>
> +void container_rss_move_lists(struct page *pg, bool active)
> +{
> + struct rss_container *rss;
> + struct page_container *pc;
> +
> + if (!page_mapped(pg))
> + return;
> +
> + pc = page_container(pg);
> + BUG_ON(!pc);
> + rss = pc->cnt;
> +
> + spin_lock_irq(&rss->res.lock);
> + if (active)
> + list_move(&pc->list, &rss->active_list);
> + else
> + list_move(&pc->list, &rss->inactive_list);
> + spin_unlock_irq(&rss->res.lock);
> +}
> +
> void container_rss_add(struct page_container *pc)
> {
>   struct page *pg;
> @@ -105,7 +126,7 @@ void container_rss_add(struct page_conta

```
>   rss = pc->cnt;
>
>   spin_lock(&rss->res.lock);
> - list_add(&pc->list, &rss->page_list);
> + list_add(&pc->list, &rss->active_list);
>   spin_unlock(&rss->res.lock);
>
>   page_container(pg) = pc;
> @@ -141,7 +162,10 @@ unsigned long container_isolate_pages(un
>   struct zone *z;
>
>   spin_lock_irq(&rss->res.lock);
> - src = &rss->page_list;
> + if (active)
> +  src = &rss->active_list;
> + else
> +  src = &rss->inactive_list;
>
>   for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
>    pc = list_entry(src->prev, struct page_container, list);
> @@ -152,13 +176,10 @@ unsigned long container_isolate_pages(un
>
>    spin_lock(&z->lru_lock);
>    if (PageLRU(page)) {
> -   if ((active && PageActive(page)) ||
> -     (!active && !PageActive(page))) {
> -    if (likely(get_page_unless_zero(page))) {
> -     ClearPageLRU(page);
> -     nr_taken++;
> -     list_move(&page->lru, dst);
> -    }
> +   if (likely(get_page_unless_zero(page))) {
> +    ClearPageLRU(page);
> +    nr_taken++;
> +    list_move(&page->lru, dst);
>     }
>    }
>    spin_unlock(&z->lru_lock);
> @@ -212,7 +233,8 @@ static int rss_create(struct container_s
>    return -ENOMEM;
>
>   res_counter_init(&rss->res);
> - INIT_LIST_HEAD(&rss->page_list);
> + INIT_LIST_HEAD(&rss->inactive_list);
> + INIT_LIST_HEAD(&rss->active_list);
>   cont->subsys[rss_subsys.subsys_id] = &rss->css;
>   return 0;
>  }
```

```
> @@ -284,7 +306,8 @@ static __init int rss_create_early(struc
>
>   rss = &init_rss_container;
>   res_counter_init(&rss->res);
> - INIT_LIST_HEAD(&rss->page_list);
> + INIT_LIST_HEAD(&rss->inactive_list);
> + INIT_LIST_HEAD(&rss->active_list);
>   cont->subsys[rss_subsys.subsys_id] = &rss->css;
>   ss->create = rss_create;
>   return 0;
> diff -puN mm/vmscan.c~rss-container-lru2 mm/vmscan.c
> --- linux-2.6.20/mm/vmscan.c~rss-container-lru2 2007-03-09 22:52:56.000000000 +0530
> +++ linux-2.6.20-balbir/mm/vmscan.c 2007-03-10 00:42:35.000000000 +0530
> @@ -1142,6 +1142,7 @@ static unsigned long container_shrink_pa
>     else
>      add_page_to_inactive_list(z, page);
>     spin_unlock_irq(&z->lru_lock);
> +   container_rss_move_lists(page, false);
>
>     put_page(page);
>    }
> @@ -1191,6 +1192,7 @@ static void container_shrink_pages_activ
>    list_move(&page->lru, &z->inactive_list);
>    z->nr_inactive++;
>    spin_unlock_irq(&z->lru_lock);
> +  container_rss_move_lists(page, false);
>
>    put_page(page);
>   }
> @@ -1206,6 +1208,7 @@ static void container_shrink_pages_activ
>    list_move(&page->lru, &z->active_list);
>    z->nr_active++;
>    spin_unlock_irq(&z->lru_lock);
> +  container_rss_move_lists(page, true);
>
>    put_page(page);
>   }
> diff -puN mm/swap.c~rss-container-lru2 mm/swap.c
> --- linux-2.6.20/mm/swap.c~rss-container-lru2 2007-03-10 00:42:38.000000000 +0530
> +++ linux-2.6.20-balbir/mm/swap.c 2007-03-10 01:20:39.000000000 +0530
> @@ -30,6 +30,7 @@
>  #include <linux/cpu.h>
>  #include <linux/notifier.h>
>  #include <linux/init.h>
> +#include <linux/rss_container.h>
>
>  /* How many pages do we try to swap or page in/out together? */
>  int page_cluster;
```

> @@ -140,6 +141,7 @@ int rotate_reclaimable_page(struct page
> void fastcall activate_page(struct page *page)
> {
>   struct zone *zone = page_zone(page);
> + bool moved = false;
>
>   spin_lock_irq(&zone->lru_lock);
>   if (PageLRU(page) && !PageActive(page)) {
> @@ -147,8 +149,11 @@ void fastcall activate_page(struct page
>   SetPageActive(page);
>   add_page_to_active_list(zone, page);
>   __count_vm_event(PGACTIVATE);
> + moved = true;
>   }
>   spin_unlock_irq(&zone->lru_lock);
> + if (moved)
> + container_rss_move_lists(page, true);
> }
>
> /*
> _

Subject: Re: [RFC][PATCH 1/7] Resource counters
Posted by xemul on Sun, 11 Mar 2007 09:01:42 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Wed, Mar 07, 2007 at 10:19:05AM +0300, Pavel Emelianov wrote:
>> Balbir Singh wrote:
>>> Pavel Emelianov wrote:
>>>> Introduce generic structures and routines for
>>>> resource accounting.
>>>>
>>>> Each resource accounting container is supposed to
>>>> aggregate it, container_subsystem_state and its
>>>> resource-specific members within.
>>>>
>>>>
>>>> ---
>>>>
>>>> diff -upr linux-2.6.20.orig/include/linux/res_counter.h
>>>> linux-2.6.20-0/include/linux/res_counter.h
>>>> --- linux-2.6.20.orig/include/linux/res_counter.h   2007-03-06
>>>> 13:39:17.000000000 +0300
>>>> +++ linux-2.6.20-0/include/linux/res_counter.h   2007-03-06
>>>> 13:33:28.000000000 +0300
>>>> @@ -0,0 +1,83 @@

```
>>>> +#ifndef __RES_COUNTER_H__
>>>> +#define __RES_COUNTER_H__
>>>> +/*
>>>> + * resource counters
>>>> + *
>>>> + * Copyright 2007 OpenVZ SWsoft Inc
>>>> + *
>>>> + * Author: Pavel Emelianov <xemul@openvz.org>
>>>> + *
>>>> + */
>>>> +
>>>> +#include <linux/container.h>
>>>> +
>>>> +struct res_counter {
>>>> +    unsigned long usage;
>>>> +    unsigned long limit;
>>>> +    unsigned long failcnt;
>>>> +    spinlock_t lock;
>>>> +};
>>>> +
>>>> +enum {
>>>> +    RES_USAGE,
>>>> +    RES_LIMIT,
>>>> +    RES_FAILCNT,
>>>> +};
>>>> +
>>>> +ssize_t res_counter_read(struct res_counter *cnt, int member,
>>>> +        const char __user *buf, size_t nbytes, loff_t *pos);
>>>> +ssize_t res_counter_write(struct res_counter *cnt, int member,
>>>> +        const char __user *buf, size_t nbytes, loff_t *pos);
>>>> +
>>>> +static inline void res_counter_init(struct res_counter *cnt)
>>>> +{
>>>> +    spin_lock_init(&cnt->lock);
>>>> +    cnt->limit = (unsigned long)LONG_MAX;
>>>> +}
>>>> +
>>> Is there any way to indicate that there are no limits on this container.
>> Yes - LONG_MAX is essentially a "no limit" value as no
>> container will ever have such many files :)
>
> -1 or ~0 is a viable choice for userspace to
> communicate 'infinite' or 'unlimited'

OK, I'll make ULONG_MAX :)

>>> LONG_MAX is quite huge, but still when the administrator wants to
>>> configure a container to *un-limited usage*, it becomes hard for
```

```
>>> the administrator.
>>>
>>>> +static inline int res_counter_charge_locked(struct res_counter *cnt,
>>>> +        unsigned long val)
>>>> +{
>>>> +    if (cnt->usage <= cnt->limit - val) {
>>>> +        cnt->usage += val;
>>>> +        return 0;
>>>> +    }
>>>> +
>>>> +    cnt->failcnt++;
>>>> +    return -ENOMEM;
>>>> +}
>>>> +
>>>> +static inline int res_counter_charge(struct res_counter *cnt,
>>>> +        unsigned long val)
>>>> +{
>>>> +    int ret;
>>>> +    unsigned long flags;
>>>> +
>>>> +    spin_lock_irqsave(&cnt->lock, flags);
>>>> +    ret = res_counter_charge_locked(cnt, val);
>>>> +    spin_unlock_irqrestore(&cnt->lock, flags);
>>>> +    return ret;
>>>> +}
>>>> +
>>> Will atomic counters help here.
>> I'm afraid no. We have to atomically check for limit and alter
>> one of usage or failcnt depending on the checking result. Making
>> this with atomic_xxx ops will require at least two ops.
>
> Linux-VServer does the accounting with atomic counters,
> so that works quite fine, just do the checks at the
> beginning of whatever resource allocation and the
> accounting once the resource is acquired ...

This works quite fine on non-preempted kernels.
>From the time you checked for resource till you really
account it kernel may preempt and let another process
pass through vx_anything_avail() check.

>> If we'll remove failcnt this would look like
>>    while (atomic_cmpxchg(...))
>> which is also not that good.
>>
>> Moreover - in RSS accounting patches I perform page list
>> manipulations under this lock, so this also saves one atomic op.
>
```

> it still hasn't been shown that this kind of RSS limit
> doesn't add big time overhead to normal operations
> (inside and outside of such a resource container)
>
> note that the 'usual' memory accounting is much more
> lightweight and serves similar purposes ...

It OOM-kills current int case of limit hit instead of
reclaiming pages or killing *memory eater* to free memory.

> best,
> Herbert
>
>>>> +static inline void res_counter_uncharge_locked(struct res_counter *cnt,
>>>> +   unsigned long val)
>>>> +{
>>>> +   if (unlikely(cnt->usage < val)) {
>>>> +   WARN_ON(1);
>>>> +   val = cnt->usage;
>>>> +   }
>>>> +
>>>> +   cnt->usage -= val;
>>>> +}
>>>> +
>>>> +static inline void res_counter_uncharge(struct res_counter *cnt,
>>>> +   unsigned long val)
>>>> +{
>>>> +   unsigned long flags;
>>>> +
>>>> +   spin_lock_irqsave(&cnt->lock, flags);
>>>> +   res_counter_uncharge_locked(cnt, val);
>>>> +   spin_unlock_irqrestore(&cnt->lock, flags);
>>>> +}
>>>> +
>>>> +#endif
>>>> diff -upr linux-2.6.20.orig/init/Kconfig linux-2.6.20-0/init/Kconfig
>>>> --- linux-2.6.20.orig/init/Kconfig   2007-03-06 13:33:28.000000000 +0300
>>>> +++ linux-2.6.20-0/init/Kconfig   2007-03-06 13:33:28.000000000 +0300
>>>> @@ -265,6 +265,10 @@ config CPUSETS
>>>>
>>>>   Say N if unsure.
>>>>
>>>> +config RESOURCE_COUNTERS
>>>> +   bool
>>>> +   select CONTAINERS
>>>> +
>>>> config SYSFS_DEPRECATED
>>>>   bool "Create deprecated sysfs files"

```
>>>>      default y
>>>> diff -upr linux-2.6.20.orig/kernel/Makefile
>>>> linux-2.6.20-0/kernel/Makefile
>>>> --- linux-2.6.20.orig/kernel/Makefile    2007-03-06 13:33:28.000000000
>>>> +0300
>>>> +++ linux-2.6.20-0/kernel/Makefile    2007-03-06 13:33:28.000000000 +0300
>>>> @@ -51,6 +51,7 @@ obj-$(CONFIG_RELAY) += relay.o
>>>>  obj-$(CONFIG_UTS_NS) += utsname.o
>>>>  obj-$(CONFIG_TASK_DELAY_ACCT) += delayacct.o
>>>>  obj-$(CONFIG_TASKSTATS) += taskstats.o tsacct.o
>>>> +obj-$(CONFIG_RESOURCE_COUNTERS) += res_counter.o
>>>>
>>>>  ifneq ($(CONFIG_SCHED_NO_NO_OMIT_FRAME_POINTER),y)
>>>>  # According to Alan Modra <alan@linuxcare.com.au>, the
>>>> -fno-omit-frame-pointer is
>>>> diff -upr linux-2.6.20.orig/kernel/res_counter.c
>>>> linux-2.6.20-0/kernel/res_counter.c
>>>> --- linux-2.6.20.orig/kernel/res_counter.c    2007-03-06
>>>> 13:39:17.000000000 +0300
>>>> +++ linux-2.6.20-0/kernel/res_counter.c    2007-03-06
>>>> 13:33:28.000000000 +0300
>>>> @@ -0,0 +1,72 @@
>>>> +/*
>>>> + * resource containers
>>>> + *
>>>> + * Copyright 2007 OpenVZ SWsoft Inc
>>>> + *
>>>> + * Author: Pavel Emelianov <xemul@openvz.org>
>>>> + *
>>>> + */
>>>> +
>>>> +#include <linux/parser.h>
>>>> +#include <linux/fs.h>
>>>> +#include <linux/res_counter.h>
>>>> +#include <asm/uaccess.h>
>>>> +
>>>> +static inline unsigned long *res_counter_member(struct res_counter
>>>> *cnt, int member)
>>>> +{
>>>> +    switch (member) {
>>>> +    case RES_USAGE:
>>>> +        return &cnt->usage;
>>>> +    case RES_LIMIT:
>>>> +        return &cnt->limit;
>>>> +    case RES_FAILCNT:
>>>> +        return &cnt->failcnt;
>>>> +    };
>>>> +
```

```
>>>> +	BUG();
>>>> +	return NULL;
>>>> +}
>>>> +
>>>> +ssize_t res_counter_read(struct res_counter *cnt, int member,
>>>> +		const char __user *userbuf, size_t nbytes, loff_t *pos)
>>>> +{
>>>> +	unsigned long *val;
>>>> +	char buf[64], *s;
>>>> +
>>>> +	s = buf;
>>>> +	val = res_counter_member(cnt, member);
>>>> +	s += sprintf(s, "%lu\n", *val);
>>>> +	return simple_read_from_buffer((void __user *)userbuf, nbytes,
>>>> +			pos, buf, s - buf);
>>>> +}
>>>> +
>>>> +ssize_t res_counter_write(struct res_counter *cnt, int member,
>>>> +		const char __user *userbuf, size_t nbytes, loff_t *pos)
>>>> +{
>>>> +	int ret;
>>>> +	char *buf, *end;
>>>> +	unsigned long tmp, *val;
>>>> +
>>>> +	buf = kmalloc(nbytes + 1, GFP_KERNEL);
>>>> +	ret = -ENOMEM;
>>>> +	if (buf == NULL)
>>>> +		goto out;
>>>> +
>>>> +	buf[nbytes] = 0;
>>>> +	ret = -EFAULT;
>>>> +	if (copy_from_user(buf, userbuf, nbytes))
>>>> +		goto out_free;
>>>> +
>>>> +	ret = -EINVAL;
>>>> +	tmp = simple_strtoul(buf, &end, 10);
>>>> +	if (*end != '\0')
>>>> +		goto out_free;
>>>> +
>>>> +	val = res_counter_member(cnt, member);
>>>> +	*val = tmp;
>>>> +	ret = nbytes;
>>>> +out_free:
>>>> +	kfree(buf);
>>>> +out:
>>>> +	return ret;
>>>> +}
>>>>
```

>>>
>>> These bits look a little out of sync, with no users for these routines in
>>> this patch. Won't you get a compiler warning, compiling this bit alone?
>>>
>> Nope - when you have a non-static function without users in a
>> file no compiler warning produced.
>> _____
>> Containers mailing list
>> Containers@lists.osdl.org
>> https://lists.osdl.org/mailman/listinfo/containers
>

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by xemul on Sun, 11 Mar 2007 09:08:16 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Tue, Mar 06, 2007 at 02:00:36PM -0800, Andrew Morton wrote:
>> On Tue, 06 Mar 2007 17:55:29 +0300
>> Pavel Emelianov <xemul@sw.ru> wrote:
>>
>>> +struct rss_container {
>>> + struct res_counter res;
>>> + struct list_head page_list;
>>> + struct container_subsys_state css;
>>> +};
>>> +
>>> +struct page_container {
>>> + struct page *page;
>>> + struct rss_container *cnt;
>>> + struct list_head list;
>>> +};
>> ah. This looks good. I'll find a hunk of time to go through this work
>> and through Paul's patches. It'd be good to get both patchsets lined
>> up in -mm within a couple of weeks. But..
>
> doesn't look so good for me, mainly becaus of the
> additional per page data and per page processing
>
> on 4GB memory, with 100 guests, 50% shared for each
> guest, this basically means ~1mio pages, 500k shared
> and 1500k x sizeof(page_container) entries, which

> roughly boils down to ~25MB of wasted memory ...
>
> increase the amount of shared pages and it starts
> getting worse, but maybe I'm missing something here

You are. Each page has only one page_container associated
with it despite the number of containers it is shared
between.

>> We need to decide whether we want to do per-container memory
>> limitation via these data structures, or whether we do it via a
>> physical scan of some software zone, possibly based on Mel's patches.
>
> why not do simple page accounting (as done currently
> in Linux) and use that for the limits, without
> keeping the reference from container to page?

As I've already answered in my previous letter simple
limiting w/o per-container reclamation and per-container
oom killer isn't a good memory management. It doesn't allow
to handle resource shortage gracefully.

This patchset provides more grace way to handle this, but
full memory management includes accounting of VMA-length
as well (returning ENOMEM from system call) but we've decided
to start with RSS.

> best,
> Herbert
>
>> _____
>> Containers mailing list
>> Containers@lists.osdl.org
>> https://lists.osdl.org/mailman/listinfo/containers
> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/
>

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by dev on Sun, 11 Mar 2007 12:13:08 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
> On Tue, 06 Mar 2007 17:55:29 +0300
> Pavel Emelianov <xemul@sw.ru> wrote:
>
>
>>+struct rss_container {
>>+ struct res_counter res;
>>+ struct list_head page_list;
>>+ struct container_subsys_state css;
>>+};
>>+
>>+struct page_container {
>>+ struct page *page;
>>+ struct rss_container *cnt;
>>+ struct list_head list;
>>+};
>
>
> ah. This looks good. I'll find a hunk of time to go through this work
> and through Paul's patches. It'd be good to get both patchsets lined
> up in -mm within a couple of weeks. But..
>
> We need to decide whether we want to do per-container memory limitation via
> these data structures, or whether we do it via a physical scan of some
> software zone, possibly based on Mel's patches.
i.e. a separate memzone for each container?
imho memzone approach is inconvinient for pages sharing and shares accounting.
it also makes memory management more strict, forbids overcommiting
per-container etc.
Maybe you have some ideas how we can decide on this?

Thanks,
Kirill

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Andrew Morton on Sun, 11 Mar 2007 12:51:11 GMT
View Forum Message <> Reply to Message

> On Sun, 11 Mar 2007 15:26:41 +0300 Kirill Korotaev <dev@sw.ru> wrote:
> Andrew Morton wrote:
> > On Tue, 06 Mar 2007 17:55:29 +0300
> > Pavel Emelianov <xemul@sw.ru> wrote:
> >

> >
> >>+struct rss_container {
> >>+ struct res_counter res;
> >>+ struct list_head page_list;
> >>+ struct container_subsys_state css;
> >>+};
> >>+
> >>+struct page_container {
> >>+ struct page *page;
> >>+ struct rss_container *cnt;
> >>+ struct list_head list;
> >>+};
> >
> >
> > ah. This looks good. I'll find a hunk of time to go through this work
> > and through Paul's patches. It'd be good to get both patchsets lined
> > up in -mm within a couple of weeks. But..
> >
> > We need to decide whether we want to do per-container memory limitation via
> > these data structures, or whether we do it via a physical scan of some
> > software zone, possibly based on Mel's patches.
> i.e. a separate memzone for each container?

Yep. Straightforward machine partitioning. An attractive thing is that it
100% reuses existing page reclaim, unaltered.

> imho memzone approach is inconvinient for pages sharing and shares accounting.
> it also makes memory management more strict, forbids overcommiting
> per-container etc.

umm, who said they were requirements?

> Maybe you have some ideas how we can decide on this?

We need to work out what the requirements are before we can settle on an
implementation.

Sigh. Who is running this show? Anyone?

You can actually do a form of overcommittment by allowing multiple
containers to share one or more of the zones. Whether that is sufficient
or suitable I don't know. That depends on the requirements, and we haven't
even discussed those, let alone agreed to them.

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Herbert Poetzl on Sun, 11 Mar 2007 14:32:55 GMT

On Sun, Mar 11, 2007 at 12:08:16PM +0300, Pavel Emelianov wrote:
> Herbert Poetzl wrote:
>> On Tue, Mar 06, 2007 at 02:00:36PM -0800, Andrew Morton wrote:
>>> On Tue, 06 Mar 2007 17:55:29 +0300
>>> Pavel Emelianov <xemul@sw.ru> wrote:
>>>
>>>> +struct rss_container {
>>>> + struct res_counter res;
>>>> + struct list_head page_list;
>>>> + struct container_subsys_state css;
>>>> +};
>>>> +
>>>> +struct page_container {
>>>> + struct page *page;
>>>> + struct rss_container *cnt;
>>>> + struct list_head list;
>>>> +};
>>> ah. This looks good. I'll find a hunk of time to go through this
>>> work and through Paul's patches. It'd be good to get both patchsets
>>> lined up in -mm within a couple of weeks. But..
>>
>> doesn't look so good for me, mainly becaus of the
>> additional per page data and per page processing
>>
>> on 4GB memory, with 100 guests, 50% shared for each
>> guest, this basically means ~1mio pages, 500k shared
>> and 1500k x sizeof(page_container) entries, which
>> roughly boils down to ~25MB of wasted memory ...
>>
>> increase the amount of shared pages and it starts
>> getting worse, but maybe I'm missing something here
>
> You are. Each page has only one page_container associated
> with it despite the number of containers it is shared
> between.
>
>>> We need to decide whether we want to do per-container memory
>>> limitation via these data structures, or whether we do it via
>>> a physical scan of some software zone, possibly based on Mel's
>>> patches.
>>
>> why not do simple page accounting (as done currently
>> in Linux) and use that for the limits, without
>> keeping the reference from container to page?
>
> As I've already answered in my previous letter simple
> limiting w/o per-container reclamation and per-container

> oom killer isn't a good memory management. It doesn't allow
> to handle resource shortage gracefully.

per container OOM killer does not require any container
page reference, you know _what_ tasks belong to the
container, and you know their _badness_ from the normal
OOM calculations, so doing them for a container is really
straight forward without having any page 'tagging'

for the reclamation part, please elaborate how that will
differ in a (shared memory) guest from what the kernel
currently does ...

TIA,
Herbert

> This patchset provides more grace way to handle this, but
> full memory management includes accounting of VMA-length
> as well (returning ENOMEM from system call) but we've decided
> to start with RSS.
>
>> best,
>> Herbert
>>
>>> _____
>>> Containers mailing list
>>> Containers@lists.osdl.org
>>> https://lists.osdl.org/mailman/listinfo/containers
>> -
>> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
>> the body of a message to majordomo@vger.kernel.org
>> More majordomo info at http://vger.kernel.org/majordomo-info.html
>> Please read the FAQ at http://www.tux.org/lkml/
>>

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by xemul on Sun, 11 Mar 2007 15:04:28 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Sun, Mar 11, 2007 at 12:08:16PM +0300, Pavel Emelianov wrote:
>> Herbert Poetzl wrote:
>>> On Tue, Mar 06, 2007 at 02:00:36PM -0800, Andrew Morton wrote:

>>>> On Tue, 06 Mar 2007 17:55:29 +0300
>>>> Pavel Emelianov <xemul@sw.ru> wrote:
>>>>
>>>>> +struct rss_container {
>>>>> + struct res_counter res;
>>>>> + struct list_head page_list;
>>>>> + struct container_subsys_state css;
>>>>> +};
>>>>> +
>>>>> +struct page_container {
>>>>> + struct page *page;
>>>>> + struct rss_container *cnt;
>>>>> + struct list_head list;
>>>>> +};
>>>> ah. This looks good. I'll find a hunk of time to go through this
>>>> work and through Paul's patches. It'd be good to get both patchsets
>>>> lined up in -mm within a couple of weeks. But..
>>> doesn't look so good for me, mainly becaus of the
>>> additional per page data and per page processing
>>>
>>> on 4GB memory, with 100 guests, 50% shared for each
>>> guest, this basically means ~1mio pages, 500k shared
>>> and 1500k x sizeof(page_container) entries, which
>>> roughly boils down to ~25MB of wasted memory ...
>>>
>>> increase the amount of shared pages and it starts
>>> getting worse, but maybe I'm missing something here
>> You are. Each page has only one page_container associated
>> with it despite the number of containers it is shared
>> between.
>>
>>>> We need to decide whether we want to do per-container memory
>>>> limitation via these data structures, or whether we do it via
>>>> a physical scan of some software zone, possibly based on Mel's
>>>> patches.
>>> why not do simple page accounting (as done currently
>>> in Linux) and use that for the limits, without
>>> keeping the reference from container to page?
>> As I've already answered in my previous letter simple
>> limiting w/o per-container reclamation and per-container
>> oom killer isn't a good memory management. It doesn't allow
>> to handle resource shortage gracefully.
>
> per container OOM killer does not require any container
> page reference, you know _what_ tasks belong to the
> container, and you know their _badness_ from the normal
> OOM calculations, so doing them for a container is really
> straight forward without having any page 'tagging'

That's true. If you look at the patches you'll
find out that no code in oom killer uses page 'tag'.

> for the reclamation part, please elaborate how that will
> differ in a (shared memory) guest from what the kernel
> currently does ...

This is all described in the code and in the
discussions we had before.

> TIA,
> Herbert
>
>> This patchset provides more grace way to handle this, but
>> full memory management includes accounting of VMA-length
>> as well (returning ENOMEM from system call) but we've decided
>> to start with RSS.
>>
>>> best,
>>> Herbert
>>>
>>>> _____
>>>> Containers mailing list
>>>> Containers@lists.osdl.org
>>>> https://lists.osdl.org/mailman/listinfo/containers
>>> -
>>> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
>>> the body of a message to majordomo@vger.kernel.org
>>> More majordomo info at http://vger.kernel.org/majordomo-info.html
>>> Please read the FAQ at http://www.tux.org/lkml/
>>>
>

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Balbir Singh on Sun, 11 Mar 2007 15:51:30 GMT
View Forum Message <> Reply to Message

On 3/11/07, Andrew Morton <akpm@linux-foundation.org> wrote:
> > On Sun, 11 Mar 2007 15:26:41 +0300 Kirill Korotaev <dev@sw.ru> wrote:
> > Andrew Morton wrote:
> > > On Tue, 06 Mar 2007 17:55:29 +0300

> > > Pavel Emelianov <xemul@sw.ru> wrote:
> > >
> > >
> > >>+struct rss_container {
> > >>+   struct res_counter res;
> > >>+   struct list_head page_list;
> > >>+   struct container_subsys_state css;
> > >>+};
> > >>+
> > >>+struct page_container {
> > >>+   struct page *page;
> > >>+   struct rss_container *cnt;
> > >>+   struct list_head list;
> > >>+};
> > >
> > >
> > > ah. This looks good. I'll find a hunk of time to go through this work
> > > and through Paul's patches. It'd be good to get both patchsets lined
> > > up in -mm within a couple of weeks. But..
> > >
> > > We need to decide whether we want to do per-container memory limitation via
> > > these data structures, or whether we do it via a physical scan of some
> > > software zone, possibly based on Mel's patches.
> > i.e. a separate memzone for each container?
>
> Yep. Straightforward machine partitioning. An attractive thing is that it
> 100% reuses existing page reclaim, unaltered.

We discussed zones for resource control and some of the disadvantages at
http://lkml.org/lkml/2006/10/30/222

I need to look at Mel's patches to determine if they are suitable for
control. But in a thread of discussion on those patches, it was agreed
that memory fragmentation and resource control are independent issues.

>
> > imho memzone approach is inconvinient for pages sharing and shares accounting.
> > it also makes memory management more strict, forbids overcommiting
> > per-container etc.
>
> umm, who said they were requirements?
>

We discussed some of the requirements in the RFC: Memory Controller
requirements thread
http://lkml.org/lkml/2006/10/30/51

> > Maybe you have some ideas how we can decide on this?
>
> We need to work out what the requirements are before we can settle on an
> implementation.
>
> Sigh. Who is running this show? Anyone?
>

All the stake holders involved in the RFC discussion :-) We've been
talking and building on top of each others patches. I hope that was a
good answer ;)

> You can actually do a form of overcommittment by allowing multiple
> containers to share one or more of the zones. Whether that is sufficient
> or suitable I don't know. That depends on the requirements, and we haven't
> even discussed those, let alone agreed to them.
>

There are other things like resizing a zone, finding the right size,
etc. I'll look
at Mel's patches to see what is supported.

Warm Regards,
Balbir Singh

Subject: Re: [RFC][PATCH 1/7] Resource counters
Posted by ebiederm on Sun, 11 Mar 2007 19:00:15 GMT
View Forum Message <> Reply to Message

Herbert Poetzl <herbert@13thfloor.at> writes:

>
> Linux-VServer does the accounting with atomic counters,
> so that works quite fine, just do the checks at the
> beginning of whatever resource allocation and the
> accounting once the resource is acquired ...

Atomic operations versus locks is only a granularity thing.
You still need the cache line which is the cost on SMP.

Are you using atomic_add_return or atomic_add_unless or
are you performing you actions in two separate steps which
is racy? What I have seen indicates you are using a racy two separate
operation form.

>> If we'll remove failcnt this would look like
>>   while (atomic_cmpxchg(...))

>> which is also not that good.
>>
>> Moreover - in RSS accounting patches I perform page list
>> manipulations under this lock, so this also saves one atomic op.
>
> it still hasn't been shown that this kind of RSS limit
> doesn't add big time overhead to normal operations
> (inside and outside of such a resource container)
>
> note that the 'usual' memory accounting is much more
> lightweight and serves similar purposes ...

Perhaps....

Eric

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 3/7] Data structures changes for RSS accounting
Posted by ebiederm on Sun, 11 Mar 2007 19:13:26 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@sw.ru> writes:

> Adds needed pointers to mm_struct and page struct,
> places hooks to core code for mm_struct initialization
> and hooks in container_init_early() to preinitialize
> RSS accounting subsystem.

An extra pointer in struct page is unlikely to fly.
Both because it increases the size of a size critical structure,
and because conceptually it is ridiculous.

If you are limiting the RSS size you are counting the number of pages in
the page tables. You don't care about the page itself.

With the rmap code it is relatively straight forward to see if this is
the first time a page has been added to a page table in your rss
group, or if this is the last reference to a particular page in your
rss group. The counters should only increment the first time a
particular page is added to your rss group. The counters should only
decrement when it is the last reference in your rss subsystem.

This allow important little cases like glibc to be properly accounted
for. One of the key features of a rss limit is that the kernel can

still keep pages that you need in-core, that are accessible with just
a minor fault. Directly owning pages works directly against that
principle.

> diff -upr linux-2.6.20.orig/include/linux/mm_types.h
> linux-2.6.20-0/include/linux/mm_types.h
> --- linux-2.6.20.orig/include/linux/mm_types.h 2007-02-04 21:44:54.000000000
> +0300
> +++ linux-2.6.20-0/include/linux/mm_types.h 2007-03-06 13:33:28.000000000 +0300
> @@ -62,6 +62,9 @@ struct page {
>   void *virtual;   /* Kernel virtual address (NULL if
>   not kmapped, ie. highmem) */
> #endif /* WANT_PAGE_VIRTUAL */
> +#ifdef CONFIG_RSS_CONTAINER
> + struct page_container *rss_container;
> +#endif
> };
>
> #endif /* _LINUX_MM_TYPES_H */

Eric

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by ebiederm on Sun, 11 Mar 2007 19:14:55 GMT

View Forum Message <> Reply to Message

Pavel Emelianov <xemul@sw.ru> writes:

> Pages are charged to their first touchers which are
> determined using pages' mapcount manipulations in
> rmap calls.

NAK pages should be charged to every rss group whose mm_struct they
are mapped into.

Eric

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by ebiederm on Sun, 11 Mar 2007 19:34:42 GMT
View Forum Message <> Reply to Message

Andrew Morton <akpm@linux-foundation.org> writes:

> Yep. Straightforward machine partitioning. An attractive thing is that it
> 100% reuses existing page reclaim, unaltered.

And misses every resource sharing opportunity in sight. Except for
filtering the which pages are eligible for reclaim an RSS limit should
not need to change the existing reclaim logic, and with things like the
memory zones we have had that kind of restriction in the reclaim logic
for a long time. So filtering out ineligible pages isn't anything new.

>> imho memzone approach is inconvinient for pages sharing and shares accounting.
>> it also makes memory management more strict, forbids overcommiting
>> per-container etc.
>
> umm, who said they were requirements?
>
>> Maybe you have some ideas how we can decide on this?
>
> We need to work out what the requirements are before we can settle on an
> implementation.

If you are talking about RSS limits the term is well defined. The
number of pages you can have mapped into your set of address space at
any given time.

Unless I'm totally blind that isn't what the patchset implements. A
true RSS limit over multiple processes has a lot of potential to be
generally useful, is very understandable, doesn't affect kernel cache
decisions so largely performance should not be affected. There is a
little more overhead in the fault logic but that is a moderately
expensive path anyway.

> Sigh. Who is running this show? Anyone?

Someone is supposed to run the show? :)

> You can actually do a form of overcommittment by allowing multiple
> containers to share one or more of the zones. Whether that is sufficient
> or suitable I don't know. That depends on the requirements, and we haven't
> even discussed those, let alone agreed to them.

Another really nasty issue is the container term as the resource guys
are using the term in a subtlety different way then it has been used
with namespaces leading to several threads where the participants talked

past each other. We need a different term to designate the group of tasks a resource controller is dealing with.

The whole filesystem interface also is over general and makes it too easy to express the hard things (like move an existing task from one group of tasks to another) leading to code complications.

On the up side I think the code the focus is likely in the right place to start delivering usable code.

Eric

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Herbert Poetzl on Mon, 12 Mar 2007 00:41:52 GMT
View Forum Message <> Reply to Message

On Sun, Mar 11, 2007 at 06:04:28PM +0300, Pavel Emelianov wrote:
> Herbert Poetzl wrote:
> > On Sun, Mar 11, 2007 at 12:08:16PM +0300, Pavel Emelianov wrote:
> >> Herbert Poetzl wrote:
> >>> On Tue, Mar 06, 2007 at 02:00:36PM -0800, Andrew Morton wrote:
> >>>> On Tue, 06 Mar 2007 17:55:29 +0300
> >>>> Pavel Emelianov <xemul@sw.ru> wrote:
> >>>>
> >>>> +struct rss_container {
> >>>> + struct res_counter res;
> >>>> + struct list_head page_list;
> >>>> + struct container_subsys_state css;
> >>>> +};
> >>>> +
> >>>> +struct page_container {
> >>>> + struct page *page;
> >>>> + struct rss_container *cnt;
> >>>> + struct list_head list;
> >>>> +};
> >>>> ah. This looks good. I'll find a hunk of time to go through this
> >>>> work and through Paul's patches. It'd be good to get both patchsets
> >>>> lined up in -mm within a couple of weeks. But..
> >>> doesn't look so good for me, mainly becaus of the
> >>> additional per page data and per page processing
> >>>
> >>> on 4GB memory, with 100 guests, 50% shared for each
> >>> guest, this basically means ~1mio pages, 500k shared

> >>> and 1500k x sizeof(page_container) entries, which
> >>> roughly boils down to ~25MB of wasted memory ...
> >>>
> >>> increase the amount of shared pages and it starts
> >>> getting worse, but maybe I'm missing something here
> >> You are. Each page has only one page_container associated
> >> with it despite the number of containers it is shared
> >> between.
> >>
> >>>> We need to decide whether we want to do per-container memory
> >>>> limitation via these data structures, or whether we do it via
> >>>> a physical scan of some software zone, possibly based on Mel's
> >>>> patches.
> >>> why not do simple page accounting (as done currently
> >>> in Linux) and use that for the limits, without
> >>> keeping the reference from container to page?
> >> As I've already answered in my previous letter simple
> >> limiting w/o per-container reclamation and per-container
> >> oom killer isn't a good memory management. It doesn't allow
> >> to handle resource shortage gracefully.
> >
> > per container OOM killer does not require any container
> > page reference, you know _what_ tasks belong to the
> > container, and you know their _badness_ from the normal
> > OOM calculations, so doing them for a container is really
> > straight forward without having any page 'tagging'
>
> That's true. If you look at the patches you'll
> find out that no code in oom killer uses page 'tag'.

so what do we keep the context -> page reference
then at all?

> > for the reclamation part, please elaborate how that will
> > differ in a (shared memory) guest from what the kernel
> > currently does ...
>
> This is all described in the code and in the
> discussions we had before.

must have missed some of them, please can you
point me to the relevant threads ...

TIA,
Herbert

> > TIA,
> > Herbert

> >
> >> This patchset provides more grace way to handle this, but
> >> full memory management includes accounting of VMA-length
> >> as well (returning ENOMEM from system call) but we've decided
> >> to start with RSS.
> >>
> >>> best,
> >>> Herbert
> >>>
> >>>> _____
> >>>> Containers mailing list
> >>>> Containers@lists.osdl.org
> >>>> https://lists.osdl.org/mailman/listinfo/containers
> >>> -
> >>> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> >>> the body of a message to majordomo@vger.kernel.org
> >>> More majordomo info at http://vger.kernel.org/majordomo-info.html
> >>> Please read the FAQ at http://www.tux.org/lkml/
> >>>
> >
_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Herbert Poetzl on Mon, 12 Mar 2007 01:00:39 GMT
View Forum Message <> Reply to Message

On Sun, Mar 11, 2007 at 04:51:11AM -0800, Andrew Morton wrote:
> > On Sun, 11 Mar 2007 15:26:41 +0300 Kirill Korotaev <dev@sw.ru> wrote:
> > Andrew Morton wrote:
> > > On Tue, 06 Mar 2007 17:55:29 +0300
> > > Pavel Emelianov <xemul@sw.ru> wrote:
> > >
> > >
> > >>+struct rss_container {
> > >>+ struct res_counter res;
> > >>+ struct list_head page_list;
> > >>+ struct container_subsys_state css;
> > >>+};
> > >>+
> > >>+struct page_container {
> > >>+ struct page *page;
> > >>+ struct rss_container *cnt;
> > >>+ struct list_head list;
> > >>+};

> > >
> > >
> > > ah. This looks good. I'll find a hunk of time to go through
> > > this work and through Paul's patches. It'd be good to get both
> > > patchsets lined up in -mm within a couple of weeks. But..
> > >
> > > We need to decide whether we want to do per-container memory
> > > limitation via these data structures, or whether we do it via
> > > a physical scan of some software zone, possibly based on Mel's
> > > patches.
> > i.e. a separate memzone for each container?
>
> Yep. Straightforward machine partitioning. An attractive thing is that
> it 100% reuses existing page reclaim, unaltered.
>
> > imho memzone approach is inconvinient for pages sharing and shares
> > accounting. it also makes memory management more strict, forbids
> > overcommiting per-container etc.
>
> umm, who said they were requirements?

well, I guess all existing OS-Level virtualizations
(Linux-VServer, OpenVZ, and FreeVPS) have stated more
than one time that _sharing_ of resources is a central
element, and one especially important resource to share
is memory (RAM) ...

if your aim is full partitioning, we do not need to
bother with OS-Level isolation, we can simply use
Paravirtualization and be done ...

> > Maybe you have some ideas how we can decide on this?
>
> We need to work out what the requirements are before we can
> settle on an implementation.

Linux-VServer (and probably OpenVZ):

- shared mappings of 'shared' files (binaries
  and libraries) to allow for reduced memory
  footprint when N identical guests are running

- virtual 'physical' limit should not cause
  swap out when there are still pages left on
  the host system (but pages of over limit guests
  can be preferred for swapping)

- accounting and limits have to be consistent

and should roughly represent the actual used
memory/swap (modulo optimizations, I can go
into detail here, if necessary)

- OOM handling on a per guest basis, i.e. some
out of memory condition in guest A must not
affect guest B

HTC,
Herbert

> Sigh. Who is running this show? Anyone?
>
> You can actually do a form of overcommittment by allowing multiple
> containers to share one or more of the zones. Whether that is
> sufficient or suitable I don't know. That depends on the requirements,
> and we haven't even discussed those, let alone agreed to them.
>
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 1/7] Resource counters
Posted by Herbert Poetzl on Mon, 12 Mar 2007 01:16:13 GMT
View Forum Message <> Reply to Message

On Sun, Mar 11, 2007 at 01:00:15PM -0600, Eric W. Biederman wrote:
> Herbert Poetzl <herbert@13thfloor.at> writes:
>
> >
> > Linux-VServer does the accounting with atomic counters,
> > so that works quite fine, just do the checks at the
> > beginning of whatever resource allocation and the
> > accounting once the resource is acquired ...
>
> Atomic operations versus locks is only a granularity thing.
> You still need the cache line which is the cost on SMP.
>
> Are you using atomic_add_return or atomic_add_unless or
> are you performing you actions in two separate steps
> which is racy? What I have seen indicates you are using
> a racy two separate operation form.

yes, this is the current implementation which
is more than sufficient, but I'm aware of the
potential issues here, and I have an experimental
patch sitting here which removes this race with
the following change:

 - doesn't store the accounted value but
   limit - accounted (i.e. the free resource)
 - uses atomic_add_return()
 - when negative, an error is returned and
   the resource amount is added back

changes to the limit have to adjust the 'current'
value too, but that is again simple and atomic

best,
Herbert

PS: atomic_add_unless() didn't exist back then
(at least I think so) but that might be an option
too ...

> >> If we'll remove failcnt this would look like
> >>   while (atomic_cmpxchg(...))
> >> which is also not that good.
> >>
> >> Moreover - in RSS accounting patches I perform page list
> >> manipulations under this lock, so this also saves one atomic op.
> >
> > it still hasn't been shown that this kind of RSS limit
> > doesn't add big time overhead to normal operations
> > (inside and outside of such a resource container)
> >
> > note that the 'usual' memory accounting is much more
> > lightweight and serves similar purposes ...
>
> Perhaps....
>
> Eric
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by xemul on Mon, 12 Mar 2007 08:31:08 GMT
View Forum Message <> Reply to Message

[snip]

>>>>>> We need to decide whether we want to do per-container memory
>>>>>> limitation via these data structures, or whether we do it via
>>>>>> a physical scan of some software zone, possibly based on Mel's
>>>>>> patches.
>>>>> why not do simple page accounting (as done currently
>>>>> in Linux) and use that for the limits, without
>>>>> keeping the reference from container to page?
>>>> As I've already answered in my previous letter simple
>>>> limiting w/o per-container reclamation and per-container
>>>> oom killer isn't a good memory management. It doesn't allow
>>>> to handle resource shortage gracefully.
>>> per container OOM killer does not require any container
>>> page reference, you know _what_ tasks belong to the
>>> container, and you know their _badness_ from the normal
>>> OOM calculations, so doing them for a container is really
>>> straight forward without having any page 'tagging'
>> That's true. If you look at the patches you'll
>> find out that no code in oom killer uses page 'tag'.
>
> so what do we keep the context -> page reference
> then at all?

We need this for
1. keeping page's owner to uncharge to IT when page
   goes away. Or do you propose to uncharge it to
   current (i.e. ANY) container like you do all across
   Vserver accounting which screws up accounting with
   pages sharing?
2. managing LRU lists for good reclamation. See Balbir's
   patches for details.
3. possible future uses - correct sharing accounting,
   dirty pages accounting, etc

>>> for the reclamation part, please elaborate how that will
>>> differ in a (shared memory) guest from what the kernel
>>> currently does ...
>> This is all described in the code and in the
>> discussions we had before.
>
> must have missed some of them, please can you
> point me to the relevant threads ...

lkml.org archives and google will help you :)

> TIA,
> Herbert

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by xemul on Mon, 12 Mar 2007 09:02:01 GMT
View Forum Message <> Reply to Message

>>> Maybe you have some ideas how we can decide on this?
>> We need to work out what the requirements are before we can
>> settle on an implementation.
>
> Linux-VServer (and probably OpenVZ):
>
> - shared mappings of 'shared' files (binaries
> and libraries) to allow for reduced memory
> footprint when N identical guests are running

This is done in current patches.

> - virtual 'physical' limit should not cause
> swap out when there are still pages left on
> the host system (but pages of over limit guests
> can be preferred for swapping)

So what to do when virtual physical limit is hit?
OOM-kill current task?

> - accounting and limits have to be consistent
> and should roughly represent the actual used
> memory/swap (modulo optimizations, I can go
> into detail here, if necessary)

This is true for current implementation for
booth - this patchset ang OpenVZ beancounters.

If you sum up the physpages values for all containers
you'll get the exact number of RAM pages used.

> - OOM handling on a per guest basis, i.e. some
> out of memory condition in guest A must not
> affect guest B

This is done in current patches.

Herbert, did you look at the patches before
sending this mail or do you just want to
'take part' in conversation w/o understanding
of hat is going on?

> HTC,
> Herbert
>
>> Sigh. Who is running this show? Anyone?
>>
>> You can actually do a form of overcommittment by allowing multiple
>> containers to share one or more of the zones. Whether that is
>> sufficient or suitable I don't know. That depends on the requirements,
>> and we haven't even discussed those, let alone agreed to them.
>>
>> _____
>> Containers mailing list
>> Containers@lists.osdl.org
>> https://lists.osdl.org/mailman/listinfo/containers
>

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Kirill Korotaev on Mon, 12 Mar 2007 09:10:22 GMT
View Forum Message <> Reply to Message

Eric,

> And misses every resource sharing opportunity in sight.

that was my point too.

> Except for
> filtering the which pages are eligible for reclaim an RSS limit should
> not need to change the existing reclaim logic, and with things like the
> memory zones we have had that kind of restriction in the reclaim logic
> for a long time. So filtering out ineligible pages isn't anything new.

exactly this is implemented in the current patches from Pavel.
the only difference is that filtering is not done in general LRU list,
which is not effective, but via per-container LRU list.

So the pointer on the page structure does 2 things:
- fast reclamation
- correct uncharging of page from where it was charged
  (e.g. shared pages can be mapped first in one container, but the last unmap
   done from another one).

>>We need to work out what the requirements are before we can settle on an
>>implementation.
>
>
> If you are talking about RSS limits the term is well defined. The
> number of pages you can have mapped into your set of address space at
> any given time.
>
> Unless I'm totally blind that isn't what the patchset implements.

Ouch, what makes you think so?
The fact that a page mapped into 2 different processes is charged only once?
Imho it is much more correct then sum of process' RSS within container, due to:
1. it is clear how much container uses physical pages, not abstract items
2. shared pages are charged only once, so the sum of containers RSS is still
   about physical RAM.

> A
> true RSS limit over multiple processes has a lot of potential to be
> generally useful, is very understandable, doesn't affect kernel cache
> decisions so largely performance should not be affected. There is a
> little more overhead in the fault logic but that is a moderately
> expensive path anyway.

100% agree here.

>>You can actually do a form of overcommittment by allowing multiple
>>containers to share one or more of the zones. Whether that is sufficient
>>or suitable I don't know. That depends on the requirements, and we haven't
>>even discussed those, let alone agreed to them.
>
>
> Another really nasty issue is the container term as the resource guys
> are using the term in a subtlety different way then it has been used
> with namespaces leading to several threads where the participants talked
> past each other. We need a different term to designate the group of
> tasks a resource controller is dealing with.
taskgrp? resgrp?

> The whole filesystem interface also is over general and makes it too
> easy to express the hard things (like move an existing task from one
> group of tasks to another) leading to code complications.

the things which are not supported are easy to disable.

> On the up side I think the code the focus is likely in the right place
> to start delivering usable code.

Thanks,
Kirill

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Balbir Singh on Mon, 12 Mar 2007 09:55:07 GMT
View Forum Message <> Reply to Message

> doesn't look so good for me, mainly becaus of the
> additional per page data and per page processing
>
> on 4GB memory, with 100 guests, 50% shared for each
> guest, this basically means ~1mio pages, 500k shared
> and 1500k x sizeof(page_container) entries, which
> roughly boils down to ~25MB of wasted memory ...
>
> increase the amount of shared pages and it starts
> getting worse, but maybe I'm missing something here
>
> > We need to decide whether we want to do per-container memory
> > limitation via these data structures, or whether we do it via a
> > physical scan of some software zone, possibly based on Mel's patches.
>
> why not do simple page accounting (as done currently
> in Linux) and use that for the limits, without
> keeping the reference from container to page?
>
> best,
> Herbert
>

Herbert,

You lost me in the cc list and I almost missed this part of the
thread. Could you please not modify the "cc" list.

Thanks,
Balbir

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 3/7] Data structures changes for RSS accounting
Posted by dev on Mon, 12 Mar 2007 16:16:59 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Pavel Emelianov <xemul@sw.ru> writes:
>
>
>>Adds needed pointers to mm_struct and page struct,
>>places hooks to core code for mm_struct initialization
>>and hooks in container_init_early() to preinitialize
>>RSS accounting subsystem.
>
>
> An extra pointer in struct page is unlikely to fly.
> Both because it increases the size of a size critical structure,
> and because conceptually it is ridiculous.
as it was discussed multiple times (and according OLS):
- it is not critical nowdays to expand struct page a bit in case
  accounting is on.
- it can be done w/o extending, e.g. via mapping page <-> container
  using hash or some other data structure.
  i.e. we can optimize it on size if considered needed.

> If you are limiting the RSS size you are counting the number of pages in
> the page tables. You don't care about the page itself.
>
> With the rmap code it is relatively straight forward to see if this is
> the first time a page has been added to a page table in your rss
> group, or if this is the last reference to a particular page in your
> rss group. The counters should only increment the first time a
> particular page is added to your rss group. The counters should only
> decrement when it is the last reference in your rss subsystem.
You are fundamentally wrong if shared pages are concerned.
Imagine a glibc page shared between 2 containers - VE1 and VE2.
VE1 was the first who mapped it, so it is accounted to VE1
(rmap count was increased by it).
now VE2 maps the same page. You can't determine whether this page is mapped
to this container or another one w/o page->container pointer.
All the choices you have are:
a) do not account this page, since it is allready accounted to some other VE.
b) account this page again to current container.

(a) is bad, since VE1 can unmap this page first, and the last user will be VE2.
Which means VE1 will be charged for it, while VE2 uncharged. Accounting screws up.

b) is bad, since:
  - the same page is accounted multiple times, which makes impossible
    to understand how much real memory pages container needs/consumes

- and because on container enter the process and it's pages
  are essentially moved to another context, while accounting
  can not be fixed up easily and we essentially have (a).

> This allow important little cases like glibc to be properly accounted
> for. One of the key features of a rss limit is that the kernel can
> still keep pages that you need in-core, that are accessible with just
> a minor fault. Directly owning pages works directly against that
> principle.
Sorry, can't understand what you mean. It doesn't work against.
Each container has it's own LRU. So if glibc has the most
often used pages - it won't be thrashed out.

Thanks,
Kirill

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by dev on Mon, 12 Mar 2007 16:23:25 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Pavel Emelianov <xemul@sw.ru> writes:
>
>
>>Pages are charged to their first touchers which are
>>determined using pages' mapcount manipulations in
>>rmap calls.
>
>
> NAK pages should be charged to every rss group whose mm_struct they
> are mapped into.
For these you essentially need per-container page->_mapcount counter,
otherwise you can't detect whether rss group still has the page in question being mapped
in its processes' address spaces or not.

1. This was discussed before and considered to be ok by all the resource management
   involved people.
2. this can be done with a-la page beancounters which are used in OVZ for shared
   fractions accounting. It's a next step forward.

If you know how to get "pages should be charged to every rss group whose mm_struct they
are mapped into" w/o additional pointer in struct page, please throw me an idea.

Thanks,
Kirill

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 3/7] Data structures changes for RSS accounting
Posted by Dave Hansen on Mon, 12 Mar 2007 16:48:04 GMT
View Forum Message <> Reply to Message

On Mon, 2007-03-12 at 19:16 +0300, Kirill Korotaev wrote:
> now VE2 maps the same page. You can't determine whether this page is mapped
> to this container or another one w/o page->container pointer.

Hi Kirill,

I thought we can always get from the page to the VMA. rmap provides
this to us via page->mapping and the 'struct address_space' or anon_vma.
Do we agree on that?

We can also get from the vma to the mm very easily, via vma->vm_mm,
right?

We can also get from a task to the container quite easily.

So, the only question becomes whether there is a 1:1 relationship
between mm_structs and containers. Does each mm_struct belong to one
and only one container? Basically, can a threaded process have
different threads in different containers?

It seems that we could bridge the gap pretty easily by either assigning
each mm_struct to a container directly, or putting some kind of
task-to-mm lookup. Perhaps just a list like
mm->tasks_using_this_mm_list.

Not rocket science, right?

-- Dave

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by Dave Hansen on Mon, 12 Mar 2007 16:50:08 GMT
View Forum Message <> Reply to Message

On Mon, 2007-03-12 at 19:23 +0300, Kirill Korotaev wrote:
>
> For these you essentially need per-container page->_mapcount counter,
> otherwise you can't detect whether rss group still has the page in question being mapped
> in its processes' address spaces or not.

What do you mean by this? You can always tell whether a process has a
particular page mapped. Could you explain the issue a bit more. I'm
not sure I get it.

-- Dave

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by dev on Mon, 12 Mar 2007 17:07:42 GMT
View Forum Message <> Reply to Message

> On Mon, 2007-03-12 at 19:23 +0300, Kirill Korotaev wrote:
>
>>For these you essentially need per-container page->_mapcount counter,
>>otherwise you can't detect whether rss group still has the page in question being mapped
>>in its processes' address spaces or not.
>
>
> What do you mean by this? You can always tell whether a process has a
> particular page mapped. Could you explain the issue a bit more. I'm
> not sure I get it.
When we do charge/uncharge we have to answer on another question:
"whether *any* task from the *container* has this page mapped", not the
"whether *this* task has this page mapped".

Thanks,
Kirill

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 3/7] Data structures changes for RSS accounting
Posted by xemul on Mon, 12 Mar 2007 17:19:13 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Mon, 2007-03-12 at 19:16 +0300, Kirill Korotaev wrote:
>> now VE2 maps the same page. You can't determine whether this page is mapped
>> to this container or another one w/o page->container pointer.
>
> Hi Kirill,
>
> I thought we can always get from the page to the VMA. rmap provides
> this to us via page->mapping and the 'struct address_space' or anon_vma.
> Do we agree on that?

Not completely. When page is unmapped from the *very last*
user its *first* toucher may already be dead. So we'll never
find out who it was.

> We can also get from the vma to the mm very easily, via vma->vm_mm,
> right?
>
> We can also get from a task to the container quite easily.
>
> So, the only question becomes whether there is a 1:1 relationship
> between mm_structs and containers. Does each mm_struct belong to one

No. The question is "how to get a container that touched the
page first" which is the same as "how to find mm_struct which
touched the page first". Obviously there's no answer on this
question unless we hold some direct page->container reference.
This may be a hash, a direct on-page pointer, or mirrored
array of pointers.

> and only one container? Basically, can a threaded process have
> different threads in different containers?
>
> It seems that we could bridge the gap pretty easily by either assigning
> each mm_struct to a container directly, or putting some kind of
> task-to-mm lookup. Perhaps just a list like
> mm->tasks_using_this_mm_list.

This could work for reclamation: we scan through all the
mm_struct-s within the container and shrink its' pages, but
we can't make LRU this way.

> Not rocket science, right?
>
> -- Dave

>
> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/
>

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 3/7] Data structures changes for RSS accounting
Posted by Balbir Singh on Mon, 12 Mar 2007 17:21:30 GMT
View Forum Message <> Reply to Message

On 3/12/07, Dave Hansen <hansendc@us.ibm.com> wrote:
> On Mon, 2007-03-12 at 19:16 +0300, Kirill Korotaev wrote:
> > now VE2 maps the same page. You can't determine whether this page is mapped
> > to this container or another one w/o page->container pointer.
>
> Hi Kirill,
>
> I thought we can always get from the page to the VMA. rmap provides
> this to us via page->mapping and the 'struct address_space' or anon_vma.
> Do we agree on that?
>
> We can also get from the vma to the mm very easily, via vma->vm_mm,
> right?
>
> We can also get from a task to the container quite easily.
>
> So, the only question becomes whether there is a 1:1 relationship
> between mm_structs and containers. Does each mm_struct belong to one
> and only one container? Basically, can a threaded process have
> different threads in different containers?
>
> It seems that we could bridge the gap pretty easily by either assigning
> each mm_struct to a container directly, or putting some kind of
> task-to-mm lookup. Perhaps just a list like
> mm->tasks_using_this_mm_list.
>
> Not rocket science, right?
>
> -- Dave
>

These patches are very similar to what I posted at
http://lwn.net/Articles/223829/
In my patches, the thread group leader owns the mm_struct and all
threads belong to the same container. I did not have a per container
LRU, walking the global list for reclaim was a bit slow, but otherwise
my patches did not add anything to struct page

I used rmap information to get to the VMA and then the mm_struct.
Kirill, it is possible to determine all the containers that map the
page. Please see the page_in_container() function of
http://lkml.org/lkml/2007/2/26/7.

I was also thinking of using the page table(s) to identify all pages
belonging to a container, by obtaining all the mm_structs of tasks
belonging to a container. But this approach would not work well for
the page cache controller, when we add that to our memory controller.

Balbir

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 3/7] Data structures changes for RSS accounting
Posted by Dave Hansen on Mon, 12 Mar 2007 17:27:51 GMT
View Forum Message <> Reply to Message

On Mon, 2007-03-12 at 20:19 +0300, Pavel Emelianov wrote:
> Dave Hansen wrote:
> > On Mon, 2007-03-12 at 19:16 +0300, Kirill Korotaev wrote:
> >> now VE2 maps the same page. You can't determine whether this page is mapped
> >> to this container or another one w/o page->container pointer.
> >
> > Hi Kirill,
> >
> > I thought we can always get from the page to the VMA. rmap provides
> > this to us via page->mapping and the 'struct address_space' or anon_vma.
> > Do we agree on that?
>
> Not completely. When page is unmapped from the *very last*
> user its *first* toucher may already be dead. So we'll never
> find out who it was.

OK, but this is assuming that we didn't *un*account for the page when
the last user of the "owning" container stopped using the page.

> > We can also get from the vma to the mm very easily, via vma->vm_mm,

> > right?
> >
> > We can also get from a task to the container quite easily.
> >
> > So, the only question becomes whether there is a 1:1 relationship
> > between mm_structs and containers. Does each mm_struct belong to one
>
> No. The question is "how to get a container that touched the
> page first" which is the same as "how to find mm_struct which
> touched the page first". Obviously there's no answer on this
> question unless we hold some direct page->container reference.
> This may be a hash, a direct on-page pointer, or mirrored
> array of pointers.

Or, you keep track of when the last user from the container goes away,
and you effectively account it to another one.

Are there problems with shifting ownership around like this?

-- Dave

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by Dave Hansen on Mon, 12 Mar 2007 17:33:16 GMT
View Forum Message <> Reply to Message

On Mon, 2007-03-12 at 20:07 +0300, Kirill Korotaev wrote:
> > On Mon, 2007-03-12 at 19:23 +0300, Kirill Korotaev wrote:
> >>For these you essentially need per-container page->_mapcount counter,
> >>otherwise you can't detect whether rss group still has the page in question being mapped
> >>in its processes' address spaces or not.
> >
> > What do you mean by this? You can always tell whether a process has a
> > particular page mapped. Could you explain the issue a bit more. I'm
> > not sure I get it.
> When we do charge/uncharge we have to answer on another question:
> "whether *any* task from the *container* has this page mapped", not the
> "whether *this* task has this page mapped".

That's a bit more clear. ;)

OK, just so I make sure I'm getting your argument here. It would be too
expensive to go looking through all of the rmap data for _any_ other

task that might be sharing the charge (in the same container) with the current task that is doing the unmapping.

The requirements you're presenting so far appear to be:

1. The first user of a page in a container must be charged
2. The second user of a page in a container must not be charged
3. A container using a page must take a diminished charge when another container is already using the page.
4. Additional fields in data structures (including 'struct page') are permitted

What have I missed? What are your requirements for performance?

I'm not quite sure how the page->container stuff fits in here, though. page->container would appear to be strictly assigning one page to one container, but I know that beancounters can do partial page charges. Care to fill me in?

-- Dave

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Dave Hansen on Mon, 12 Mar 2007 18:42:59 GMT
View Forum Message <> Reply to Message

How about we drill down on these a bit more.

On Mon, 2007-03-12 at 02:00 +0100, Herbert Poetzl wrote:
> - shared mappings of 'shared' files (binaries
> and libraries) to allow for reduced memory
> footprint when N identical guests are running

So, it sounds like this can be phrased as a requirement like:

"Guests must be able to share pages."

Can you give us an idea why this is so? On a typical vserver system, how much memory would be lost if guests were not permitted to share pages like this? How much does this decrease the density of vservers?

> - virtual 'physical' limit should not cause
> swap out when there are still pages left on

> the host system (but pages of over limit guests
> can be preferred for swapping)

Is this a really hard requirement? It seems a bit fluffy to me. An added bonus if we can do it, but certainly not the most important requirement in the bunch.

What are the consequences if this isn't done? Doesn't a loaded system eventually have all of its pages used anyway, so won't this always be a temporary situation?

This also seems potentially harmful if we aren't able to get pages *back* that we've given to a guest. Tasks can pin pages in lots of creative ways.

> - accounting and limits have to be consistent
> and should roughly represent the actual used
> memory/swap (modulo optimizations, I can go
> into detail here, if necessary)

So, consistency is important, but is precision? If we, for instance, used one of the hashing schemes, we could have some imprecise decisions made but the system would stay consistent overall.

This requirement also doesn't seem to push us in the direction of having distinct page owners, or some sharing mechanism, because both would be consistent.

> - OOM handling on a per guest basis, i.e. some
> out of memory condition in guest A must not
> affect guest B

I'll agree that this one is important and well stated as-is. Any disagreement on this one?

-- Dave

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Herbert Poetzl on Mon, 12 Mar 2007 21:11:11 GMT
View Forum Message <> Reply to Message

On Mon, Mar 12, 2007 at 12:02:01PM +0300, Pavel Emelianov wrote:

> >>> Maybe you have some ideas how we can decide on this?
> >> We need to work out what the requirements are before we can
> >> settle on an implementation.
> >
> > Linux-VServer (and probably OpenVZ):
> >
> > - shared mappings of 'shared' files (binaries
> >   and libraries) to allow for reduced memory
> >   footprint when N identical guests are running
>
> This is done in current patches.

nice, but the question was about _requirements_
(so your requirements are?)

> > - virtual 'physical' limit should not cause
> >   swap out when there are still pages left on
> >   the host system (but pages of over limit guests
> >   can be preferred for swapping)
>
> So what to do when virtual physical limit is hit?
> OOM-kill current task?

when the RSS limit is hit, but there _are_ enough
pages left on the physical system, there is no
good reason to swap out the page at all

 - there is no benefit in doing so (performance
   wise, that is)

 - it actually hurts performance, and could
   become a separate source for DoS

what should happen instead (in an ideal world :)
is that the page is considered swapped out for
the guest (add guest penality for swapout), and
when the page would be swapped in again, the guest
takes a penalty (for the 'virtual' page in) and
the page is returned to the guest, possibly kicking
out (again virtually) a different page

> > - accounting and limits have to be consistent
> >   and should roughly represent the actual used
> >   memory/swap (modulo optimizations, I can go
> >   into detail here, if necessary)
>
> This is true for current implementation for
> booth - this patchset ang OpenVZ beancounters.

>
> If you sum up the physpages values for all containers
> you'll get the exact number of RAM pages used.

hmm, including or excluding the host pages?

> > - OOM handling on a per guest basis, i.e. some
> > out of memory condition in guest A must not
> > affect guest B
>
> This is done in current patches.

> Herbert, did you look at the patches before
> sending this mail or do you just want to
> 'take part' in conversation w/o understanding
> of hat is going on?

again, the question was about requirements, not
your patches, and yes, I had a look at them _and_
the OpenVZ implementations ...

best,
Herbert

PS: hat is going on? :)

> > HTC,
> > Herbert
> >
> >> Sigh. Who is running this show? Anyone?
> >>
> >> You can actually do a form of overcommittment by allowing multiple
> >> containers to share one or more of the zones. Whether that is
> >> sufficient or suitable I don't know. That depends on the requirements,
> >> and we haven't even discussed those, let alone agreed to them.
> >>
> >> _____
> >> Containers mailing list
> >> Containers@lists.osdl.org
> >> https://lists.osdl.org/mailman/listinfo/containers
> >

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Herbert Poetzl on Mon, 12 Mar 2007 22:41:29 GMT
View Forum Message <> Reply to Message

On Mon, Mar 12, 2007 at 11:42:59AM -0700, Dave Hansen wrote:
> How about we drill down on these a bit more.
>
> On Mon, 2007-03-12 at 02:00 +0100, Herbert Poetzl wrote:
> > - shared mappings of 'shared' files (binaries
> > and libraries) to allow for reduced memory
> > footprint when N identical guests are running
>
> So, it sounds like this can be phrased as a requirement like:
>
> "Guests must be able to share pages."
>
> Can you give us an idea why this is so?

sure, one reason for this is that guests tend to
be similar (or almost identical) which results
in quite a lot of 'shared' libraries and executables
which would otherwise get cached for each guest and
would also be mapped for each guest separately

> On a typical vserver system,

there is nothing like a typical Linux-VServer system :)

> how much memory would be lost if guests were not permitted
> to share pages like this?

let me give a real world example here:

- typical guest with 600MB disk space
- about 100MB guest specific data (not shared)
- assumed that 80% of the libs/tools are used

gives 400MB of shared read only data

assumed you are running 100 guests on a host,
that makes ~39GB of virtual memory which will
get paged in and out over and over again ...

.. compared to 400MB shared pages in memory :)

> How much does this decrease the density of vservers?

well, let's look at the overall memory resource
function with the above assumptions:

with sharing: f(N) = N*80M + 400M
without sharing: g(N) = N*480M

so the decrease N->inf: g/f -> 6 (factor)

which is quite realistic, if you consider that
there are only so many distributions, OTOH, the
factor might become less important when the
guest specific data grows ...

> > - virtual 'physical' limit should not cause
> >   swap out when there are still pages left on
> >   the host system (but pages of over limit guests
> >   can be preferred for swapping)
>
> Is this a really hard requirement?

no, not hard, but a reasonable optimization ...

let me note once again, that for full isolation
you better go with Xen or some other Hypervisor
because if you make it work like Xen, it will
become as slow and resource hungry as any other
paravirtualization solution ...

> It seems a bit fluffy to me.

most optimizations might look strange at first
glance, but when you check what the limitting
factors for OS-Level virtualizations are, you
will find that it looks like this:

(in order of decreasing relevance)

- I/O subsystem
- available memory
- network performance
- CPU performance

note: this is for 'typical' guests, not for
number crunching or special database, or pure
network bound applications/guests ...

> An added bonus if we can do it, but certainly not the
> most important requirement in the bunch.

nope, not the _most_ important one, but it

all summs up :)

> What are the consequences if this isn't done? Doesn't
> a loaded system eventually have all of its pages used
> anyway, so won't this always be a temporary situation?

let's consider a quite limited guest (or several
of them) which have a 'RAM' limit of 64MB and
additional 64MB of 'virtual swap' assigned ...

if they use roughly 96MB (memory footprint) then
having this 'fluffy' optimization will keep them
running without any effect on the host side, but
without, they will continously swap in and out
which will affect not only the host, but also the
other guests ...

> This also seems potentially harmful if we aren't able
> to get pages *back* that we've given to a guest.

no, the idea is not to keep them unconditionally,
the concept is to allow them to stay, even if the
guest has reached the RSS limit and a 'real' system
would have to swap pages out (or simply drop them)
to get other pages mapped ...

> Tasks can pin pages in lots of creative ways.

sure, this is why we should have proper limits
for that too :)

> > - accounting and limits have to be consistent
> >   and should roughly represent the actual used
> >   memory/swap (modulo optimizations, I can go
> >   into detail here, if necessary)
>
> So, consistency is important, but is precision?

IMHO precision is not that important, of course,
the values should be in the same ballpark ...

> If we, for instance, used one of the hashing schemes,
> we could have some imprecise decisions made but the
> system would stay consistent overall.

it is also important that the lack of precision
cannot be exploited to allocate unreasonable
ammounts of resources ...

at least Linux-VServer could live with +/- 10%
(or probably more) as I said, it is mainly used
for preventing DoS or DoR attacks ...

> This requirement also doesn't seem to push us in the
> direction of having distinct page owners, or some
> sharing mechanism, because both would be consistent.

> > - OOM handling on a per guest basis, i.e. some
> >   out of memory condition in guest A must not
> >   affect guest B
>
> I'll agree that this one is important and well stated
> as-is. Any disagreement on this one?

nope ...

best,
Herbert

> -- Dave

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Dave Hansen on Mon, 12 Mar 2007 23:02:08 GMT
View Forum Message <> Reply to Message

On Mon, 2007-03-12 at 23:41 +0100, Herbert Poetzl wrote:
> On Mon, Mar 12, 2007 at 11:42:59AM -0700, Dave Hansen wrote:
> > How about we drill down on these a bit more.
> >
> > On Mon, 2007-03-12 at 02:00 +0100, Herbert Poetzl wrote:
> > > - shared mappings of 'shared' files (binaries
> > >   and libraries) to allow for reduced memory
> > >   footprint when N identical guests are running
> >
> > So, it sounds like this can be phrased as a requirement like:
> >
> > "Guests must be able to share pages."
> >
> > Can you give us an idea why this is so?
>
> sure, one reason for this is that guests tend to

> be similar (or almost identical) which results
> in quite a lot of 'shared' libraries and executables
> which would otherwise get cached for each guest and
> would also be mapped for each guest separately
>
> > On a typical vserver system,
>
> there is nothing like a typical Linux-VServer system :)
>
> > how much memory would be lost if guests were not permitted
> > to share pages like this?
>
> let me give a real world example here:
>
> - typical guest with 600MB disk space
> - about 100MB guest specific data (not shared)
> - assumed that 80% of the libs/tools are used

I get the general idea here, but I just don't think those numbers are very accurate. My laptop has a bunch of gunk open (xterm, evolution, firefox, xchat, etc...). I ran this command:

lsof | egrep '/(usr/|lib.*\.so)' | awk '{print $9}' | sort | uniq | xargs du -Dcs

and got:

113840 total

On a web/database server that I have (ps aux | wc -l == 128), I just ran the same:

39168   total

That's assuming that all of the libraries are fully read in and populated, just by their on-disk sizes. Is that not a reasonable measure of the kinds of things that we can expect to be shared in a vserver? If so, it's a long way from 400MB.

Could you try a similar measurement on some of your machines? Perhaps mine are just weird.

> > > - virtual 'physical' limit should not cause
> > >   swap out when there are still pages left on
> > >   the host system (but pages of over limit guests
> > >   can be preferred for swapping)
> >
> > Is this a really hard requirement?
>

> no, not hard, but a reasonable optimization ...
>
> let me note once again, that for full isolation
> you better go with Xen or some other Hypervisor
> because if you make it work like Xen, it will
> become as slow and resource hungry as any other
> paravirtualization solution ...

Believe me, _I_ don't want Xen. :)

> > It seems a bit fluffy to me.
>
> most optimizations might look strange at first
> glance, but when you check what the limitting
> factors for OS-Level virtualizations are, you
> will find that it looks like this:
>
> (in order of decreasing relevance)
>
> - I/O subsystem
> - available memory
> - network performance
> - CPU performance
>
> note: this is for 'typical' guests, not for
> number crunching or special database, or pure
> network bound applications/guests ...

I don't doubt this, but doing this two-level page-out thing for
containers/vservers over their limits is surely something that we should
consider farther down the road, right?

It's important to you, but you're obviously not doing any of the
mainline coding, right?

> > What are the consequences if this isn't done? Doesn't
> > a loaded system eventually have all of its pages used
> > anyway, so won't this always be a temporary situation?
>
> let's consider a quite limited guest (or several
> of them) which have a 'RAM' limit of 64MB and
> additional 64MB of 'virtual swap' assigned ...
>
> if they use roughly 96MB (memory footprint) then
> having this 'fluffy' optimization will keep them
> running without any effect on the host side, but
> without, they will continously swap in and out
> which will affect not only the host, but also the

> other guests ...

All workloads that use $limit+1 pages of memory will always pay the
price, right? :)

-- Dave

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Herbert Poetzl on Mon, 12 Mar 2007 23:43:35 GMT
View Forum Message <> Reply to Message

On Mon, Mar 12, 2007 at 03:25:07PM +0530, Balbir Singh wrote:
> > doesn't look so good for me, mainly becaus of the
> > additional per page data and per page processing
> >
> > on 4GB memory, with 100 guests, 50% shared for each
> > guest, this basically means ~1mio pages, 500k shared
> > and 1500k x sizeof(page_container) entries, which
> > roughly boils down to ~25MB of wasted memory ...
> >
> > increase the amount of shared pages and it starts
> > getting worse, but maybe I'm missing something here
> >
> > > We need to decide whether we want to do per-container memory
> > > limitation via these data structures, or whether we do it via
> > > a physical scan of some software zone, possibly based on Mel's
> > > patches.
> >
> > why not do simple page accounting (as done currently
> > in Linux) and use that for the limits, without
> > keeping the reference from container to page?
> >
> > best,
> > Herbert
> >
>
> Herbert,
>
> You lost me in the cc list and I almost missed this part of the
> thread.

hmm, it is very unlikely that this would happen,

for several reasons ... and indeed, checking the
thread in my mailbox shows that akpm dropped you ...

Subject: [RFC][PATCH 2/7] RSS controller core
From: Pavel Emelianov <xemul@sw.ru>
To: Andrew Morton <akpm@osdl.org>, Paul Menage <menage@google.com>,
Srivatsa Vaddagiri <vatsa@in.ibm.com>,
Balbir Singh <balbir@in.ibm.com>
Cc: containers@lists.osdl.org,
Linux Kernel Mailing List <linux-kernel@vger.kernel.org>
Date: Tue, 06 Mar 2007 17:55:29 +0300

Subject: Re: [RFC][PATCH 2/7] RSS controller core
From: Andrew Morton <akpm@linux-foundation.org>
To: Pavel Emelianov <xemul@sw.ru>
Cc: Kirill@smtp.osdl.org, Linux@smtp.osdl.org, containers@lists.osdl.org,
Paul Menage <menage@google.com>,
List <linux-kernel@vger.kernel.org>
Date: Tue, 6 Mar 2007 14:00:36 -0800
---
that's the one I 'group' replied to ...

> Could you please not modify the "cc" list.

I never modify the cc unless explicitely asked
to do so. I wish others would have it that way
too :)

best,
Herbert

> Thanks,
> Balbir
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by Herbert Poetzl on Mon, 12 Mar 2007 23:54:52 GMT
View Forum Message <> Reply to Message

On Mon, Mar 12, 2007 at 09:50:08AM -0700, Dave Hansen wrote:
> On Mon, 2007-03-12 at 19:23 +0300, Kirill Korotaev wrote:
> >
> > For these you essentially need per-container page->_mapcount counter,
> > otherwise you can't detect whether rss group still has the page
> > in question being mapped in its processes' address spaces or not.

> What do you mean by this? You can always tell whether a process has a
> particular page mapped. Could you explain the issue a bit more. I'm
> not sure I get it.

OpenVZ wants to account _shared_ pages in a guest
different than separate pages, so that the RSS
accounted values reflect the actual used RAM instead
of the sum of all processes RSS' pages, which for
sure is more relevant to the administrator, but IMHO
not so terribly important to justify memory consuming
structures and sacrifice performance to get it right

YMMV, but maybe we can find a smart solution to the
issue too :)

best,
Herbert

> -- Dave
>
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Balbir Singh on Tue, 13 Mar 2007 01:57:06 GMT
View Forum Message <> Reply to Message

> hmm, it is very unlikely that this would happen,
> for several reasons ... and indeed, checking the
> thread in my mailbox shows that akpm dropped you ...
>

But, I got Andrew's email.

> ---
> Subject: [RFC][PATCH 2/7] RSS controller core
> From: Pavel Emelianov <xemul@sw.ru>
> To: Andrew Morton <akpm@osdl.org>, Paul Menage <menage@google.com>,
> Srivatsa Vaddagiri <vatsa@in.ibm.com>,
> Balbir Singh <balbir@in.ibm.com>
> Cc: containers@lists.osdl.org,
> Linux Kernel Mailing List <linux-kernel@vger.kernel.org>
> Date: Tue, 06 Mar 2007 17:55:29 +0300
> ---
> Subject: Re: [RFC][PATCH 2/7] RSS controller core
> From: Andrew Morton <akpm@linux-foundation.org>
> To: Pavel Emelianov <xemul@sw.ru>
> Cc: Kirill@smtp.osdl.org, Linux@smtp.osdl.org, containers@lists.osdl.org,
> Paul Menage <menage@google.com>,
> List <linux-kernel@vger.kernel.org>
> Date: Tue, 6 Mar 2007 14:00:36 -0800
> ---
> that's the one I 'group' replied to ...
>
> > Could you please not modify the "cc" list.
>
> I never modify the cc unless explicitely asked
> to do so. I wish others would have it that way
> too :)
>

Thats good to know, but my mailer shows

Andrew Morton <akpm@linux-foundation.org>
 to Pavel Emelianov <xemul@sw.ru>
 cc
 Paul Menage <menage@google.com>,
Srivatsa Vaddagiri <vatsa@in.ibm.com>,
Balbir Singh <balbir@in.ibm.com> (see I am <<HERE>>),
devel@openvz.org,
Linux Kernel Mailing List <linux-kernel@vger.kernel.org>,
containers@lists.osdl.org,
Kirill Korotaev <dev@sw.ru>
 date Mar 7, 2007 3:30 AM
 subject Re: [RFC][PATCH 2/7] RSS controller core
 mailed-by vger.kernel.org
On Tue, 06 Mar 2007 17:55:29 +0300

and your reply as

Andrew Morton <akpm@linux-foundation.org>,

Pavel Emelianov <xemul@sw.ru>,
Kirill@smtp.osdl.org,
Linux@smtp.osdl.org,
containers@lists.osdl.org,
Paul Menage <menage@google.com>,
List <linux-kernel@vger.kernel.org>
 to Andrew Morton <akpm@linux-foundation.org>
 cc
 Pavel Emelianov <xemul@sw.ru>,
Kirill@smtp.osdl.org,
Linux@smtp.osdl.org,
containers@lists.osdl.org,
Paul Menage <menage@google.com>,
List <linux-kernel@vger.kernel.org>
 date Mar 9, 2007 10:18 PM
 subject Re: [RFC][PATCH 2/7] RSS controller core
 mailed-by vger.kernel.org

I am not sure what went wrong. Could you please check your mail
client, cause it seemed to even change email address to smtp.osdl.org
which bounced back when I wrote to you earlier.

> best,
> Herbert
>

Cheers,
Balbir

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Srivatsa Vaddagiri on Tue, 13 Mar 2007 02:24:29 GMT
View Forum Message <> Reply to Message

On Tue, Mar 13, 2007 at 07:27:06AM +0530, Balbir Singh wrote:
> I am not sure what went wrong. Could you please check your mail
> client, cause it seemed to even change email address to smtp.osdl.org
> which bounced back when I wrote to you earlier.

I have a problem doing a group-reply in mutt to Herbert's mails. His
email id gets dropped from the To or Cc list. Is that his email setting?
Don't know.

--

Regards,
vatsa

_____

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by akpm on Tue, 13 Mar 2007 06:04:39 GMT
View Forum Message <> Reply to Message

> On Mon, 12 Mar 2007 23:41:29 +0100 Herbert Poetzl <herbert@13thfloor.at> wrote:
> On Mon, Mar 12, 2007 at 11:42:59AM -0700, Dave Hansen wrote:
> > How about we drill down on these a bit more.
> >
> > On Mon, 2007-03-12 at 02:00 +0100, Herbert Poetzl wrote:
> > > - shared mappings of 'shared' files (binaries
> > >   and libraries) to allow for reduced memory
> > >   footprint when N identical guests are running
> >
> > So, it sounds like this can be phrased as a requirement like:
> >
> > "Guests must be able to share pages."
> >
> > Can you give us an idea why this is so?
>
> sure, one reason for this is that guests tend to
> be similar (or almost identical) which results
> in quite a lot of 'shared' libraries and executables
> which would otherwise get cached for each guest and
> would also be mapped for each guest separately

noooooooo. What you're saying there amounts to text replication. There is no proposal here to create duplicated copies of pagecache pages: the VM just doesn't support that (Nick has soe protopatches which do this as a possible NUMA optimisation).

So these mmapped pages will contiue to be shared across all guests. The problem boils down to "which guest(s) get charged for each shared page".

A simple and obvious and easy-to-implement answer is "the guest which paged it in". I think we should firstly explain why that is insufficient.

_____

Subject: Re: [RFC][PATCH 3/7] Data structures changes for RSS accounting
Posted by xemul on Tue, 13 Mar 2007 07:10:43 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Mon, 2007-03-12 at 20:19 +0300, Pavel Emelianov wrote:
>> Dave Hansen wrote:
>>> On Mon, 2007-03-12 at 19:16 +0300, Kirill Korotaev wrote:
>>>> now VE2 maps the same page. You can't determine whether this page is mapped
>>>> to this container or another one w/o page->container pointer.
>>> Hi Kirill,
>>>
>>> I thought we can always get from the page to the VMA. rmap provides
>>> this to us via page->mapping and the 'struct address_space' or anon_vma.
>>> Do we agree on that?
>> Not completely. When page is unmapped from the *very last*
>> user its *first* toucher may already be dead. So we'll never
>> find out who it was.
>
> OK, but this is assuming that we didn't *un*account for the page when
> the last user of the "owning" container stopped using the page.

That's exactly what we agreed on during our discussions:
When page is get touched it is charged to this container.
When page is get touched again by new container it is NOT
charged to new container, but keeps holding the old one
till it (the page) is completely freed. Nobody worried the
fact that a single page can hold container for good.

OpenVZ beancounters work the other way (and we proposed this
solution when we first sent the patches). We keep track of
all the containers (i.e. beancounters) holding this page.

>>> We can also get from the vma to the mm very easily, via vma->vm_mm,
>>> right?
>>>
>>> We can also get from a task to the container quite easily.
>>>
>>> So, the only question becomes whether there is a 1:1 relationship
>>> between mm_structs and containers. Does each mm_struct belong to one
>> No. The question is "how to get a container that touched the
>> page first" which is the same as "how to find mm_struct which
>> touched the page first". Obviously there's no answer on this
>> question unless we hold some direct page->container reference.
>> This may be a hash, a direct on-page pointer, or mirrored
>> array of pointers.
>
> Or, you keep track of when the last user from the container goes away,
> and you effectively account it to another one.

We can migrate page to another user but we decided
to implement it later after accepting simple accounting.

> Are there problems with shifting ownership around like this?
>
> -- Dave
>
>

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by xemul on Tue, 13 Mar 2007 07:17:54 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Mon, Mar 12, 2007 at 12:02:01PM +0300, Pavel Emelianov wrote:
>>>>> Maybe you have some ideas how we can decide on this?
>>>> We need to work out what the requirements are before we can
>>>> settle on an implementation.
>>> Linux-VServer (and probably OpenVZ):
>>>
>>> - shared mappings of 'shared' files (binaries
>>>   and libraries) to allow for reduced memory
>>>   footprint when N identical guests are running
>> This is done in current patches.
>
> nice, but the question was about _requirements_
> (so your requirements are?)
>
>>> - virtual 'physical' limit should not cause
>>>   swap out when there are still pages left on
>>>   the host system (but pages of over limit guests
>>>   can be preferred for swapping)
>> So what to do when virtual physical limit is hit?
>> OOM-kill current task?
>
> when the RSS limit is hit, but there _are_ enough
> pages left on the physical system, there is no
> good reason to swap out the page at all
>
> - there is no benefit in doing so (performance
>   wise, that is)

>
> - it actually hurts performance, and could
>   become a separate source for DoS
>
> what should happen instead (in an ideal world :)
> is that the page is considered swapped out for
> the guest (add guest penality for swapout), and

Is the page stays mapped for the container or not?
If yes then what's the use of limits? Container mapped
pages more than the limit is but all the pages are
still in memory. Sounds weird.

> when the page would be swapped in again, the guest
> takes a penalty (for the 'virtual' page in) and
> the page is returned to the guest, possibly kicking
> out (again virtually) a different page
>
>>> - accounting and limits have to be consistent
>>>   and should roughly represent the actual used
>>>   memory/swap (modulo optimizations, I can go
>>>   into detail here, if necessary)
>> This is true for current implementation for
>> booth - this patchset ang OpenVZ beancounters.
>>
>> If you sum up the physpages values for all containers
>> you'll get the exact number of RAM pages used.
>
> hmm, including or excluding the host pages?

Depends on whether you account host pages or not.

>>> - OOM handling on a per guest basis, i.e. some
>>>   out of memory condition in guest A must not
>>>   affect guest B
>> This is done in current patches.
>
>> Herbert, did you look at the patches before
>> sending this mail or do you just want to
>> 'take part' in conversation w/o understanding
>> of hat is going on?
>
> again, the question was about requirements, not
> your patches, and yes, I had a look at them _and_
> the OpenVZ implementations ...
>
> best,
> Herbert

>
> PS: hat is going on? :)
>
>>> HTC,
>>> Herbert
>>>
>>>> Sigh. Who is running this show? Anyone?
>>>>
>>>> You can actually do a form of overcommittment by allowing multiple
>>>> containers to share one or more of the zones. Whether that is
>>>> sufficient or suitable I don't know. That depends on the requirements,
>>>> and we haven't even discussed those, let alone agreed to them.
>>>>
>>>> _____
>>>> Containers mailing list
>>>> Containers@lists.osdl.org
>>>> https://lists.osdl.org/mailman/listinfo/containers
>

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 1/7] Resource counters
Posted by ebiederm on Tue, 13 Mar 2007 09:09:06 GMT
View Forum Message <> Reply to Message

Herbert Poetzl <herbert@13thfloor.at> writes:

> On Sun, Mar 11, 2007 at 01:00:15PM -0600, Eric W. Biederman wrote:
>> Herbert Poetzl <herbert@13thfloor.at> writes:
>>
>> >
>> > Linux-VServer does the accounting with atomic counters,
>> > so that works quite fine, just do the checks at the
>> > beginning of whatever resource allocation and the
>> > accounting once the resource is acquired ...
>>
>> Atomic operations versus locks is only a granularity thing.
>> You still need the cache line which is the cost on SMP.
>>
>> Are you using atomic_add_return or atomic_add_unless or
>> are you performing you actions in two separate steps
>> which is racy? What I have seen indicates you are using
>> a racy two separate operation form.
>

> yes, this is the current implementation which
> is more than sufficient, but I'm aware of the
> potential issues here, and I have an experimental
> patch sitting here which removes this race with
> the following change:
>
> - doesn't store the accounted value but
>   limit - accounted (i.e. the free resource)
> - uses atomic_add_return()
> - when negative, an error is returned and
>   the resource amount is added back
>
> changes to the limit have to adjust the 'current'
> value too, but that is again simple and atomic
>
> best,
> Herbert
>
> PS: atomic_add_unless() didn't exist back then
> (at least I think so) but that might be an option
> too ...

I think as far as having this discussion if you can remove that race
people will be more willing to talk about what vserver does.

That said anything that uses locks or atomic operations (finer grained locks)
because of the cache line ping pong is going to have scaling issues on large
boxes.

So in that sense anything short of per cpu variables sucks at scale. That said
I would much rather get a simple correct version without the complexity of
per cpu counters, before we optimize the counters that much.

Eric

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: Re: [RFC][PATCH 2/7] RSS controller core
Posted by ebiederm on Tue, 13 Mar 2007 09:26:53 GMT
View Forum Message <> Reply to Message

Kirill Korotaev <dev@openvz.org> writes:

> Eric,
>

>> And misses every resource sharing opportunity in sight.
>
> that was my point too.
>
>> Except for
>> filtering the which pages are eligible for reclaim an RSS limit should
>> not need to change the existing reclaim logic, and with things like the
>> memory zones we have had that kind of restriction in the reclaim logic
>> for a long time. So filtering out ineligible pages isn't anything new.
>
> exactly this is implemented in the current patches from Pavel.
> the only difference is that filtering is not done in general LRU list,
> which is not effective, but via per-container LRU list.
> So the pointer on the page structure does 2 things:
> - fast reclamation
   Better than the rmap list?
> - correct uncharging of page from where it was charged
>   (e.g. shared pages can be mapped first in one container, but the last unmap
>   done from another one).
   We should charge/uncharge all of them, not just one.

>>>We need to work out what the requirements are before we can settle on an
>>>implementation.
>>
>>
>> If you are talking about RSS limits the term is well defined. The
>> number of pages you can have mapped into your set of address space at
>> any given time.
>>
>> Unless I'm totally blind that isn't what the patchset implements.
>
> Ouch, what makes you think so?
> The fact that a page mapped into 2 different processes is charged only once?
> Imho it is much more correct then sum of process' RSS within container, due to:
> 1. it is clear how much container uses physical pages, not abstract items
> 2. shared pages are charged only once, so the sum of containers RSS is still
>   about physical RAM.

No the fact that a page mapped into 2 separate mm_structs in two
separate accounting domains is counted only once. This is very likely
to happen with things like glibc if you have a read-only shared copy
of your distro. There appears to be no technical reason for such a
restriction.

A page should not be owned.

Going further unless the limits are draconian I don't expect users to
hit the rss limits often or frequently. So in 99% of all cases page

reclaim should continue to be global. Which makes me question messing
with the general page reclaim lists.

Now if the normal limits turn out to be draconian it may make sense to
split the first level of page lists by some reasonable approximation
to their rss group, so we don't normally scan unnecessary pages.

>> The whole filesystem interface also is over general and makes it too
>> easy to express the hard things (like move an existing task from one
>> group of tasks to another) leading to code complications.
> the things which are not supported are easy to disable.

Maybe. The extra locking complexity gives me fits. But in the grand
scheme of things it is minor as long as it is not user perceptible we
can fix it later. I'm still wrapping my head around the weird fs concepts.

Eric

Subject: Re: [RFC][PATCH 1/7] Resource counters
Posted by xemul on Tue, 13 Mar 2007 09:27:15 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Herbert Poetzl <herbert@13thfloor.at> writes:
>
>> On Sun, Mar 11, 2007 at 01:00:15PM -0600, Eric W. Biederman wrote:
>>> Herbert Poetzl <herbert@13thfloor.at> writes:
>>>
>>>> Linux-VServer does the accounting with atomic counters,
>>>> so that works quite fine, just do the checks at the
>>>> beginning of whatever resource allocation and the
>>>> accounting once the resource is acquired ...
>>> Atomic operations versus locks is only a granularity thing.
>>> You still need the cache line which is the cost on SMP.
>>>
>>> Are you using atomic_add_return or atomic_add_unless or
>>> are you performing you actions in two separate steps
>>> which is racy? What I have seen indicates you are using
>>> a racy two separate operation form.
>> yes, this is the current implementation which
>> is more than sufficient, but I'm aware of the
>> potential issues here, and I have an experimental
>> patch sitting here which removes this race with
>> the following change:
>>
>> - doesn't store the accounted value but
>>   limit - accounted (i.e. the free resource)

>> - uses atomic_add_return()
>> - when negative, an error is returned and
>>   the resource amount is added back
>>
>> changes to the limit have to adjust the 'current'
>> value too, but that is again simple and atomic
>>
>> best,
>> Herbert
>>
>> PS: atomic_add_unless() didn't exist back then
>> (at least I think so) but that might be an option
>> too ...
>
> I think as far as having this discussion if you can remove that race
> people will be more willing to talk about what vserver does.
>
> That said anything that uses locks or atomic operations (finer grained locks)
> because of the cache line ping pong is going to have scaling issues on large
> boxes.

BTW atomic_add_unless() is essentially a loop!!! Just
like spin_lock() is, so why is one better that another?

spin_lock() can go to schedule() on preemptive kernels
thus increasing interactivity, while atomic can't.

> So in that sense anything short of per cpu variables sucks at scale. That said
> I would much rather get a simple correct version without the complexity of
> per cpu counters, before we optimize the counters that much.
>
> Eric
>

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: Re: [RFC][PATCH 1/7] Resource counters
Posted by dev on Tue, 13 Mar 2007 09:36:08 GMT
View Forum Message <> Reply to Message

>> - doesn't store the accounted value but
>>   limit - accounted (i.e. the free resource)
>> - uses atomic_add_return()
>> - when negative, an error is returned and

>> the resource amount is added back
>>
>>changes to the limit have to adjust the 'current'
>>value too, but that is again simple and atomic
>>
>>best,
>>Herbert
>>
>>PS: atomic_add_unless() didn't exist back then
>>(at least I think so) but that might be an option
>>too ...
>
>
> I think as far as having this discussion if you can remove that race
> people will be more willing to talk about what vserver does.
>
> That said anything that uses locks or atomic operations (finer grained locks)
> because of the cache line ping pong is going to have scaling issues on large
> boxes.

> So in that sense anything short of per cpu variables sucks at scale. That said
> I would much rather get a simple correct version without the complexity of
> per cpu counters, before we optimize the counters that much.
fully agree with it. We need to get a working version first.

FYI, in OVZ we recently added such optimizations: reserves like in TCP/IP,
e.g. for kmemsize, numfile these reserves are done on task-basis for
fast charges/uncharges w/o involving lock operations.
On task exit reserves are returned back to the beancounter.

As it demonstrated atomic counters can be replaced with
task-reserves on the next step.

Thanks,
Kirill

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by ebiederm on Tue, 13 Mar 2007 09:43:02 GMT
View Forum Message <> Reply to Message

Dave Hansen <hansendc@us.ibm.com> writes:

> On Mon, 2007-03-12 at 20:07 +0300, Kirill Korotaev wrote:
>> > On Mon, 2007-03-12 at 19:23 +0300, Kirill Korotaev wrote:
>> >>For these you essentially need per-container page->_mapcount counter,
>> >>otherwise you can't detect whether rss group still has the page in question
> being mapped

>> >>in its processes' address spaces or not.
>> >
>> > What do you mean by this? You can always tell whether a process has a
>> > particular page mapped. Could you explain the issue a bit more. I'm
>> > not sure I get it.
>> When we do charge/uncharge we have to answer on another question:
>> "whether *any* task from the *container* has this page mapped", not the
>> "whether *this* task has this page mapped".
>
> That's a bit more clear. ;)
>
> OK, just so I make sure I'm getting your argument here. It would be too
> expensive to go looking through all of the rmap data for _any_ other
> task that might be sharing the charge (in the same container) with the
> current task that is doing the unmapping.

Which is a questionable assumption. Worse case we are talking a list
several thousand entries long, and generally if you are used by the same
container you will hit one of your processes long before you traverse
the whole list.

So at least the average case performance should be good.

It is only in the case when you a page is shared between multiple
containers when this matters.

Eric

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by ebiederm on Tue, 13 Mar 2007 09:58:05 GMT
View Forum Message <> Reply to Message

Herbert Poetzl <herbert@13thfloor.at> writes:

> On Mon, Mar 12, 2007 at 09:50:08AM -0700, Dave Hansen wrote:
>> On Mon, 2007-03-12 at 19:23 +0300, Kirill Korotaev wrote:
>> >
>> > For these you essentially need per-container page->_mapcount counter,
>> > otherwise you can't detect whether rss group still has the page
>> > in question being mapped in its processes' address spaces or not.
>
>> What do you mean by this? You can always tell whether a process has a
>> particular page mapped. Could you explain the issue a bit more. I'm

>> not sure I get it.
>
> OpenVZ wants to account _shared_ pages in a guest
> different than separate pages, so that the RSS
> accounted values reflect the actual used RAM instead
> of the sum of all processes RSS' pages, which for
> sure is more relevant to the administrator, but IMHO
> not so terribly important to justify memory consuming
> structures and sacrifice performance to get it right
>
> YMMV, but maybe we can find a smart solution to the
> issue too :)

I will tell you what I want.

I want a shared page cache that has nothing to do with RSS limits.

I want an RSS limit that once I know I can run a deterministic
application with a fixed set of inputs in I want to know it will
always run.

First touch page ownership does not guarantee give me anything useful
for knowing if I can run my application or not. Because of page
sharing my application might run inside the rss limit only because
I got lucky and happened to share a lot of pages with another running
application. If the next I run and it isn't running my application
will fail. That is ridiculous.

I don't want sharing between vservers/VE/containers to affect how many
pages I can have mapped into my processes at once.

Now sharing is sufficiently rare that I'm pretty certain that problems
come up rarely. So maybe these problems have not shown up in testing
yet. But until I see the proof that actually doing the accounting for
sharing properly has intolerable overhead. I want proper accounting
not this hand waving that is only accurate on the third Tuesday of the
month.

Ideally all of this will be followed by smarter rss based swapping.
There are some very cool things that can be done to eliminate machine
overload once you have the ability to track real rss values.

Eric

_____

Subject: Re: Re: [RFC][PATCH 2/7] RSS controller core
Posted by dev on Tue, 13 Mar 2007 10:06:48 GMT
View Forum Message <> Reply to Message

Andrew Morton wrote:
>>>> - shared mappings of 'shared' files (binaries
>>>> and libraries) to allow for reduced memory
>>>> footprint when N identical guests are running
>>>
>>>So, it sounds like this can be phrased as a requirement like:
>>>
>>> "Guests must be able to share pages."
>>>
>>>Can you give us an idea why this is so?
>>
>>sure, one reason for this is that guests tend to
>>be similar (or almost identical) which results
>>in quite a lot of 'shared' libraries and executables
>>which would otherwise get cached for each guest and
>>would also be mapped for each guest separately
>
>
> noooooooo. What you're saying there amounts to text replication. There is
> no proposal here to create duplicated copies of pagecache pages: the VM
> just doesn't support that (Nick has soe protopatches which do this as a
> possible NUMA optimisation).
>
> So these mmapped pages will contiue to be shared across all guests. The
> problem boils down to "which guest(s) get charged for each shared page".
>
> A simple and obvious and easy-to-implement answer is "the guest which paged
> it in". I think we should firstly explain why that is insufficient.
I guess by "paged it in" you essentially mean
"mapped the page into address space for the *first* time"?

i.e. how many times the same page mapped into 2 address spaces
in the same container should be accounted for?

We believe ONE. It is better due to:
- it allows better estimate how much RAM container uses.
- if one container mapped a single page 10,000 times,
  it doesn't mean it is worse than a container which mapped only 200 pages
  and that it should be killed in case of OOM.

Thanks,
Kirill

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by Nick Piggin on Tue, 13 Mar 2007 10:25:49 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Herbert Poetzl <herbert@13thfloor.at> writes:
>
>
>>On Mon, Mar 12, 2007 at 09:50:08AM -0700, Dave Hansen wrote:
>>
>>>On Mon, 2007-03-12 at 19:23 +0300, Kirill Korotaev wrote:
>>>
>>>>For these you essentially need per-container page->_mapcount counter,
>>>>otherwise you can't detect whether rss group still has the page
>>>>in question being mapped in its processes' address spaces or not.
>>
>>>What do you mean by this? You can always tell whether a process has a
>>>particular page mapped. Could you explain the issue a bit more. I'm
>>>not sure I get it.
>>
>>OpenVZ wants to account _shared_ pages in a guest
>>different than separate pages, so that the RSS
>>accounted values reflect the actual used RAM instead
>>of the sum of all processes RSS' pages, which for
>>sure is more relevant to the administrator, but IMHO
>>not so terribly important to justify memory consuming
>>structures and sacrifice performance to get it right
>>
>>YMMV, but maybe we can find a smart solution to the
>>issue too :)
>
>
> I will tell you what I want.
>
> I want a shared page cache that has nothing to do with RSS limits.
>
> I want an RSS limit that once I know I can run a deterministic
> application with a fixed set of inputs in I want to know it will
> always run.
>
> First touch page ownership does not guarantee give me anything useful
> for knowing if I can run my application or not. Because of page
> sharing my application might run inside the rss limit only because
> I got lucky and happened to share a lot of pages with another running
> application. If the next I run and it isn't running my application
> will fail. That is ridiculous.

Let's be practical here, what you're asking is basically impossible.

Unless by deterministic you mean that it never enters the a non
trivial syscall, in which case, you just want to know about maximum
RSS of the process, which we already account).

> I don't want sharing between vservers/VE/containers to affect how many
> pages I can have mapped into my processes at once.

You seem to want total isolation. You could use virtualization?

> Now sharing is sufficiently rare that I'm pretty certain that problems
> come up rarely. So maybe these problems have not shown up in testing
> yet. But until I see the proof that actually doing the accounting for
> sharing properly has intolerable overhead. I want proper accounting
> not this hand waving that is only accurate on the third Tuesday of the
> month.

It is basically handwaving anyway. The only approach I've seen with
a sane (not perfect, but good) way of accounting memory use is this
one. If you care to define "proper", then we could discuss that.

--
SUSE Labs, Novell Inc.

_____

Subject: Re: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Andrew Morton on Tue, 13 Mar 2007 10:49:26 GMT
View Forum Message <> Reply to Message

> On Tue, 13 Mar 2007 13:19:53 +0300 Kirill Korotaev <dev@sw.ru> wrote:
> Andrew Morton wrote:
> >>>> - shared mappings of 'shared' files (binaries
> >>>>   and libraries) to allow for reduced memory
> >>>>   footprint when N identical guests are running
> >>>
> >>>So, it sounds like this can be phrased as a requirement like:
> >>>
> >>> "Guests must be able to share pages."
> >>>
> >>>Can you give us an idea why this is so?
> >>
> >>sure, one reason for this is that guests tend to
> >>be similar (or almost identical) which results

> >>in quite a lot of 'shared' libraries and executables
> >>which would otherwise get cached for each guest and
> >>would also be mapped for each guest separately
> >
> >
> > noooooooo. What you're saying there amounts to text replication. There is
> > no proposal here to create duplicated copies of pagecache pages: the VM
> > just doesn't support that (Nick has soe protopatches which do this as a
> > possible NUMA optimisation).
> >
> > So these mmapped pages will contiue to be shared across all guests. The
> > problem boils down to "which guest(s) get charged for each shared page".
> >
> > A simple and obvious and easy-to-implement answer is "the guest which paged
> > it in". I think we should firstly explain why that is insufficient.
> I guess by "paged it in" you essentially mean
> "mapped the page into address space for the *first* time"?

Not really - I mean "first allocated the page". ie: major fault(), read(),
write(), etc.

> i.e. how many times the same page mapped into 2 address spaces
> in the same container should be accounted for?
>
> We believe ONE. It is better due to:
> - it allows better estimate how much RAM container uses.
> - if one container mapped a single page 10,000 times,
>   it doesn't mean it is worse than a container which mapped only 200 pages
>   and that it should be killed in case of OOM.

I'm not sure that we need to account for pages at all, nor care about rss.

If we use a physical zone-based containment scheme: fake-numa,
variable-sized zones, etc then it all becomes moot. You set up a container
which has 1.5GB of physial memory then toss processes into it. As that
process set increases in size it will toss out stray pages which shouldn't
be there, then it will start reclaiming and swapping out its own pages and
eventually it'll get an oom-killing.

No RSS acounting or page acounting in sight, because we already *have* that
stuff, at the physical level, in the zone.

Overcommitment can be performed by allowing different containers to share
the same zone set, or by dynamically increasing or decreasing the size of
a physical container.

This all works today with fake-numa and cpusets, no kernel changes needed.

It could be made to work fairly simply with a multi-zone approach, or with
resizeable zones.

I'd be interested in knowing what you think the shortcomings of this are
likely to be,.

Subject: Re: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Herbert Poetzl on Tue, 13 Mar 2007 14:59:38 GMT
View Forum Message <> Reply to Message

On Tue, Mar 13, 2007 at 03:48:34AM -0800, Andrew Morton wrote:
> > On Tue, 13 Mar 2007 13:19:53 +0300 Kirill Korotaev <dev@sw.ru> wrote:
> > Andrew Morton wrote:
> > >>>> - shared mappings of 'shared' files (binaries
> > >>>> and libraries) to allow for reduced memory
> > >>>> footprint when N identical guests are running
> > >>>
> > >>>So, it sounds like this can be phrased as a requirement like:
> > >>>
> > >>> "Guests must be able to share pages."
> > >>>
> > >>>Can you give us an idea why this is so?
> > >>
> > >>sure, one reason for this is that guests tend to
> > >>be similar (or almost identical) which results
> > >>in quite a lot of 'shared' libraries and executables
> > >>which would otherwise get cached for each guest and
> > >>would also be mapped for each guest separately
> > >
> > > nooooooo. What you're saying there amounts to text replication.
> > > There is no proposal here to create duplicated copies of pagecache
> > > pages: the VM just doesn't support that (Nick has soe protopatches
> > > which do this as a possible NUMA optimisation).
> > >
> > > So these mmapped pages will contiue to be shared across all
> > > guests. The problem boils down to "which guest(s) get charged for
> > > each shared page".
> > >
> > > A simple and obvious and easy-to-implement answer is "the guest
> > > which paged it in". I think we should firstly explain why that is
> > > insufficient.
>
> > I guess by "paged it in" you essentially mean
> > "mapped the page into address space for the *first* time"?
>
> Not really - I mean "first allocated the page". ie: major fault(),
> read(), write(), etc.

>
> > i.e. how many times the same page mapped into 2 address spaces
> > in the same container should be accounted for?
> >
> > We believe ONE. It is better due to:
> > - it allows better estimate how much RAM container uses.
> > - if one container mapped a single page 10,000 times,
> >   it doesn't mean it is worse than a container which mapped only 200
> >   pages and that it should be killed in case of OOM.
>
> I'm not sure that we need to account for pages at all, nor care about
> rss.
>
> If we use a physical zone-based containment scheme: fake-numa,
> variable-sized zones, etc then it all becomes moot.

sounds good to me, just not sure it provides what we
need, but I'm sure I'll figure that with your help ...

> You set up a container which has 1.5GB of physial memory then toss
> processes into it. As that process set increases in size it will
> toss out stray pages which shouldn't be there, then it will start
> reclaiming and swapping out its own pages and eventually it'll get an
> oom-killing.

okay, let me ask a few naive questions about this scheme:

how does this work for a _file_ which is shared between
two guests (e.g. an executable like bash, hardlinked
between guests) when both guests are in a different
zone-based container?

+ assumed that the file is read in the first time,
  will it be accounted to the first guest doing so?

+ assumed it is accessed in the second guest, will
  it cause any additional cache/mapping besides the
  dentry stuff?

+ will container A be able to 'toss out' pages
  'shared' with container B (assumed sharing is
  possible :)

+ when the container A tosses out the pages for this
  executable, will guest B still be able to use them?

+ when the pages are tossed out, will they require
  the system to read them in again, or will they

stay available ala swap cache?

> No RSS acounting or page acounting in sight, because we already *have*
> that stuff, at the physical level, in the zone.

I'm fine with that ...

> Overcommitment can be performed by allowing different containers to
> share the same zone set, or by dynamically increasing or decreasing
> the size of a physical container.

here the question is, can a guest have several of
those 'virtual zones' assigned, so that there is a
container specific and a shared zone for example?

> This all works today with fake-numa and cpusets, no kernel changes
> needed.

sounds good!

> It could be made to work fairly simply with a multi-zone approach, or
> with resizeable zones.
>
> I'd be interested in knowing what you think the shortcomings of
> this are likely to be,.

will do so once I have a better understanding how this
approach will work ...

TIA,
Herbert

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Herbert Poetzl on Tue, 13 Mar 2007 15:05:10 GMT
View Forum Message <> Reply to Message

On Tue, Mar 13, 2007 at 10:17:54AM +0300, Pavel Emelianov wrote:
> Herbert Poetzl wrote:
> > On Mon, Mar 12, 2007 at 12:02:01PM +0300, Pavel Emelianov wrote:
> >>>> Maybe you have some ideas how we can decide on this?
> >>> We need to work out what the requirements are before we can
> >>> settle on an implementation.
> >> Linux-VServer (and probably OpenVZ):
> >>
> >> - shared mappings of 'shared' files (binaries
> >>   and libraries) to allow for reduced memory
> >>   footprint when N identical guests are running

> >> This is done in current patches.
> >
> > nice, but the question was about _requirements_
> > (so your requirements are?)
> >
> >>> - virtual 'physical' limit should not cause
> >>>   swap out when there are still pages left on
> >>>   the host system (but pages of over limit guests
> >>>   can be preferred for swapping)
> >> So what to do when virtual physical limit is hit?
> >> OOM-kill current task?
> >
> > when the RSS limit is hit, but there _are_ enough
> > pages left on the physical system, there is no
> > good reason to swap out the page at all
> >
> > - there is no benefit in doing so (performance
> >   wise, that is)
> >
> > - it actually hurts performance, and could
> >   become a separate source for DoS
> >
> > what should happen instead (in an ideal world :)
> > is that the page is considered swapped out for
> > the guest (add guest penality for swapout), and
>
> Is the page stays mapped for the container or not?
> If yes then what's the use of limits? Container mapped
> pages more than the limit is but all the pages are
> still in memory. Sounds weird.

sounds weird, but makes sense if you look at the full picture

just because the guest is over its page limit doesn't
mean that you actually want the system to swap stuff
out, what you really want to happen is the following:

 - somehow mark those pages as 'gone' for the guest
 - penalize the guest (and only the guest) for the
   'virtual' swap/page operation
 - penalize the guest again for paging in the page
 - drop/swap/page out those pages when the host system
   decides to reclaim pages (from the host PoV)

> > when the page would be swapped in again, the guest
> > takes a penalty (for the 'virtual' page in) and
> > the page is returned to the guest, possibly kicking
> > out (again virtually) a different page

> >
> >>> - accounting and limits have to be consistent
> >>>   and should roughly represent the actual used
> >>>   memory/swap (modulo optimizations, I can go
> >>>   into detail here, if necessary)
> >> This is true for current implementation for
> >> booth - this patchset ang OpenVZ beancounters.
> >>
> >> If you sum up the physpages values for all containers
> >> you'll get the exact number of RAM pages used.
> >
> > hmm, including or excluding the host pages?
>
> Depends on whether you account host pages or not.

you tell me? or is that an option in OpenVZ?

best,
Herbert

> >>> - OOM handling on a per guest basis, i.e. some
> >>>   out of memory condition in guest A must not
> >>>   affect guest B
> >> This is done in current patches.
> >
> >> Herbert, did you look at the patches before
> >> sending this mail or do you just want to
> >> 'take part' in conversation w/o understanding
> >> of hat is going on?
> >
> > again, the question was about requirements, not
> > your patches, and yes, I had a look at them _and_
> > the OpenVZ implementations ...
> >
> > best,
> > Herbert
> >
> > PS: hat is going on? :)
> >
> >>> HTC,
> >>> Herbert
> >>>
> >>>> Sigh. Who is running this show? Anyone?
> >>>>
> >>>> You can actually do a form of overcommittment by allowing multiple
> >>>> containers to share one or more of the zones. Whether that is
> >>>> sufficient or suitable I don't know. That depends on the requirements,
> >>>> and we haven't even discussed those, let alone agreed to them.

> >>>>
> >>>> _____
> >>>> Containers mailing list
> >>>> Containers@lists.osdl.org
> >>>> https://lists.osdl.org/mailman/listinfo/containers
> >

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by dev on Tue, 13 Mar 2007 15:10:55 GMT
View Forum Message <> Reply to Message

>>So what to do when virtual physical limit is hit?
>>OOM-kill current task?
>
>
> when the RSS limit is hit, but there _are_ enough
> pages left on the physical system, there is no
> good reason to swap out the page at all
>
> - there is no benefit in doing so (performance
>   wise, that is)
>
> - it actually hurts performance, and could
>   become a separate source for DoS
>
> what should happen instead (in an ideal world :)
> is that the page is considered swapped out for
> the guest (add guest penality for swapout), and
> when the page would be swapped in again, the guest
> takes a penalty (for the 'virtual' page in) and
> the page is returned to the guest, possibly kicking
> out (again virtually) a different page

great. I agree with that.
Just curious why current vserver code kills arbitrary
task in container then?

>>> - accounting and limits have to be consistent
>>>   and should roughly represent the actual used
>>>   memory/swap (modulo optimizations, I can go
>>>   into detail here, if necessary)
>>

>>This is true for current implementation for
>>booth - this patchset ang OpenVZ beancounters.
>>
>>If you sum up the physpages values for all containers
>>you'll get the exact number of RAM pages used.
>
>
> hmm, including or excluding the host pages?

depends on whether you will include beanocunter 0 usages or not :)

Kirill

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Herbert Poetzl on Tue, 13 Mar 2007 15:11:55 GMT
View Forum Message <> Reply to Message

On Tue, Mar 13, 2007 at 06:10:55PM +0300, Kirill Korotaev wrote:
> >>So what to do when virtual physical limit is hit?
> >>OOM-kill current task?
> >
> >
> > when the RSS limit is hit, but there _are_ enough
> > pages left on the physical system, there is no
> > good reason to swap out the page at all
> >
> > - there is no benefit in doing so (performance
> >   wise, that is)
> >
> > - it actually hurts performance, and could
> >   become a separate source for DoS
> >
> > what should happen instead (in an ideal world :)
> > is that the page is considered swapped out for
> > the guest (add guest penality for swapout), and
> > when the page would be swapped in again, the guest
> > takes a penalty (for the 'virtual' page in) and
> > the page is returned to the guest, possibly kicking
> > out (again virtually) a different page
>
> great. I agree with that.

> Just curious why current vserver code kills arbitrary

> task in container then?

because it obviously lacks the finess of OpenVZ code :)

seriously, handling the OOM kills inside a container
has never been a real world issue, as once you are
really out of memory (and OOM starts killing) you
usually have lost the game anyways (i.e. a guest restart
or similar is required to get your services up and
running again) and OOM killer decisions are not perfect
in mainline either, but, you've probably seen the
FIXME and TODO entries in the code showing that this
is work in progress ...

> >>> - accounting and limits have to be consistent
> >>> and should roughly represent the actual used
> >>> memory/swap (modulo optimizations, I can go
> >>> into detail here, if necessary)
> >>
> >>This is true for current implementation for
> >>booth - this patchset ang OpenVZ beancounters.
> >>
> >>If you sum up the physpages values for all containers
> >>you'll get the exact number of RAM pages used.
> >
> >
> > hmm, including or excluding the host pages?
>
> depends on whether you will include beanocunter 0 usages or not :)

so that is an option then?

best,
Herbert

> Kirill

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 1/7] Resource counters
Posted by Herbert Poetzl on Tue, 13 Mar 2007 15:21:50 GMT
View Forum Message <> Reply to Message

On Tue, Mar 13, 2007 at 03:09:06AM -0600, Eric W. Biederman wrote:
> Herbert Poetzl <herbert@13thfloor.at> writes:

>
> > On Sun, Mar 11, 2007 at 01:00:15PM -0600, Eric W. Biederman wrote:
> >> Herbert Poetzl <herbert@13thfloor.at> writes:
> >>
> >> >
> >> > Linux-VServer does the accounting with atomic counters,
> >> > so that works quite fine, just do the checks at the
> >> > beginning of whatever resource allocation and the
> >> > accounting once the resource is acquired ...
> >>
> >> Atomic operations versus locks is only a granularity thing.
> >> You still need the cache line which is the cost on SMP.
> >>
> >> Are you using atomic_add_return or atomic_add_unless or
> >> are you performing you actions in two separate steps
> >> which is racy? What I have seen indicates you are using
> >> a racy two separate operation form.
> >
> > yes, this is the current implementation which
> > is more than sufficient, but I'm aware of the
> > potential issues here, and I have an experimental
> > patch sitting here which removes this race with
> > the following change:
> >
> > - doesn't store the accounted value but
> >   limit - accounted (i.e. the free resource)
> > - uses atomic_add_return()
> > - when negative, an error is returned and
> >   the resource amount is added back
> >
> > changes to the limit have to adjust the 'current'
> > value too, but that is again simple and atomic
> >
> > best,
> > Herbert
> >
> > PS: atomic_add_unless() didn't exist back then
> > (at least I think so) but that might be an option
> > too ...
>
> I think as far as having this discussion if you can remove that race
> people will be more willing to talk about what vserver does.

well, shouldn't be a big deal to brush that patch up
(if somebody actually _is_ interested)

> That said anything that uses locks or atomic operations (finer grained
> locks) because of the cache line ping pong is going to have scaling

> issues on large boxes.

right, but atomic ops have much less impact on most
architectures than locks :)

> So in that sense anything short of per cpu variables sucks at scale.
> That said I would much rather get a simple correct version without the
> complexity of per cpu counters, before we optimize the counters that
> much.

actually I thought about per cpu counters quite a lot, and
we (Llinux-VServer) use them for accounting, but please
tell me how you use per cpu structures for implementing
limits

TIA,
Herbert

> Eric

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: Re: [RFC][PATCH 2/7] RSS controller core
Posted by dev on Tue, 13 Mar 2007 15:30:40 GMT
View Forum Message <> Reply to Message

Eric,

>>>And misses every resource sharing opportunity in sight.
>>
>>that was my point too.
>>
>>
>>>Except for
>>>filtering the which pages are eligible for reclaim an RSS limit should
>>>not need to change the existing reclaim logic, and with things like the
>>>memory zones we have had that kind of restriction in the reclaim logic
>>>for a long time. So filtering out ineligible pages isn't anything new.
>>
>>exactly this is implemented in the current patches from Pavel.
>>the only difference is that filtering is not done in general LRU list,
>>which is not effective, but via per-container LRU list.
>>So the pointer on the page structure does 2 things:
>>- fast reclamation

>
>   Better than the rmap list?
>
>>- correct uncharging of page from where it was charged
>> (e.g. shared pages can be mapped first in one container, but the last unmap
>>   done from another one).
>
>   We should charge/uncharge all of them, not just one.
>
>
>>>>We need to work out what the requirements are before we can settle on an
>>>>implementation.
>>>
>>>
>>>If you are talking about RSS limits the term is well defined. The
>>>number of pages you can have mapped into your set of address space at
>>>any given time.
>>>
>>>Unless I'm totally blind that isn't what the patchset implements.
>>
>>Ouch, what makes you think so?
>>The fact that a page mapped into 2 different processes is charged only once?
>>Imho it is much more correct then sum of process' RSS within container, due to:
>>1. it is clear how much container uses physical pages, not abstract items
>>2. shared pages are charged only once, so the sum of containers RSS is still
>>   about physical RAM.
>
>
> No the fact that a page mapped into 2 separate mm_structs in two
> separate accounting domains is counted only once. This is very likely
> to happen with things like glibc if you have a read-only shared copy
> of your distro. There appears to be no technical reason for such a
> restriction.
>
> A page should not be owned.

I would be happy to propose OVZ approach then, where a page is tracked
with page_beancounter data structure, which ties together
a page with beancounters which use it like this:

page -> page_beancounter -> list of beanocunters which has the page mapped

This gives a number of advantages:
- the page is accounted to all the VEs which actually use it.
- allows almost accurate tracking of page fractions used by VEs
  depending on how many VEs mapped the page.
- allows to track dirty pages, i.e. which VE dirtied the page
  and implement correct disk I/O accounting and CFQ write scheduling

based on VE priorities.

> Going further unless the limits are draconian I don't expect users to
> hit the rss limits often or frequently. So in 99% of all cases page
> reclaim should continue to be global. Which makes me question messing
> with the general page reclaim lists.

It is not that rare when containers hit their limits, believe me :/
In trusted environments - probably you are right, in hosting - no.

Thanks,
Kirill

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by xemul on Tue, 13 Mar 2007 15:32:08 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Tue, Mar 13, 2007 at 10:17:54AM +0300, Pavel Emelianov wrote:
>> Herbert Poetzl wrote:
>>> On Mon, Mar 12, 2007 at 12:02:01PM +0300, Pavel Emelianov wrote:
>>>>>> Maybe you have some ideas how we can decide on this?
>>>>>> We need to work out what the requirements are before we can
>>>>>> settle on an implementation.
>>>>> Linux-VServer (and probably OpenVZ):
>>>>>
>>>>> - shared mappings of 'shared' files (binaries
>>>>> and libraries) to allow for reduced memory
>>>>> footprint when N identical guests are running
>>>> This is done in current patches.
>>> nice, but the question was about _requirements_
>>> (so your requirements are?)
>>>
>>>>> - virtual 'physical' limit should not cause
>>>>> swap out when there are still pages left on
>>>>> the host system (but pages of over limit guests
>>>>> can be preferred for swapping)
>>>> So what to do when virtual physical limit is hit?
>>>> OOM-kill current task?
>>> when the RSS limit is hit, but there _are_ enough
>>> pages left on the physical system, there is no
>>> good reason to swap out the page at all
>>>
>>> - there is no benefit in doing so (performance
>>> wise, that is)
>>>
>>> - it actually hurts performance, and could

>>>   become a separate source for DoS
>>>
>>> what should happen instead (in an ideal world :)
>>> is that the page is considered swapped out for
>>> the guest (add guest penality for swapout), and
>> Is the page stays mapped for the container or not?
>> If yes then what's the use of limits? Container mapped
>> pages more than the limit is but all the pages are
>> still in memory. Sounds weird.
>
> sounds weird, but makes sense if you look at the full picture
>
> just because the guest is over its page limit doesn't
> mean that you actually want the system to swap stuff
> out, what you really want to happen is the following:
>
> - somehow mark those pages as 'gone' for the guest
> - penalize the guest (and only the guest) for the
>   'virtual' swap/page operation
> - penalize the guest again for paging in the page
> - drop/swap/page out those pages when the host system
>   decides to reclaim pages (from the host PoV)

Yeah! And slow down the container which caused global
limit hit (w/o hitting it's own limit!) by swapping
some others' pages out. This breaks the idea of isolation.

>>> when the page would be swapped in again, the guest
>>> takes a penalty (for the 'virtual' page in) and
>>> the page is returned to the guest, possibly kicking
>>> out (again virtually) a different page
>>>
>>>>> - accounting and limits have to be consistent
>>>>>   and should roughly represent the actual used
>>>>>   memory/swap (modulo optimizations, I can go
>>>>>   into detail here, if necessary)
>>>> This is true for current implementation for
>>>> booth - this patchset ang OpenVZ beancounters.
>>>>
>>>> If you sum up the physpages values for all containers
>>>> you'll get the exact number of RAM pages used.
>>> hmm, including or excluding the host pages?
>> Depends on whether you account host pages or not.
>
> you tell me? or is that an option in OpenVZ?

In OpenVZ we account resources in host system as well.
However we have an opportunity to turn this off.

> best,
> Herbert
>
>>>>> - OOM handling on a per guest basis, i.e. some
>>>>>   out of memory condition in guest A must not
>>>>>   affect guest B
>>>> This is done in current patches.
>>>> Herbert, did you look at the patches before
>>>> sending this mail or do you just want to
>>>> 'take part' in conversation w/o understanding
>>>> of hat is going on?
>>> again, the question was about requirements, not
>>> your patches, and yes, I had a look at them _and_
>>> the OpenVZ implementations ...
>>>
>>> best,
>>> Herbert
>>>
>>> PS: hat is going on? :)
>>>
>>>>>>> HTC,
>>>>>>> Herbert
>>>>>>>
>>>>>>> Sigh. Who is running this show?   Anyone?
>>>>>>>
>>>>>>> You can actually do a form of overcommittment by allowing multiple
>>>>>>> containers to share one or more of the zones. Whether that is
>>>>>>> sufficient or suitable I don't know. That depends on the requirements,
>>>>>>> and we haven't even discussed those, let alone agreed to them.
>>>>>>>
>>>>>>> _____
>>>>>>> Containers mailing list
>>>>>>> Containers@lists.osdl.org
>>>>>>> https://lists.osdl.org/mailman/listinfo/containers
>

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 1/7] Resource counters
Posted by xemul on Tue, 13 Mar 2007 15:41:05 GMT
View Forum Message <> Reply to Message

>>> PS: atomic_add_unless() didn't exist back then

>>> (at least I think so) but that might be an option
>>> too ...
>> I think as far as having this discussion if you can remove that race
>> people will be more willing to talk about what vserver does.
>
> well, shouldn't be a big deal to brush that patch up
> (if somebody actually _is_ interested)
>
>> That said anything that uses locks or atomic operations (finer grained
>> locks) because of the cache line ping pong is going to have scaling
>> issues on large boxes.
>
> right, but atomic ops have much less impact on most
> architectures than locks :)

Right. But atomic_add_unless() is slower as it is
essentially a loop. See my previous letter in this sub-thread.

>> So in that sense anything short of per cpu variables sucks at scale.
>> That said I would much rather get a simple correct version without the
>> complexity of per cpu counters, before we optimize the counters that
>> much.
>
> actually I thought about per cpu counters quite a lot, and
> we (Llinux-VServer) use them for accounting, but please
> tell me how you use per cpu structures for implementing
> limits

Did you ever look at how get_empty_filp() works?
I agree, that this is not a "strict" limit, but it
limits the usage wit some "precision".

/* off-the-topic */ Herbert, you've lost Balbir again:
In this sub-thread some letters up Eric wrote a letter with
Balbir in Cc:. The next reply from you doesn't include him.

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by dev on Tue, 13 Mar 2007 15:54:39 GMT
View Forum Message <> Reply to Message

Herbert,

>>Just curious why current vserver code kills arbitrary

>>task in container then?
>
>
> because it obviously lacks the finess of OpenVZ code :)
>
> seriously, handling the OOM kills inside a container
> has never been a real world issue, as once you are
> really out of memory (and OOM starts killing) you
> usually have lost the game anyways (i.e. a guest restart
> or similar is required to get your services up and
> running again) and OOM killer decisions are not perfect
> in mainline either, but, you've probably seen the
> FIXME and TODO entries in the code showing that this
> is work in progress ...

I'm talking not about the finess of the code,
but rather about the lack of isolation,
i.e. one VE can affect others.

Kirill

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by ebiederm on Tue, 13 Mar 2007 16:01:54 GMT
View Forum Message <> Reply to Message

Nick Piggin <nickpiggin@yahoo.com.au> writes:

> Eric W. Biederman wrote:
>>
>> First touch page ownership does not guarantee give me anything useful
>> for knowing if I can run my application or not. Because of page
>> sharing my application might run inside the rss limit only because
>> I got lucky and happened to share a lot of pages with another running
>> application. If the next I run and it isn't running my application
>> will fail. That is ridiculous.
>
> Let's be practical here, what you're asking is basically impossible.
>
> Unless by deterministic you mean that it never enters the a non
> trivial syscall, in which case, you just want to know about maximum
> RSS of the process, which we already account).

Not per process I want this on a group of processes, and yes that

is all I want just. I just want accounting of the maximum RSS of
a group of processes and then the mechanism to limit that maximum rss.

>> I don't want sharing between vservers/VE/containers to affect how many
>> pages I can have mapped into my processes at once.
>
> You seem to want total isolation. You could use virtualization?

No. I don't want the meaning of my rss limit to be affected by what
other processes are doing. We have constraints of how many resources
the box actually has. But I don't want accounting so sloppy that
processes outside my group of processes can artificially
lower my rss value, which magically raises my rss limit.

>> Now sharing is sufficiently rare that I'm pretty certain that problems
>> come up rarely. So maybe these problems have not shown up in testing
>> yet. But until I see the proof that actually doing the accounting for
>> sharing properly has intolerable overhead. I want proper accounting
>> not this hand waving that is only accurate on the third Tuesday of the
>> month.
>
> It is basically handwaving anyway. The only approach I've seen with
> a sane (not perfect, but good) way of accounting memory use is this
> one. If you care to define "proper", then we could discuss that.

I will agree that this patchset is probably in the right general ballpark.
But the fact that pages are assigned exactly one owner is pure non-sense.
We can do better. That is all I am asking for someone to at least attempt
to actually account for the rss of a group of processes and get the numbers
right when we have shared pages, between different groups of
processes. We have the data structures to support this with rmap.

Let me describe the situation where I think the accounting in the
patchset goes totally wonky.

Gcc as I recall maps the pages it is compiling with mmap.
If in a single kernel tree I do:
make -jN O=../compile1 &
make -jN O=../compile2 &

But set it up so that the two compiles are in different rss groups.
If I run the concurrently they will use the same files at the same
time and most likely because of the first touch rss limit rule even
if I have a draconian rss limit the compiles will both be able to
complete and finish. However if I run either of them alone if I
use the most draconian rss limit I can that allows both compiles to
finish I won't be able to compile a single kernel tree.

The reason for the failure with a single tree (in my thought
experiment) is that the rss limit was set below the what is actually
needed for the code to work. When we were compiling two kernels and
they were mapping the same pages at the same time we could put the rss
limit below the minimum rss needed for the compile to execute and
still have it complete because of with first touch only one group
accounted for the pages and the other just leached of the first, as
long as both compiles grabbed some of the pages they could complete.

No I know in practice most draconian limits will simply result in the
page staying in the page cache but not mapped into processes in the
group with the draconian limit, or they will result in pages of the
group with the draconian limit being pushed out into the swap cache.
So the chances of actual application failure even with a draconian
rss limit are quite unlikely. (I actually really appreciate this
fact).

However the messed up accounting that doesn't handle sharing between
groups of processes properly really bugs me. Especially when we have
the infrastructure to do it right.

Does that make more sense?

Eric

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Herbert Poetzl on Tue, 13 Mar 2007 16:06:32 GMT
View Forum Message <> Reply to Message

On Tue, Mar 13, 2007 at 07:27:06AM +0530, Balbir Singh wrote:
> > hmm, it is very unlikely that this would happen,
> > for several reasons ... and indeed, checking the
> > thread in my mailbox shows that akpm dropped you ...
>
> But, I got Andrew's email.
>
> > ----------
> > Subject: [RFC][PATCH 2/7] RSS controller core
> > From: Pavel Emelianov <xemul@sw.ru>
> > To: Andrew Morton <akpm@osdl.org>, Paul Menage <menage@google.com>,
> > Srivatsa Vaddagiri <vatsa@in.ibm.com>,
> > Balbir Singh <balbir@in.ibm.com>

> > Cc: containers@lists.osdl.org,
> > Linux Kernel Mailing List <linux-kernel@vger.kernel.org>
> > Date: Tue, 06 Mar 2007 17:55:29 +0300
> > ---
> > Subject: Re: [RFC][PATCH 2/7] RSS controller core
> > From: Andrew Morton <akpm@linux-foundation.org>
> > To: Pavel Emelianov <xemul@sw.ru>
> > Cc: Kirill@smtp.osdl.org, Linux@smtp.osdl.org, containers@lists.osdl.org,
> > Paul Menage <menage@google.com>,
> > List <linux-kernel@vger.kernel.org>
> > Date: Tue, 6 Mar 2007 14:00:36 -0800
> > ---
> > that's the one I 'group' replied to ...
> >
> > > Could you please not modify the "cc" list.
> >
> > I never modify the cc unless explicitely asked
> > to do so. I wish others would have it that way
> > too :)
>
> Thats good to know, but my mailer shows
>
>
> Andrew Morton <akpm@linux-foundation.org>
> to Pavel Emelianov <xemul@sw.ru>
> cc
> Paul Menage <menage@google.com>,
> Srivatsa Vaddagiri <vatsa@in.ibm.com>,
> Balbir Singh <balbir@in.ibm.com> (see I am <<HERE>>),
> devel@openvz.org,
> Linux Kernel Mailing List <linux-kernel@vger.kernel.org>,
> containers@lists.osdl.org,
> Kirill Korotaev <dev@sw.ru>
> date Mar 7, 2007 3:30 AM
> subject Re: [RFC][PATCH 2/7] RSS controller core
> mailed-by vger.kernel.org
> On Tue, 06 Mar 2007 17:55:29 +0300
>
> and your reply as
>
> Andrew Morton <akpm@linux-foundation.org>,
> Pavel Emelianov <xemul@sw.ru>,
> Kirill@smtp.osdl.org,
> Linux@smtp.osdl.org,
> containers@lists.osdl.org,
> Paul Menage <menage@google.com>,
> List <linux-kernel@vger.kernel.org>
> to Andrew Morton <akpm@linux-foundation.org>

> cc
> Pavel Emelianov <xemul@sw.ru>,
> Kirill@smtp.osdl.org,
> Linux@smtp.osdl.org,
> containers@lists.osdl.org,
> Paul Menage <menage@google.com>,
> List <linux-kernel@vger.kernel.org>
> date Mar 9, 2007 10:18 PM
> subject Re: [RFC][PATCH 2/7] RSS controller core
> mailed-by vger.kernel.org
>
> I am not sure what went wrong. Could you please check your mail
> client, cause it seemed to even change email address to smtp.osdl.org
> which bounced back when I wrote to you earlier.

my mail client is not involved in receiving the emails,
so the email I replied to did already miss you in the cc
(i.e. I doubt that mutt would hide you from the cc, if
it would be present in the mailbox :)

maybe one of the mailing lists is removing receipients
according to some strange scheme?

here are the full headers for the email I replied to:

-8<----------
>From containers-bounces@lists.osdl.org Tue Mar 6 23:01:21 2007
Return-Path: containers-bounces@lists.osdl.org
X-Original-To: herbert@13thfloor.at
Delivered-To: herbert@13thfloor.at
Received: from smtp.osdl.org (smtp.osdl.org [65.172.181.24])
(using TLSv1 with cipher EDH-RSA-DES-CBC3-SHA (168/168 bits))
(No client certificate requested)
by mail.13thfloor.at (Postfix) with ESMTP id 0CD0F707FC
for <herbert@13thfloor.at>; Tue, 6 Mar 2007 23:00:52 +0100 (CET)
Received: from fire-2.osdl.org (localhost [127.0.0.1])
by smtp.osdl.org (8.12.8/8.12.8) with ESMTP id l26M0eqA023167;
Tue, 6 Mar 2007 14:00:47 -0800
Received: from shell0.pdx.osdl.net (fw.osdl.org [65.172.181.6])
by smtp.osdl.org (8.12.8/8.12.8) with ESMTP id l26M0bq8023159
(version=TLSv1/SSLv3 cipher=EDH-RSA-DES-CBC3-SHA bits=168 verify=NO);
Tue, 6 Mar 2007 14:00:37 -0800
Received: from akpm.corp.google.com (shell0.pdx.osdl.net [10.9.0.31])
by shell0.pdx.osdl.net (8.13.1/8.11.6) with SMTP id l26M0ate010730;
Tue, 6 Mar 2007 14:00:36 -0800
Date: Tue, 6 Mar 2007 14:00:36 -0800
From: Andrew Morton <akpm@linux-foundation.org>
To: Pavel Emelianov <xemul@sw.ru>

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Message-Id: <20070306140036.4e85bd2f.akpm@linux-foundation.org>
In-Reply-To: <45ED80E1.7030406@sw.ru>
References: <45ED7DEC.7010403@sw.ru>
	<45ED80E1.7030406@sw.ru>
X-Mailer: Sylpheed version 2.2.7 (GTK+ 2.8.6; i686-pc-linux-gnu)
Mime-Version: 1.0
X-Spam-Status: No, hits=-1.453 required=5
+tests=AWL,OSDL_HEADER_LISTID_KNOWN,OSDL_HEADER_SUBJECT_BRACKETED
X-Spam-Checker-Version: SpamAssassin 2.63-osdl_revision__1.119__
X-MIMEDefang-Filter: osdl$Revision: 1.176 $
Cc: Kirill@smtp.osdl.org, Linux@smtp.osdl.org, containers@lists.osdl.org,
	Paul Menage <menage@google.com>,
	List <linux-kernel@vger.kernel.org>
X-BeenThere: containers@lists.osdl.org
X-Mailman-Version: 2.1.8
Precedence: list
List-Id: Linux Containers <containers.lists.osdl.org>
List-Unsubscribe: <https://lists.osdl.org/mailman/listinfo/containers>,
	<mailto:containers-request@lists.osdl.org?subject=unsubscribe>
List-Archive: <http://lists.osdl.org/pipermail/containers>
List-Post: <mailto:containers@lists.osdl.org>
List-Help: <mailto:containers-request@lists.osdl.org?subject=help>
List-Subscribe: <https://lists.osdl.org/mailman/listinfo/containers>,
	<mailto:containers-request@lists.osdl.org?subject=subscribe>
Content-Type: text/plain; charset="iso-8859-1"
Content-Transfer-Encoding: quoted-printable
Sender: containers-bounces@lists.osdl.org
Errors-To: containers-bounces@lists.osdl.org
Received-SPF: pass (localhost is always allowed.)
Status: RO
X-Status: A
Content-Length: 854
Lines: 27

-8<--

> > best,
> > Herbert
> >
>
> Cheers,
> Balbir
> _____
> Containers mailing list
> Containers@lists.osdl.org
> https://lists.osdl.org/mailman/listinfo/containers

_____

Subject: Re: [RFC][PATCH 1/7] Resource counters
Posted by Srivatsa Vaddagiri on Tue, 13 Mar 2007 16:07:07 GMT
View Forum Message <> Reply to Message

On Tue, Mar 13, 2007 at 06:41:05PM +0300, Pavel Emelianov wrote:
> > right, but atomic ops have much less impact on most
> > architectures than locks :)
>
> Right. But atomic_add_unless() is slower as it is
> essentially a loop. See my previous letter in this sub-thread.

If I am not mistaken, you shouldn't loop in normal cases, which means
it boils down to a atomic_read() + atomic_cmpxch()

--
Regards,
vatsa

Subject: Re: [RFC][PATCH 1/7] Resource counters
Posted by Herbert Poetzl on Tue, 13 Mar 2007 16:32:54 GMT
View Forum Message <> Reply to Message

On Tue, Mar 13, 2007 at 06:41:05PM +0300, Pavel Emelianov wrote:
> >>> PS: atomic_add_unless() didn't exist back then
> >>> (at least I think so) but that might be an option
> >>> too ...
> >> I think as far as having this discussion if you can remove that race
> >> people will be more willing to talk about what vserver does.
> >
> > well, shouldn't be a big deal to brush that patch up
> > (if somebody actually _is_ interested)
> >
> >> That said anything that uses locks or atomic operations (finer grained
> >> locks) because of the cache line ping pong is going to have scaling
> >> issues on large boxes.
> >

> > right, but atomic ops have much less impact on most
> > architectures than locks :)
>
> Right. But atomic_add_unless() is slower as it is
> essentially a loop. See my previous letter in this sub-thread.

fine, nobody actually uses atomic_add_unless(), or am I
missing something?

using two locks will be slower than using a single
lock, adding a loop which counts from 0 to 100 will
eat up some cpu, so what? don't do it :)

> >> So in that sense anything short of per cpu variables sucks at scale.
> >> That said I would much rather get a simple correct version without the
> >> complexity of per cpu counters, before we optimize the counters that
> >> much.
> >
> > actually I thought about per cpu counters quite a lot, and
> > we (Llinux-VServer) use them for accounting, but please
> > tell me how you use per cpu structures for implementing
> > limits
>
> Did you ever look at how get_empty_filp() works?
> I agree, that this is not a "strict" limit, but it
> limits the usage wit some "precision".
>
> /* off-the-topic */ Herbert, you've lost Balbir again:
> In this sub-thread some letters up Eric wrote a letter with
> Balbir in Cc:. The next reply from you doesn't include him.

I can happily add him to every email I reply to, but he
definitely isn't removed by my mailer (as I already stated,
it might be the mailing list which does this), fact is, the
email arrives here without him in the cc, so a reply does
not contain it either ...

best,
Herbert

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core

On Tue, 2007-03-13 at 03:48 -0800, Andrew Morton wrote:
> If we use a physical zone-based containment scheme: fake-numa,
> variable-sized zones, etc then it all becomes moot. You set up a container
> which has 1.5GB of physial memory then toss processes into it. As that
> process set increases in size it will toss out stray pages which shouldn't
> be there, then it will start reclaiming and swapping out its own pages and
> eventually it'll get an oom-killing.

I was just reading through the (comprehensive) thread about this from last week, so forgive me if I missed some of it. The idea is really tempting, precisely because I don't think anyone really wants to have to screw with the reclaim logic.

I'm just brain-dumping here, hoping that somebody has already thought through some of this stuff. It's not a bitch-fest, I promise. :)

How do we determine what is shared, and goes into the shared zones? Once we've allocated a page, it's too late because we already picked. Do we just assume all page cache is shared? Base it on filesystem, mount, ...? Mount seems the most logical to me, that a sysadmin would have to set up a container's fs, anyway, and will likely be doing special things to shared data, anyway (r/o bind mounts :).

There's a conflict between the resize granularity of the zones, and the storage space their lookup consumes. We'd want a container to have a limited ability to fill up memory with stuff like the dcache, so we'd appear to need to put the dentries inside the software zone. But, that gets us to our inability to evict arbitrary dentries. After a while, would containers tend to pin an otherwise empty zone into place? We could resize it, but what is the cost of keeping zones that can be resized down to a small enough size that we don't mind keeping it there? We could merge those "orphaned" zones back into the shared zone. Were there any requirements about physical contiguity? What about minimum zone sizes?

If we really do bind a set of processes strongly to a set of memory on a set of nodes, then those really do become its home NUMA nodes. If the CPUs there get overloaded, running it elsewhere will continue to grab pages from the home. Would this basically keep us from ever being able to move tasks around a NUMA system?

-- Dave

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Dave Hansen on Tue, 13 Mar 2007 17:26:33 GMT
View Forum Message <> Reply to Message

On Mon, 2007-03-12 at 22:04 -0800, Andrew Morton wrote:
> So these mmapped pages will contiue to be shared across all guests. The
> problem boils down to "which guest(s) get charged for each shared page".
>
> A simple and obvious and easy-to-implement answer is "the guest which paged
> it in". I think we should firstly explain why that is insufficient.

My first worry was that this approach is unfair to the poor bastard that happened to get started up first. If we have a bunch of containerized web servers, the poor guy who starts Apache first will pay the price for keeping it in memory for everybody else.

That said, I think this is naturally worked around. The guy charged unfairly will get reclaim started on himself sooner. This will tend to page out those pages that he was being unfairly charged for. Hopefully, they will eventually get pretty randomly (eventually evenly) spread among all users. We just might want to make sure that we don't allow ptes (or other new references) to be re-established to pages like this when we're trying to reclaim them. Either that, or force the next toucher to take ownership of the thing. But, that kind of arbitrary ownership transfer can't happen if we have rigidly defined boundaries for the containers.

The other concern is that the memory load on the system doesn't come from the first user ("the guy who paged it in"). The long-term load comes from "the guy who keeps using it." The best way to exemplify this is somebody who read()s a page in, followed by another guy mmap()ing the same page. The guy who did the read will get charged, and the mmap()er will get a free ride. We could probably get an idea when this kind of stuff is happening by comparing page->count and page->_mapcount, but it certainly wouldn't be conclusive. But, does this kind of nonsense even happen in practice?

-- Dave

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Alan Cox on Tue, 13 Mar 2007 19:09:31 GMT
View Forum Message <> Reply to Message

> stuff is happening by comparing page->count and page->_mapcount, but it
> certainly wouldn't be conclusive. But, does this kind of nonsense even
> happen in practice?

"Is it useful for me as a bad guy to make it happen ?"

Alan

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Dave Hansen on Tue, 13 Mar 2007 20:28:26 GMT
View Forum Message <> Reply to Message

On Tue, 2007-03-13 at 19:09 +0000, Alan Cox wrote:
> > stuff is happening by comparing page->count and page->_mapcount, but it
> > certainly wouldn't be conclusive. But, does this kind of nonsense even
> > happen in practice?
>
> "Is it useful for me as a bad guy to make it happen ?"

A very fine question. ;)

To exploit this, you'd need to:
1. need to access common data with another user
2. be patient enough to wait
3. determine when one of those users had actually pulled
   a page in from disk, which sys_mincore() can do, right?

I guess that might be a decent reason to not charge the guy who brings
the page in for the page's entire lifetime.

So, unless we can change page ownership after it has been allocated,
anyone accessing shared data can get around resource limits if they are
patient.

-- Dave

_____
Containers mailing list
Containers@lists.osdl.org

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by Nick Piggin on Wed, 14 Mar 2007 03:51:37 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Nick Piggin <nickpiggin@yahoo.com.au> writes:
>
>
>>Eric W. Biederman wrote:
>>
>>>First touch page ownership does not guarantee give me anything useful
>>>for knowing if I can run my application or not. Because of page
>>>sharing my application might run inside the rss limit only because
>>>I got lucky and happened to share a lot of pages with another running
>>>application. If the next I run and it isn't running my application
>>>will fail. That is ridiculous.
>>
>>Let's be practical here, what you're asking is basically impossible.
>>
>>Unless by deterministic you mean that it never enters the a non
>>trivial syscall, in which case, you just want to know about maximum
>>RSS of the process, which we already account).
>
>
> Not per process I want this on a group of processes, and yes that
> is all I want just. I just want accounting of the maximum RSS of
> a group of processes and then the mechanism to limit that maximum rss.

Well don't you just sum up the maximum for each process?

Or do you want to only count shared pages inside a container once,
or something difficult like that?

>>>I don't want sharing between vservers/VE/containers to affect how many
>>>pages I can have mapped into my processes at once.
>>
>>You seem to want total isolation. You could use virtualization?
>
>
> No. I don't want the meaning of my rss limit to be affected by what
> other processes are doing. We have constraints of how many resources
> the box actually has. But I don't want accounting so sloppy that
> processes outside my group of processes can artificially
> lower my rss value, which magically raises my rss limit.

So what are you going to do about all the shared caches and slabs
inside the kernel?

>>It is basically handwaving anyway. The only approach I've seen with
>>a sane (not perfect, but good) way of accounting memory use is this
>>one. If you care to define "proper", then we could discuss that.
>
>
> I will agree that this patchset is probably in the right general ballpark.
> But the fact that pages are assigned exactly one owner is pure non-sense.
> We can do better. That is all I am asking for someone to at least attempt
> to actually account for the rss of a group of processes and get the numbers
> right when we have shared pages, between different groups of
> processes. We have the data structures to support this with rmap.

Well rmap only supports mapped, userspace pages.

> Let me describe the situation where I think the accounting in the
> patchset goes totally wonky.
>
>
> Gcc as I recall maps the pages it is compiling with mmap.
> If in a single kernel tree I do:
> make -jN O=../compile1 &
> make -jN O=../compile2 &
>
> But set it up so that the two compiles are in different rss groups.
> If I run the concurrently they will use the same files at the same
> time and most likely because of the first touch rss limit rule even
> if I have a draconian rss limit the compiles will both be able to
> complete and finish. However if I run either of them alone if I
> use the most draconian rss limit I can that allows both compiles to
> finish I won't be able to compile a single kernel tree.

Yeah it is not perfect. Fortunately, there is no perfect solution,
so we don't have to be too upset about that.

And strangely, this example does not go outside the parameters of
what you asked for AFAIKS. In the worst case of one container getting
all the shared pages, they will still remain inside their maximum
rss limit.

So they might get penalised a bit on reclaim, but maximum rss limits
will work fine, and you can (almost) guarantee X amount of memory for
a given container, and it will _work_.

But I also take back my comments about this being the only design I
have seen that gets everything, because the node-per-container idea
is a really good one on the surface. And it could mean even less impact
on the core VM than this patch. That is also a first-touch scheme.

> However the messed up accounting that doesn't handle sharing between
> groups of processes properly really bugs me. Especially when we have
> the infrastructure to do it right.
>
> Does that make more sense?

I think it is simplistic.

Sure you could probably use some of the rmap stuff to account shared
mapped _user_ pages once for each container that touches them. And
this patchset isn't preventing that.

But how do you account kernel allocations? How do you account unmapped
pagecache?

What's the big deal so many accounting people have with just RSS? I'm
not a container person, this is an honest question. Because from my
POV if you conveniently ignore everything else... you may as well just
not do any accounting at all.

--
SUSE Labs, Novell Inc.
Send instant messages to your online friends http://au.messenger.yahoo.com

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by Balbir Singh on Wed, 14 Mar 2007 06:42:06 GMT
View Forum Message <> Reply to Message

Nick Piggin wrote:
> Eric W. Biederman wrote:
>> Nick Piggin <nickpiggin@yahoo.com.au> writes:
>>
>>
>>> Eric W. Biederman wrote:
>>>

>>>> First touch page ownership does not guarantee give me anything useful
>>>> for knowing if I can run my application or not. Because of page
>>>> sharing my application might run inside the rss limit only because
>>>> I got lucky and happened to share a lot of pages with another running
>>>> application. If the next I run and it isn't running my application
>>>> will fail. That is ridiculous.
>>>
>>> Let's be practical here, what you're asking is basically impossible.
>>>
>>> Unless by deterministic you mean that it never enters the a non
>>> trivial syscall, in which case, you just want to know about maximum
>>> RSS of the process, which we already account).
>>
>>
>> Not per process I want this on a group of processes, and yes that
>> is all I want just. I just want accounting of the maximum RSS of
>> a group of processes and then the mechanism to limit that maximum rss.
>
> Well don't you just sum up the maximum for each process?
>
> Or do you want to only count shared pages inside a container once,
> or something difficult like that?
>
>
>>>> I don't want sharing between vservers/VE/containers to affect how many
>>>> pages I can have mapped into my processes at once.
>>>
>>> You seem to want total isolation. You could use virtualization?
>>
>>
>> No. I don't want the meaning of my rss limit to be affected by what
>> other processes are doing. We have constraints of how many resources
>> the box actually has. But I don't want accounting so sloppy that
>> processes outside my group of processes can artificially
>> lower my rss value, which magically raises my rss limit.
>
> So what are you going to do about all the shared caches and slabs
> inside the kernel?
>
>
>>> It is basically handwaving anyway. The only approach I've seen with
>>> a sane (not perfect, but good) way of accounting memory use is this
>>> one. If you care to define "proper", then we could discuss that.
>>
>>
>> I will agree that this patchset is probably in the right general
>> ballpark.
>> But the fact that pages are assigned exactly one owner is pure non-sense.

>> We can do better. That is all I am asking for someone to at least
>> attempt
>> to actually account for the rss of a group of processes and get the
>> numbers
>> right when we have shared pages, between different groups of
>> processes. We have the data structures to support this with rmap.
>
> Well rmap only supports mapped, userspace pages.
>
>
>> Let me describe the situation where I think the accounting in the
>> patchset goes totally wonky.
>>
>> Gcc as I recall maps the pages it is compiling with mmap.
>> If in a single kernel tree I do:
>> make -jN O=../compile1 &
>> make -jN O=../compile2 &
>>
>> But set it up so that the two compiles are in different rss groups.
>> If I run the concurrently they will use the same files at the same
>> time and most likely because of the first touch rss limit rule even
>> if I have a draconian rss limit the compiles will both be able to
>> complete and finish.   However if I run either of them alone if I
>> use the most draconian rss limit I can that allows both compiles to
>> finish I won't be able to compile a single kernel tree.
>
> Yeah it is not perfect. Fortunately, there is no perfect solution,
> so we don't have to be too upset about that.
>
> And strangely, this example does not go outside the parameters of
> what you asked for AFAIKS. In the worst case of one container getting
> _all_ the shared pages, they will still remain inside their maximum
> rss limit.
>

When that does happen and if a container hits it limit, with a LRU
per-container, if the container is not actually using those pages,
they'll get thrown out of that container and get mapped into the
container that is using those pages most frequently.

> So they might get penalised a bit on reclaim, but maximum rss limits
> will work fine, and you can (almost) guarantee X amount of memory for
> a given container, and it will _work_.
>
> But I also take back my comments about this being the only design I
> have seen that gets everything, because the node-per-container idea
> is a really good one on the surface. And it could mean even less impact
> on the core VM than this patch. That is also a first-touch scheme.

>

With the proposed node-per-container, we will need to make massive core
VM changes to reorganize zones and nodes. We would want to allow

1. For sharing of nodes
2. Resizing nodes
3. May be more

With the node-per-container idea, it will hard to control page cache
limits, independent of RSS limits or mlock limits.

NOTE: page cache == unmapped page cache here.

>
>> However the messed up accounting that doesn't handle sharing between
>> groups of processes properly really bugs me. Especially when we have
>> the infrastructure to do it right.
>>
>> Does that make more sense?
>
> I think it is simplistic.
>
> Sure you could probably use some of the rmap stuff to account shared
> mapped _user_ pages once for each container that touches them. And
> this patchset isn't preventing that.
>
> But how do you account kernel allocations? How do you account unmapped
> pagecache?
>
> What's the big deal so many accounting people have with just RSS? I'm
> not a container person, this is an honest question. Because from my
> POV if you conveniently ignore everything else... you may as well just
> not do any accounting at all.
>

We decided to implement accounting and control in phases

1. RSS control
2. unmapped page cache control
3. mlock control
4. Kernel accounting and limits

This has several advantages

1. The limits can be individually set and controlled.
2. The code is broken down into simpler chunks for review and merging.

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

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by Nick Piggin on Wed, 14 Mar 2007 06:57:55 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Nick Piggin wrote:
>
>> And strangely, this example does not go outside the parameters of
>> what you asked for AFAIKS. In the worst case of one container getting
>> _all_ the shared pages, they will still remain inside their maximum
>> rss limit.
>>
>
> When that does happen and if a container hits it limit, with a LRU
> per-container, if the container is not actually using those pages,
> they'll get thrown out of that container and get mapped into the
> container that is using those pages most frequently.

Exactly. Statistically, first touch will work OK. It may mean some
reclaim inefficiencies in corner cases, but things will tend to
even out.

>> So they might get penalised a bit on reclaim, but maximum rss limits
>> will work fine, and you can (almost) guarantee X amount of memory for
>> a given container, and it will _work_.
>>
>> But I also take back my comments about this being the only design I
>> have seen that gets everything, because the node-per-container idea
>> is a really good one on the surface. And it could mean even less impact
>> on the core VM than this patch. That is also a first-touch scheme.
>>
>
> With the proposed node-per-container, we will need to make massive core
> VM changes to reorganize zones and nodes. We would want to allow
>
> 1. For sharing of nodes
> 2. Resizing nodes

> 3. May be more

But a lot of that is happening anyway for other reasons (eg. memory
plug/unplug). And I don't consider node/zone setup to be part of the
"core VM" as such... it is _good_ if we can move extra work into setup
rather than have it in the mm.

That said, I don't think this patch is terribly intrusive either.

> With the node-per-container idea, it will hard to control page cache
> limits, independent of RSS limits or mlock limits.
>
> NOTE: page cache == unmapped page cache here.

I don't know that it would be particularly harder than any other
first-touch scheme. If one container ends up being charged with too
much pagecache, eventually they'll reclaim a bit of it and the pages
will get charged to more frequent users.

>>> However the messed up accounting that doesn't handle sharing between
>>> groups of processes properly really bugs me. Especially when we have
>>> the infrastructure to do it right.
>>>
>>> Does that make more sense?
>>
>>
>> I think it is simplistic.
>>
>> Sure you could probably use some of the rmap stuff to account shared
>> mapped _user_ pages once for each container that touches them. And
>> this patchset isn't preventing that.
>>
>> But how do you account kernel allocations? How do you account unmapped
>> pagecache?
>>
>> What's the big deal so many accounting people have with just RSS? I'm
>> not a container person, this is an honest question. Because from my
>> POV if you conveniently ignore everything else... you may as well just
>> not do any accounting at all.
>>
>
> We decided to implement accounting and control in phases
>
> 1. RSS control
> 2. unmapped page cache control
> 3. mlock control

> 4. Kernel accounting and limits
>
> This has several advantages
>
> 1. The limits can be individually set and controlled.
> 2. The code is broken down into simpler chunks for review and merging.

But this patch gives the groundwork to handle 1-4, and it is in a small
chunk, and one would be able to apply different limits to different types
of pages with it. Just using rmap to handle 1 does not really seem like a
viable alternative because it fundamentally isn't going to handle 2 or 4.

I'm not saying that you couldn't _later_ add something that uses rmap or
our current RSS accounting to tweak container-RSS semantics. But isn't it
sensible to lay the groundwork first? Get a clear path to something that
is good (not perfect), but *works*?

--
SUSE Labs, Novell Inc.

Subject: Re: [RFC][PATCH 1/7] Resource counters
Posted by xemul on Wed, 14 Mar 2007 07:12:11 GMT
View Forum Message <> Reply to Message

Srivatsa Vaddagiri wrote:
> On Tue, Mar 13, 2007 at 06:41:05PM +0300, Pavel Emelianov wrote:
>>> right, but atomic ops have much less impact on most
>>> architectures than locks :)
>> Right. But atomic_add_unless() is slower as it is
>> essentially a loop. See my previous letter in this sub-thread.
>
> If I am not mistaken, you shouldn't loop in normal cases, which means
> it boils down to a atomic_read() + atomic_cmpxch()
>
>

So does the lock - in a normal case (when it's not
heavily contented) it will boil down to atomic_dec_and_test().

Nevertheless, making charge like in this patchset
requires two atomic ops with atomic_xxx and only

one with spin_lock().

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by Balbir Singh on Wed, 14 Mar 2007 07:48:35 GMT
View Forum Message <> Reply to Message

Nick Piggin wrote:
> Balbir Singh wrote:
>> Nick Piggin wrote:
>
>>> And strangely, this example does not go outside the parameters of
>>> what you asked for AFAIKS. In the worst case of one container getting
>>> _all_ the shared pages, they will still remain inside their maximum
>>> rss limit.
>>>
>>
>> When that does happen and if a container hits it limit, with a LRU
>> per-container, if the container is not actually using those pages,
>> they'll get thrown out of that container and get mapped into the
>> container that is using those pages most frequently.
>
> Exactly. Statistically, first touch will work OK. It may mean some
> reclaim inefficiencies in corner cases, but things will tend to
> even out.
>

Exactly!

>>> So they might get penalised a bit on reclaim, but maximum rss limits
>>> will work fine, and you can (almost) guarantee X amount of memory for
>>> a given container, and it will _work_.
>>>
>>> But I also take back my comments about this being the only design I
>>> have seen that gets everything, because the node-per-container idea
>>> is a really good one on the surface. And it could mean even less impact
>>> on the core VM than this patch. That is also a first-touch scheme.
>>>
>>
>> With the proposed node-per-container, we will need to make massive core
>> VM changes to reorganize zones and nodes. We would want to allow
>>
>> 1. For sharing of nodes
>> 2. Resizing nodes

>> 3. May be more
>
> But a lot of that is happening anyway for other reasons (eg. memory
> plug/unplug). And I don't consider node/zone setup to be part of the
> "core VM" as such... it is _good_ if we can move extra work into setup
> rather than have it in the mm.
>
> That said, I don't think this patch is terribly intrusive either.
>

Thanks, thats one of our goals, to keep it simple, understandable and
non-intrusive.

>
>> With the node-per-container idea, it will hard to control page cache
>> limits, independent of RSS limits or mlock limits.
>>
>> NOTE: page cache == unmapped page cache here.
>
> I don't know that it would be particularly harder than any other
> first-touch scheme. If one container ends up being charged with too
> much pagecache, eventually they'll reclaim a bit of it and the pages
> will get charged to more frequent users.
>
>

Yes, true, but what if a user does not want to control the page
cache usage in a particular container or wants to turn off
RSS control.

>>>> However the messed up accounting that doesn't handle sharing between
>>>> groups of processes properly really bugs me. Especially when we have
>>>> the infrastructure to do it right.
>>>>
>>>> Does that make more sense?
>>>
>>>
>>> I think it is simplistic.
>>>
>>> Sure you could probably use some of the rmap stuff to account shared
>>> mapped _user_ pages once for each container that touches them. And
>>> this patchset isn't preventing that.
>>>
>>> But how do you account kernel allocations? How do you account unmapped
>>> pagecache?
>>>
>>> What's the big deal so many accounting people have with just RSS? I'm
>>> not a container person, this is an honest question. Because from my

>>> POV if you conveniently ignore everything else... you may as well just
>>> not do any accounting at all.
>>>
>>
>> We decided to implement accounting and control in phases
>>
>> 1. RSS control
>> 2. unmapped page cache control
>> 3. mlock control
>> 4. Kernel accounting and limits
>>
>> This has several advantages
>>
>> 1. The limits can be individually set and controlled.
>> 2. The code is broken down into simpler chunks for review and merging.
>
> But this patch gives the groundwork to handle 1-4, and it is in a small
> chunk, and one would be able to apply different limits to different types
> of pages with it. Just using rmap to handle 1 does not really seem like a
> viable alternative because it fundamentally isn't going to handle 2 or 4.
>

For (2), we have the basic setup in the form of a per-container LRU list
and a pointer from struct page to the container that first brought in
the page.

> I'm not saying that you couldn't _later_ add something that uses rmap or
> our current RSS accounting to tweak container-RSS semantics. But isn't it
> sensible to lay the groundwork first? Get a clear path to something that
> is good (not perfect), but *works*?
>

I agree with your development model suggestion. One of things we are going
to do in the near future is to build (2) and then add (3) and (4). So far,
we've not encountered any difficulties on building on top of (1).

Vaidy, any comments?

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

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by Vaidyanathan Srinivas on Wed, 14 Mar 2007 13:25:08 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Nick Piggin wrote:
>> Balbir Singh wrote:
>>> Nick Piggin wrote:
>>>> And strangely, this example does not go outside the parameters of
>>>> what you asked for AFAIKS. In the worst case of one container getting
>>>> _all_ the shared pages, they will still remain inside their maximum
>>>> rss limit.
>>>>
>>> When that does happen and if a container hits it limit, with a LRU
>>> per-container, if the container is not actually using those pages,
>>> they'll get thrown out of that container and get mapped into the
>>> container that is using those pages most frequently.
>> Exactly. Statistically, first touch will work OK. It may mean some
>> reclaim inefficiencies in corner cases, but things will tend to
>> even out.
>>
>
> Exactly!
>
>>>> So they might get penalised a bit on reclaim, but maximum rss limits
>>>> will work fine, and you can (almost) guarantee X amount of memory for
>>>> a given container, and it will _work_.
>>>>
>>>> But I also take back my comments about this being the only design I
>>>> have seen that gets everything, because the node-per-container idea
>>>> is a really good one on the surface. And it could mean even less impact
>>>> on the core VM than this patch. That is also a first-touch scheme.
>>>>
>>> With the proposed node-per-container, we will need to make massive core
>>> VM changes to reorganize zones and nodes. We would want to allow
>>>
>>> 1. For sharing of nodes
>>> 2. Resizing nodes
>>> 3. May be more
>> But a lot of that is happening anyway for other reasons (eg. memory
>> plug/unplug). And I don't consider node/zone setup to be part of the
>> "core VM" as such... it is _good_ if we can move extra work into setup
>> rather than have it in the mm.
>>
>> That said, I don't think this patch is terribly intrusive either.
>>
>
> Thanks, thats one of our goals, to keep it simple, understandable and
> non-intrusive.

>
>>> With the node-per-container idea, it will hard to control page cache
>>> limits, independent of RSS limits or mlock limits.
>>>
>>> NOTE: page cache == unmapped page cache here.
>> I don't know that it would be particularly harder than any other
>> first-touch scheme. If one container ends up being charged with too
>> much pagecache, eventually they'll reclaim a bit of it and the pages
>> will get charged to more frequent users.
>>
>>
>
> Yes, true, but what if a user does not want to control the page
> cache usage in a particular container or wants to turn off
> RSS control.
>
>>>>> However the messed up accounting that doesn't handle sharing between
>>>>> groups of processes properly really bugs me. Especially when we have
>>>>> the infrastructure to do it right.
>>>>>
>>>>> Does that make more sense?
>>>>
>>>> I think it is simplistic.
>>>>
>>>> Sure you could probably use some of the rmap stuff to account shared
>>>> mapped _user_ pages once for each container that touches them. And
>>>> this patchset isn't preventing that.
>>>>
>>>> But how do you account kernel allocations? How do you account unmapped
>>>> pagecache?
>>>>
>>>> What's the big deal so many accounting people have with just RSS? I'm
>>>> not a container person, this is an honest question. Because from my
>>>> POV if you conveniently ignore everything else... you may as well just
>>>> not do any accounting at all.
>>>>
>>> We decided to implement accounting and control in phases
>>>
>>> 1. RSS control
>>> 2. unmapped page cache control
>>> 3. mlock control
>>> 4. Kernel accounting and limits
>>>
>>> This has several advantages
>>>
>>> 1. The limits can be individually set and controlled.
>>> 2. The code is broken down into simpler chunks for review and merging.
>> But this patch gives the groundwork to handle 1-4, and it is in a small

>> chunk, and one would be able to apply different limits to different types
>> of pages with it. Just using rmap to handle 1 does not really seem like a
>> viable alternative because it fundamentally isn't going to handle 2 or 4.
>>
>
> For (2), we have the basic setup in the form of a per-container LRU list
> and a pointer from struct page to the container that first brought in
> the page.
>
>> I'm not saying that you couldn't _later_ add something that uses rmap or
>> our current RSS accounting to tweak container-RSS semantics. But isn't it
>> sensible to lay the groundwork first? Get a clear path to something that
>> is good (not perfect), but *works*?
>>
>
> I agree with your development model suggestion. One of things we are going
> to do in the near future is to build (2) and then add (3) and (4). So far,
> we've not encountered any difficulties on building on top of (1).
>
> Vaidy, any comments?

Accounting becomes easy if we have a container pointer in struct page.
This can form base ground for building controllers since any memory related controller would be interested in tracking pages. However we still want to evaluate if we can build them without bloating the struct page. Pagecache controller (2) we can implement with container pointer in struct page or container pointer in struct address space.

Building on this patchset is much simple and and we hope the bloat in struct page will be compensated by the benefits in memory controllers in terms of performance and simplicity.

Adding too many controllers and accounting parameters to start with will make the patch too big and complex. As Balbir mentioned, we have a plan and we shall add new control parameters in stages.

--Vaidy

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by Nick Piggin on Wed, 14 Mar 2007 13:49:07 GMT
View Forum Message <> Reply to Message

Vaidyanathan Srinivasan wrote:

> Accounting becomes easy if we have a container pointer in struct page.
> This can form base ground for building controllers since any memory
> related controller would be interested in tracking pages. However we
> still want to evaluate if we can build them without bloating the
> struct page. Pagecache controller (2) we can implement with container
> pointer in struct page or container pointer in struct address space.

The thing is, you have to worry about actually getting anything in the
kernel rather than trying to do fancy stuff.

The approaches I have seen that don't have a struct page pointer, do
intrusive things like try to put hooks everywhere throughout the kernel
where a userspace task can cause an allocation (and of course end up
missing many, so they aren't secure anyway)... and basically just
nasty stuff that will never get merged.

Struct page overhead really isn't bad. Sure, nobody who doesn't use
containers will want to turn it on, but unless you're using a big PAE
system you're actually unlikely to notice.

But again, I'll say the node-container approach of course does avoid
this nicely (because we already can get the node from the page). So
definitely that approach needs to be discredited before going with this
one.

> Building on this patchset is much simple and and we hope the bloat in
> struct page will be compensated by the benefits in memory controllers
> in terms of performance and simplicity.
>
> Adding too many controllers and accounting parameters to start with
> will make the patch too big and complex. As Balbir mentioned, we have
> a plan and we shall add new control parameters in stages.

Everyone seems to have a plan ;) I don't read the containers list...
does everyone still have *different* plans, or is any sort of consensus
being reached?

--
SUSE Labs, Novell Inc.
Send instant messages to your online friends http://au.messenger.yahoo.com

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by Vaidyanathan Srinivas on Wed, 14 Mar 2007 14:43:12 GMT
View Forum Message <> Reply to Message

Nick Piggin wrote:
> Vaidyanathan Srinivasan wrote:
>
>> Accounting becomes easy if we have a container pointer in struct page.
>> This can form base ground for building controllers since any memory
>> related controller would be interested in tracking pages. However we
>> still want to evaluate if we can build them without bloating the
>> struct page. Pagecache controller (2) we can implement with container
>> pointer in struct page or container pointer in struct address space.
>
> The thing is, you have to worry about actually getting anything in the
> kernel rather than trying to do fancy stuff.
>
> The approaches I have seen that don't have a struct page pointer, do
> intrusive things like try to put hooks everywhere throughout the kernel
> where a userspace task can cause an allocation (and of course end up
> missing many, so they aren't secure anyway)... and basically just
> nasty stuff that will never get merged.
>
> Struct page overhead really isn't bad. Sure, nobody who doesn't use
> containers will want to turn it on, but unless you're using a big PAE
> system you're actually unlikely to notice.
>
> But again, I'll say the node-container approach of course does avoid
> this nicely (because we already can get the node from the page). So
> definitely that approach needs to be discredited before going with this
> one.

I agree :)

>> Building on this patchset is much simple and and we hope the bloat in
>> struct page will be compensated by the benefits in memory controllers
>> in terms of performance and simplicity.
>>
>> Adding too many controllers and accounting parameters to start with
>> will make the patch too big and complex. As Balbir mentioned, we have
>> a plan and we shall add new control parameters in stages.
>
> Everyone seems to have a plan ;) I don't read the containers list...
> does everyone still have *different* plans, or is any sort of consensus
> being reached?

Consensus? I believe at this point we have a sort of consensus on the
base container infrastructure and the need for memory controller to
control RSS, pagecache, mlock, kernel memory etc. However the

implementation and approach taken is still being discussed :)

--Vaidy

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by Cedric Le Goater on Wed, 14 Mar 2007 15:37:59 GMT
View Forum Message <> Reply to Message

> --- linux-2.6.20.orig/mm/migrate.c 2007-02-04 21:44:54.000000000 +0300
> +++ linux-2.6.20-0/mm/migrate.c 2007-03-06 13:33:28.000000000 +0300
> @@ -134,6 +134,7 @@ static void remove_migration_pte(struct
> pte_t *ptep, pte;
> spinlock_t *ptl;
> unsigned long addr = page_address_in_vma(new, vma);
> + struct page_container *pcont;
>
> if (addr == -EFAULT)
> return;
> @@ -157,6 +158,11 @@ static void remove_migration_pte(struct
> return;
> }
>
> + if (container_rss_prepare(new, vma, &pcont)) {
> + pte_unmap(ptep);
> + return;
> + }
> +
> ptl = pte_lockptr(mm, pmd);
> spin_lock(ptl);
> pte = *ptep;
> @@ -175,16 +181,19 @@ static void remove_migration_pte(struct
> set_pte_at(mm, addr, ptep, pte);
>
> if (PageAnon(new))
> - page_add_anon_rmap(new, vma, addr);
> + page_add_anon_rmap(new, vma, addr, pcont);
> else
> - page_add_file_rmap(new);
> + page_add_file_rmap(new, pcont);
>
> /* No need to invalidate - it was non-present before */
> update_mmu_cache(vma, addr, pte);

> lazy_mmu_prot_update(pte);
> + pte_unmap_unlock(ptep, ptl);
> + return;
>
> out:
> pte_unmap_unlock(ptep, ptl);
> + container_rss_release(pcont);
> }
>
> /*

you missed out an include in mm/migrate.c

cheers,

C.
Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
 mm/migrate.c | 1 +
 1 file changed, 1 insertion(+)

Index: 2.6.20/mm/migrate.c
 ========== =======
--- 2.6.20.orig/mm/migrate.c
+++ 2.6.20/mm/migrate.c
@@ -28,6 +28,7 @@
 #include <linux/mempolicy.h>
 #include <linux/vmalloc.h>
 #include <linux/security.h>
+#include <linux/rss_container.h>

 #include "internal.h"

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by mel on Wed, 14 Mar 2007 15:38:24 GMT
View Forum Message <> Reply to Message

On (13/03/07 10:05), Dave Hansen didst pronounce:
> On Tue, 2007-03-13 at 03:48 -0800, Andrew Morton wrote:
> > If we use a physical zone-based containment scheme: fake-numa,
> > variable-sized zones, etc then it all becomes moot. You set up a container
> > which has 1.5GB of physial memory then toss processes into it. As that
> > process set increases in size it will toss out stray pages which shouldn't
> > be there, then it will start reclaiming and swapping out its own pages and
> > eventually it'll get an oom-killing.
>
> I was just reading through the (comprehensive) thread about this from

> last week, so forgive me if I missed some of it. The idea is really
> tempting, precisely because I don't think anyone really wants to have to
> screw with the reclaim logic.
>
> I'm just brain-dumping here, hoping that somebody has already thought
> through some of this stuff. It's not a bitch-fest, I promise. :)
>
> How do we determine what is shared, and goes into the shared zones?

Assuming we had a means of creating a zone that was assigned to a container,
a second zone for shared data between a set of containers. For shared data,
the time the pages are being allocated is at page fault time. At that point,
the faulting VMA is known and you also know if it's MAP_SHARED or not.

The caller allocating the page would select (or create) a zonelist that
is appropriate for the container. For shared mappings, it would be one
zone - the shared zone for the set. For private mappings, it would be
one zone - the shared zone for the set.

For overcommit, the allowable zones for overcommit could be included.
Allowing overcommit opens the possibility for containers to interfere with
each other but I'm guessing that if overcommit is enabled, the administrator
is willing to live with that interference.

This has the awkward possibility of having two "shared" zones for two container
sets and one file that needs sharing. Similarly, there is a possibility for
having a container that has no shared zone and faulted in shared data. In
that case, the page ends up in the first faulting container set and it's
too bad it got "charged" for the page use on behalf of other containers. I'm
not sure there is a sane way of accounting this situation fairly.

I think that it's important to note that once data is shared between containers
at all that they have the potential to interfere with each other (by reclaiming
within the shared zone for example).

> Once we've allocated a page, it's too late because we already picked.

We'd choose the appropriate zonelist before faulting. Once allocated,
the page stays there.

> Do we just assume all page cache is shared? Base it on filesystem,
> mount, ...? Mount seems the most logical to me, that a sysadmin would
> have to set up a container's fs, anyway, and will likely be doing
> special things to shared data, anyway (r/o bind mounts :).
>

I have no strong feelings here. To me, it's "who do I assign this fake
zone to?" I guess you would have at least one zone per container mount

for private data.

> There's a conflict between the resize granularity of the zones, and the
> storage space their lookup consumes. We'd want a container to have a
> limited ability to fill up memory with stuff like the dcache, so we'd
> appear to need to put the dentries inside the software zone. But, that
> gets us to our inability to evict arbitrary dentries.

Stuff like shrinking dentry caches is already pretty course-grained.
Last I looked, we couldn't even shrink within a specific node, let alone
a zone or a specific dentry. This is a separate problem.

> After a while,
> would containers tend to pin an otherwise empty zone into place? We
> could resize it, but what is the cost of keeping zones that can be
> resized down to a small enough size that we don't mind keeping it there?
> We could merge those "orphaned" zones back into the shared zone.

Merging "orphaned" zones back into the "main" zone would seem a sensible
choice.

> Were there any requirements about physical contiguity?

For the lookup to software zone to be efficient, it would be easiest to have
them as MAX_ORDER_NR_PAGES contiguous. This would avoid having to break the
existing assumptions in the buddy allocator about MAX_ORDER_NR_PAGES
always being in the same zone.

> What about minimum
> zone sizes?
>

MAX_ORDER_NR_PAGES would be the minimum zone size.

> If we really do bind a set of processes strongly to a set of memory on a
> set of nodes, then those really do become its home NUMA nodes. If the
> CPUs there get overloaded, running it elsewhere will continue to grab
> pages from the home. Would this basically keep us from ever being able
> to move tasks around a NUMA system?
>

Moving the tasks around would not be easy. It would require a new zone
to be created based on the new NUMA node and all the data migrated. hmm

--
Mel Gorman
Part-time Phd Student Linux Technology Center
University of Limerick IBM Dublin Software Lab

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by xemul on Wed, 14 Mar 2007 15:43:37 GMT
View Forum Message <> Reply to Message

Cedric Le Goater wrote:
>> --- linux-2.6.20.orig/mm/migrate.c 2007-02-04 21:44:54.000000000 +0300
>> +++ linux-2.6.20-0/mm/migrate.c 2007-03-06 13:33:28.000000000 +0300
>> @@ -134,6 +134,7 @@ static void remove_migration_pte(struct
>> pte_t *ptep, pte;
>> spinlock_t *ptl;
>> unsigned long addr = page_address_in_vma(new, vma);
>> + struct page_container *pcont;
>>
>> if (addr == -EFAULT)
>> return;
>> @@ -157,6 +158,11 @@ static void remove_migration_pte(struct
>> return;
>> }
>>
>> + if (container_rss_prepare(new, vma, &pcont)) {
>> + pte_unmap(ptep);
>> + return;
>> + }
>> +
>> ptl = pte_lockptr(mm, pmd);
>> spin_lock(ptl);
>> pte = *ptep;
>> @@ -175,16 +181,19 @@ static void remove_migration_pte(struct
>> set_pte_at(mm, addr, ptep, pte);
>>
>> if (PageAnon(new))
>> - page_add_anon_rmap(new, vma, addr);
>> + page_add_anon_rmap(new, vma, addr, pcont);
>> else
>> - page_add_file_rmap(new);
>> + page_add_file_rmap(new, pcont);
>>
>> /* No need to invalidate - it was non-present before */
>> update_mmu_cache(vma, addr, pte);
>> lazy_mmu_prot_update(pte);
>> + pte_unmap_unlock(ptep, ptl);
>> + return;

>>
>> out:
>>   pte_unmap_unlock(ptep, ptl);
>> + container_rss_release(pcont);
>> }
>>
>> /*
>
> you missed out an include in mm/migrate.c
>
> cheers,

Thanks! :)

> C.
> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
> ---
> mm/migrate.c | 1 +
> 1 file changed, 1 insertion(+)
>
> Index: 2.6.20/mm/migrate.c
> ========== =======
> --- 2.6.20.orig/mm/migrate.c
> +++ 2.6.20/mm/migrate.c
> @@ -28,6 +28,7 @@
> #include <linux/mempolicy.h>
> #include <linux/vmalloc.h>
> #include <linux/security.h>
> +#include <linux/rss_container.h>
>
> #include "internal.h"
>
> -
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/
>

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by dev on Wed, 14 Mar 2007 16:16:26 GMT
View Forum Message <> Reply to Message

Nick,

>>Accounting becomes easy if we have a container pointer in struct page.
>> This can form base ground for building controllers since any memory

>>related controller would be interested in tracking pages. However we
>>still want to evaluate if we can build them without bloating the
>>struct page. Pagecache controller (2) we can implement with container
>>pointer in struct page or container pointer in struct address space.
>
>
> The thing is, you have to worry about actually getting anything in the
> kernel rather than trying to do fancy stuff.
>
> The approaches I have seen that don't have a struct page pointer, do
> intrusive things like try to put hooks everywhere throughout the kernel
> where a userspace task can cause an allocation (and of course end up
> missing many, so they aren't secure anyway)... and basically just
> nasty stuff that will never get merged.

User beancounters patch has got through all these...
The approach where each charged object has a pointer to the owner container,
who has charged it - is the most easy/clean way to handle
all the problems with dynamic context change, races, etc.
and 1 pointer in page struct is just 0.1% overehad.

> Struct page overhead really isn't bad. Sure, nobody who doesn't use
> containers will want to turn it on, but unless you're using a big PAE
> system you're actually unlikely to notice.

big PAE doesn't make any difference IMHO
(until struct pages are not created for non-present physical memory areas)

> But again, I'll say the node-container approach of course does avoid
> this nicely (because we already can get the node from the page). So
> definitely that approach needs to be discredited before going with this
> one.

But it lacks some other features:
1. page can't be shared easily with another container
2. shared page can't be accounted honestly to containers
   as fraction=PAGE_SIZE/containers-using-it
3. It doesn't help accounting of kernel memory structures.
   e.g. in OpenVZ we use exactly the same pointer on the page
   to track which container owns it, e.g. pages used for page
   tables are accounted this way.
4. I guess container destroy requires destroy of memory zone,
   which means write out of dirty data. Which doesn't sound
   good for me as well.
5. memory reclamation in case of global memory shortage
   becomes a tricky/unfair task.
6. You cannot overcommit. AFAIU, the memory should be granted
   to node exclusive usage and cannot be used by by another containers,

even if it is unused. This is not an option for us.

>>Building on this patchset is much simple and and we hope the bloat in
>>struct page will be compensated by the benefits in memory controllers
>>in terms of performance and simplicity.
>>
>>Adding too many controllers and accounting parameters to start with
>>will make the patch too big and complex. As Balbir mentioned, we have
>>a plan and we shall add new control parameters in stages.
>
> Everyone seems to have a plan ;) I don't read the containers list...
> does everyone still have *different* plans, or is any sort of consensus
> being reached?

hope we'll have it soon :)

Thanks,
Kirill

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by mel on Wed, 14 Mar 2007 16:47:39 GMT
View Forum Message <> Reply to Message

On (13/03/07 10:26), Dave Hansen didst pronounce:
> On Mon, 2007-03-12 at 22:04 -0800, Andrew Morton wrote:
> > So these mmapped pages will contiue to be shared across all guests. The
> > problem boils down to "which guest(s) get charged for each shared page".
> >
> > A simple and obvious and easy-to-implement answer is "the guest which paged
> > it in". I think we should firstly explain why that is insufficient.
>
> My first worry was that this approach is unfair to the poor bastard that
> happened to get started up first. If we have a bunch of containerized
> web servers, the poor guy who starts Apache first will pay the price for
> keeping it in memory for everybody else.
>

I think it would be very difficult in practice to exploit a situation where
an evil guy forces another container to hold shared pages that the container
is not using themselves.

> That said, I think this is naturally worked around. The guy charged

> unfairly will get reclaim started on himself sooner. This will tend to
> page out those pages that he was being unfairly charged for.

Exactly. That said, the "poor bastard" will have to be pretty determined
to page out because the pages will appear active but it should happen
eventually especially if the container is under pressure.

> Hopefully,
> they will eventually get pretty randomly (eventually evenly) spread
> among all users. We just might want to make sure that we don't allow
> ptes (or other new references) to be re-established to pages like this
> when we're trying to reclaim them.

I don't think anything like that currently exists. It's almost the opposite
of what the current reclaim algorithm would be trying to do because it has no
notion of containers. Currently, the idea of paging out something in active
use is a mad plan.

Maybe what would be needed is something where the shared page is unmapped from
page tables and the next faulter must copy the page instead of reestablishing
the PTE. The data copy is less than ideal but it'd be cheaper than reclaim
and help the accounting. However, it would require a counter to track "how
many processes in this container have mapped the page".

> Either that, or force the next
> toucher to take ownership of the thing. But, that kind of arbitrary
> ownership transfer can't happen if we have rigidly defined boundaries
> for the containers.
>

Right, charging the next toucher would not work in the zones case. The next
toucher would establish a PTE to the page which is still in the zone of the
container being unfairly charged. It would need to be paged out or copied.

> The other concern is that the memory load on the system doesn't come
> from the first user ("the guy who paged it in"). The long-term load
> comes from "the guy who keeps using it." The best way to exemplify this
> is somebody who read()s a page in, followed by another guy mmap()ing the
> same page. The guy who did the read will get charged, and the mmap()er
> will get a free ride. We could probably get an idea when this kind of
> stuff is happening by comparing page->count and page->_mapcount, but it
> certainly wouldn't be conclusive. But, does this kind of nonsense even
> happen in practice?
>

I think this problem would happen with other accounting mechanisms as
well. However, it's more pronounced with zones because there are harder
limits on memory usage.

If the counter existed to track "how many processes in this container have
mapped the page", the problem of free-riders could be investigated by comparing
_mapcount to the container count. That would determine if additional steps
are required or not to force another container to assume the accounting cost.

--
Mel Gorman
Part-time Phd Student Linux Technology Center
University of Limerick IBM Dublin Software Lab

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Dave Hansen on Wed, 14 Mar 2007 20:42:18 GMT
View Forum Message <> Reply to Message

On Wed, 2007-03-14 at 15:38 +0000, Mel Gorman wrote:
> On (13/03/07 10:05), Dave Hansen didst pronounce:
> > How do we determine what is shared, and goes into the shared zones?
>
> Assuming we had a means of creating a zone that was assigned to a container,
> a second zone for shared data between a set of containers. For shared data,
> the time the pages are being allocated is at page fault time. At that point,
> the faulting VMA is known and you also know if it's MAP_SHARED or not.

Well, but MAP_SHARED does not necessarily mean shared outside of the
container, right? Somebody wishing to get around resource limits could
just MAP_SHARED any data they wished to use, and get it into the shared
area before their initial use, right?

How do normal read/write()s fit into this?

> > There's a conflict between the resize granularity of the zones, and the
> > storage space their lookup consumes. We'd want a container to have a
> > limited ability to fill up memory with stuff like the dcache, so we'd
> > appear to need to put the dentries inside the software zone. But, that
> > gets us to our inability to evict arbitrary dentries.
>
> Stuff like shrinking dentry caches is already pretty course-grained.
> Last I looked, we couldn't even shrink within a specific node, let alone
> a zone or a specific dentry. This is a separate problem.

I shouldn't have used dentries as an example. I'm just saying that if
we end up (or can end up with) with a whole ton of these software zones,

we might have troubles storing them. I would imagine the issue would
come immediately from lack of page->flags to address lots of them.

> > After a while,
> > would containers tend to pin an otherwise empty zone into place? We
> > could resize it, but what is the cost of keeping zones that can be
> > resized down to a small enough size that we don't mind keeping it there?
> > We could merge those "orphaned" zones back into the shared zone.
>
> Merging "orphaned" zones back into the "main" zone would seem a sensible
> choice.

OK, but merging wouldn't be possible if they're not physically
contiguous. I guess this could be worked around by just calling it a
shared zone, no matter where it is physically.

> > Were there any requirements about physical contiguity?
>
> For the lookup to software zone to be efficient, it would be easiest to have
> them as MAX_ORDER_NR_PAGES contiguous. This would avoid having to break the
> existing assumptions in the buddy allocator about MAX_ORDER_NR_PAGES
> always being in the same zone.

I was mostly wondering about zones spanning other zones. We _do_
support this today, and it might make quite a bit more merging possible.

> > If we really do bind a set of processes strongly to a set of memory on a
> > set of nodes, then those really do become its home NUMA nodes. If the
> > CPUs there get overloaded, running it elsewhere will continue to grab
> > pages from the home. Would this basically keep us from ever being able
> > to move tasks around a NUMA system?
>
> Moving the tasks around would not be easy. It would require a new zone
> to be created based on the new NUMA node and all the data migrated. hmm

I know we _try_ to avoid this these days, but I'm not sure how taking it
away as an option will affect anything.

-- Dave

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code

Kirill Korotaev wrote:

>>The approaches I have seen that don't have a struct page pointer, do
>>intrusive things like try to put hooks everywhere throughout the kernel
>>where a userspace task can cause an allocation (and of course end up
>>missing many, so they aren't secure anyway)... and basically just
>>nasty stuff that will never get merged.
>
>
> User beancounters patch has got through all these...
> The approach where each charged object has a pointer to the owner container,
> who has charged it - is the most easy/clean way to handle
> all the problems with dynamic context change, races, etc.
> and 1 pointer in page struct is just 0.1% overehad.

The pointer in struct page approach is a decent one, which I have
liked since this whole container effort came up. IIRC Linus and Alan
also thought that was a reasonable way to go.

I haven't reviewed the rest of the beancounters patch since looking
at it quite a few months ago... I probably don't have time for a
good review at the moment, but I should eventually.

>>Struct page overhead really isn't bad. Sure, nobody who doesn't use
>>containers will want to turn it on, but unless you're using a big PAE
>>system you're actually unlikely to notice.
>
>
> big PAE doesn't make any difference IMHO
> (until struct pages are not created for non-present physical memory areas)

The issue is just that struct pages use low memory, which is a really
scarce commodity on PAE. One more pointer in the struct page means
64MB less lowmem.

But PAE is crap anyway. We've already made enough concessions in the
kernel to support it. I agree: struct page overhead is not really
significant. The benefits of simplicity seems to outweigh the downside.

>>But again, I'll say the node-container approach of course does avoid
>>this nicely (because we already can get the node from the page). So
>>definitely that approach needs to be discredited before going with this
>>one.
>
>
> But it lacks some other features:

> 1. page can't be shared easily with another container

I think they could be shared. You allocate _new_ pages from your own
node, but you can definitely use existing pages allocated to other
nodes.

> 2. shared page can't be accounted honestly to containers
>   as fraction=PAGE_SIZE/containers-using-it

Yes there would be some accounting differences. I think it is hard
to say exactly what containers are "using" what page anyway, though.
What do you say about unmapped pages? Kernel allocations? etc.

> 3. It doesn't help accounting of kernel memory structures.
>   e.g. in OpenVZ we use exactly the same pointer on the page
>   to track which container owns it, e.g. pages used for page
>   tables are accounted this way.

?
page_to_nid(page) ~= container that owns it.

> 4. I guess container destroy requires destroy of memory zone,
>   which means write out of dirty data. Which doesn't sound
>   good for me as well.

I haven't looked at any implementation, but I think it is fine for
the zone to stay around.

> 5. memory reclamation in case of global memory shortage
>   becomes a tricky/unfair task.

I don't understand why? You can much more easily target a specific
container for reclaim with this approach than with others (because
you have an lru per container).

> 6. You cannot overcommit. AFAIU, the memory should be granted
>   to node exclusive usage and cannot be used by by another containers,
>   even if it is unused. This is not an option for us.

I'm not sure about that. If you have a larger number of nodes, then
you could assign more free nodes to a container on demand. But I
think there would definitely be less flexibility with nodes...

I don't know... and seeing as I don't really know where the google
guys are going with it, I won't misrepresent their work any further ;)

>>Everyone seems to have a plan ;) I don't read the containers list...

>>does everyone still have *different* plans, or is any sort of consensus
>>being reached?
>
>
> hope we'll have it soon :)

Good luck ;)

--
SUSE Labs, Novell Inc.
Send instant messages to your online friends http://au.messenger.yahoo.com

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by Balbir Singh on Thu, 15 Mar 2007 05:44:27 GMT
View Forum Message <> Reply to Message

Nick Piggin wrote:
> Kirill Korotaev wrote:
>
>>> The approaches I have seen that don't have a struct page pointer, do
>>> intrusive things like try to put hooks everywhere throughout the kernel
>>> where a userspace task can cause an allocation (and of course end up
>>> missing many, so they aren't secure anyway)... and basically just
>>> nasty stuff that will never get merged.
>>
>>
>> User beancounters patch has got through all these...
>> The approach where each charged object has a pointer to the owner
>> container,
>> who has charged it - is the most easy/clean way to handle
>> all the problems with dynamic context change, races, etc.
>> and 1 pointer in page struct is just 0.1% overehad.
>
> The pointer in struct page approach is a decent one, which I have
> liked since this whole container effort came up. IIRC Linus and Alan
> also thought that was a reasonable way to go.
>
> I haven't reviewed the rest of the beancounters patch since looking
> at it quite a few months ago... I probably don't have time for a
> good review at the moment, but I should eventually.
>

This patch is not really beancounters.

1. It uses the containers framework
2. It is similar to my RSS controller (http://lkml.org/lkml/2007/2/26/8)

I would say that beancounters are changing and evolving.

>>> Struct page overhead really isn't bad. Sure, nobody who doesn't use
>>> containers will want to turn it on, but unless you're using a big PAE
>>> system you're actually unlikely to notice.
>>
>>
>> big PAE doesn't make any difference IMHO
>> (until struct pages are not created for non-present physical memory
>> areas)
>
> The issue is just that struct pages use low memory, which is a really
> scarce commodity on PAE. One more pointer in the struct page means
> 64MB less lowmem.
>
> But PAE is crap anyway. We've already made enough concessions in the
> kernel to support it. I agree: struct page overhead is not really
> significant. The benefits of simplicity seems to outweigh the downside.
>
>>> But again, I'll say the node-container approach of course does avoid
>>> this nicely (because we already can get the node from the page). So
>>> definitely that approach needs to be discredited before going with this
>>> one.
>>
>>
>> But it lacks some other features:
>> 1. page can't be shared easily with another container
>
> I think they could be shared. You allocate _new_ pages from your own
> node, but you can definitely use existing pages allocated to other
> nodes.
>
>> 2. shared page can't be accounted honestly to containers
>>   as fraction=PAGE_SIZE/containers-using-it
>
> Yes there would be some accounting differences. I think it is hard
> to say exactly what containers are "using" what page anyway, though.
> What do you say about unmapped pages? Kernel allocations? etc.
>
>> 3. It doesn't help accounting of kernel memory structures.
>>   e.g. in OpenVZ we use exactly the same pointer on the page
>>   to track which container owns it, e.g. pages used for page
>>   tables are accounted this way.

>
> ?
> page_to_nid(page) ~= container that owns it.
>
>> 4. I guess container destroy requires destroy of memory zone,
>>   which means write out of dirty data. Which doesn't sound
>>   good for me as well.
>
> I haven't looked at any implementation, but I think it is fine for
> the zone to stay around.
>
>> 5. memory reclamation in case of global memory shortage
>>   becomes a tricky/unfair task.
>
> I don't understand why? You can much more easily target a specific
> container for reclaim with this approach than with others (because
> you have an lru per container).
>

Yes, but we break the global LRU. With these RSS patches, reclaim not
triggered by containers still uses the global LRU, by using nodes,
we would have lost the global LRU.

>> 6. You cannot overcommit. AFAIU, the memory should be granted
>>   to node exclusive usage and cannot be used by by another containers,
>>   even if it is unused. This is not an option for us.
>
> I'm not sure about that. If you have a larger number of nodes, then
> you could assign more free nodes to a container on demand. But I
> think there would definitely be less flexibility with nodes...
>
> I don't know... and seeing as I don't really know where the google
> guys are going with it, I won't misrepresent their work any further ;)
>
>
>>> Everyone seems to have a plan ;) I don't read the containers list...
>>> does everyone still have *different* plans, or is any sort of consensus
>>> being reached?
>>
>>
>> hope we'll have it soon :)
>
> Good luck ;)
>

I think we have made some forward progress on the consensus.

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

_____

Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 1/7] Resource counters
Posted by ebiederm on Thu, 15 Mar 2007 16:51:46 GMT
View Forum Message <> Reply to Message

Pavel Emelianov <xemul@sw.ru> writes:

> Srivatsa Vaddagiri wrote:
>> On Tue, Mar 13, 2007 at 06:41:05PM +0300, Pavel Emelianov wrote:
>>>> right, but atomic ops have much less impact on most
>>>> architectures than locks :)
>>> Right. But atomic_add_unless() is slower as it is
>>> essentially a loop. See my previous letter in this sub-thread.
>>
>> If I am not mistaken, you shouldn't loop in normal cases, which means
>> it boils down to a atomic_read() + atomic_cmpxch()
>>
>>
>
> So does the lock - in a normal case (when it's not
> heavily contented) it will boil down to atomic_dec_and_test().
>
> Nevertheless, making charge like in this patchset
> requires two atomic ops with atomic_xxx and only
> one with spin_lock().

To be very clear. If you care about optimization cache lines
and lock hold times (to keep contention down) are the important
things.

With spin locks you have to be a little more careful to put them
on the same cache line as your data and to keep should hold times
short. With atomic ops you get that automatically.

There is really no significant advantage in either approach.
The number of atomic ops doesn't matter. You bring in
the cache line and manipulate it. The expensive part is
acquiring the cache line exclusively. This is expensive even if
things are never contended but there are many users.

Sorry for the rant, but I just wanted to set the record straight.
spin_locks vs atomic ops is a largely meaningless debate.

Eric

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by ebiederm on Fri, 16 Mar 2007 00:55:45 GMT
View Forum Message <> Reply to Message

Alan Cox <alan@lxorguk.ukuu.org.uk> writes:

>> stuff is happening by comparing page->count and page->_mapcount, but it
>> certainly wouldn't be conclusive. But, does this kind of nonsense even
>> happen in practice?
>
> "Is it useful for me as a bad guy to make it happen ?"

To create a DOS attack.

- Allocate some memory you know your victim will want in the future,
  (shared libraries and the like).
- Wait until your victim is using the memory you allocated.
- Terminate your memory resource group.
- Victim is pushed over memory limits by your exiting.
- Victim can no longer allocate memory
- Victim dies

It's not quite that easy unless your victim calls mlockall(MCL_FUTURE),
but the potential is clearly there.

Am I missing something? Or is this fundamental to any first touch scenario?

I just know I have problems with first touch because it is darn hard to
reason about.

Eric

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Dave Hansen on Fri, 16 Mar 2007 16:31:00 GMT
View Forum Message <> Reply to Message

On Thu, 2007-03-15 at 18:55 -0600, Eric W. Biederman wrote:
> To create a DOS attack.
>
> - Allocate some memory you know your victim will want in the future,
> (shared libraries and the like).
> - Wait until your victim is using the memory you allocated.
> - Terminate your memory resource group.
> - Victim is pushed over memory limits by your exiting.
> - Victim can no longer allocate memory
> - Victim dies
>
> It's not quite that easy unless your victim calls mlockall(MCL_FUTURE),
> but the potential is clearly there.
>
> Am I missing something? Or is this fundamental to any first touch scenario?
>
> I just know I have problems with first touch because it is darn hard to
> reason about.

I think it's fundamental to any case where two containers share the use
of the page, but either one _can_ be charged but does not receive a
full charge for it.

I don't think it's uniquely associated with first-touch schemes.

The software zones approach where there would be a set of "shared" zones
would not have this problem, because any sharing would have to occur on
data on which neither one was being charged.

http://linux-mm.org/SoftwareZones

-- Dave

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by ebiederm on Fri, 16 Mar 2007 18:54:09 GMT
View Forum Message <> Reply to Message

Dave Hansen <hansendc@us.ibm.com> writes:

> On Thu, 2007-03-15 at 18:55 -0600, Eric W. Biederman wrote:
>> To create a DOS attack.
>>
>> - Allocate some memory you know your victim will want in the future,
>>   (shared libraries and the like).
>> - Wait until your victim is using the memory you allocated.
>> - Terminate your memory resource group.
>> - Victim is pushed over memory limits by your exiting.
>> - Victim can no longer allocate memory
>> - Victim dies
>>
>> It's not quite that easy unless your victim calls mlockall(MCL_FUTURE),
>> but the potential is clearly there.
>>
>> Am I missing something? Or is this fundamental to any first touch scenario?
>>
>> I just know I have problems with first touch because it is darn hard to
>> reason about.
>
> I think it's fundamental to any case where two containers share the use
> of the page, but either one _can_ be charged but does not receive a
> _full_ charge for it.

Reasonable.

> I don't think it's uniquely associated with first-touch schemes.
>
> The software zones approach where there would be a set of "shared" zones
> would not have this problem, because any sharing would have to occur on
> data on which neither one was being charged.

True. The "shared" zones approach would simply have the problem that it
would make sharing hard and thus reduce the effectiveness of the page cache.

The "shared" zone approach also would seem to interact in very weird ways
with real NUMA and memory hotplug or process migration. The fact that we
actually have to care about the real memory size on the machine makes me
look at it strange.

Zones should definitely be penalized in some category for the reduction
in efficiency of the page cache. It took us decades to learn that the
most efficient page cache was one that could resize and reallocate memory
on demand based on the current usage. Zones and possibly anything else
with the concept of page ownership seems to be trying to be ignoring
that wisdom.

> http://linux-mm.org/SoftwareZones

Looking at your page, and I'm too lazy to figure out how to update it
I have a couple of comments.

- Why do limits have to apply to the unmapped page cache?

- Could you mention proper multi process RSS limits.
  (I.e. we count the number of pages each group of processes have mapped
  and limit that).
  It is the same basic idea as partial page ownership, but instead of
  page ownership you just count how many pages each group is using and
  strictly limit that. There is no page owner ship or partial charges.
  The overhead is just walking the rmap list at map and unmap time to
  see if this is the first users in the container. No additional kernel
  data structures are needed.

Eric

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Dave Hansen on Fri, 16 Mar 2007 19:46:52 GMT
View Forum Message <> Reply to Message

On Fri, 2007-03-16 at 12:54 -0600, Eric W. Biederman wrote:
> Dave Hansen <hansendc@us.ibm.com> writes:
> > http://linux-mm.org/SoftwareZones

> Looking at your page, and I'm too lazy to figure out how to update it
> I have a couple of comments.

You just need to create an account by clicking the Login button. It
lets you edit things after that. But, I'd be happy to put anything in
there you see fit.

> - Why do limits have to apply to the unmapped page cache?

To me, it is just because it consumes memory. Unmapped cache is, of
couse, much more easily reclaimed than mapped files, but it still
fundamentally causes pressure on the VM.

To me, a process sitting there doing constant reads of 10 pages has the
same overhead to the VM as a process sitting there with a 10 page file
mmaped, and reading that.

> - Could you mention proper multi process RSS limits.
> (I.e. we count the number of pages each group of processes have mapped
> and limit that).
> It is the same basic idea as partial page ownership, but instead of
> page ownership you just count how many pages each group is using and
> strictly limit that. There is no page owner ship or partial charges.
> The overhead is just walking the rmap list at map and unmap time to
> see if this is the first users in the container. No additional kernel
> data structures are needed.

I've tried to capture this. Let me know what else you think it needs.

http://linux-mm.org/SoftwareZones

-- Dave

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

---

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by ebiederm on Sun, 18 Mar 2007 16:58:06 GMT
View Forum Message <> Reply to Message

Dave Hansen <hansendc@us.ibm.com> writes:

> On Mon, 2007-03-12 at 23:41 +0100, Herbert Poetzl wrote:
>>
>> let me give a real world example here:
>>
>> - typical guest with 600MB disk space
>> - about 100MB guest specific data (not shared)
>> - assumed that 80% of the libs/tools are used
>
> I get the general idea here, but I just don't think those numbers are
> very accurate. My laptop has a bunch of gunk open (xterm, evolution,
> firefox, xchat, etc...). I ran this command:
>
> lsof | egrep '/(usr/|lib.*\.so)' | awk '{print $9}' | sort | uniq | xargs du
> -Dcs
>
> and got:
>
> 113840 total
>

> On a web/database server that I have (ps aux | wc -l == 128), I just ran
> the same:
>
> 39168   total
>
> That's assuming that all of the libraries are fully read in and
> populated, just by their on-disk sizes. Is that not a reasonable measure
> of the kinds of things that we can expect to be shared in a vserver? If
> so, it's a long way from 400MB.
>
> Could you try a similar measurement on some of your machines? Perhaps
> mine are just weird.

Think shell scripts and the like. From what I have seen I would agree
that is typical for application code not to dominate application memory usage.
However on the flip side it is non uncommon for application code to dominate
disk usage. Some of us have giant music, video or code databases that consume
a lot of disk space but in many instances servers don't have enormous chunks
of private files, and even when they do they share the files from the distribution.

The result of this is that there are a lot of unmapped pages cached in the page
cache for rarely run executables, that are cached just in case we need them.

So while Herbert's numbers may be a little off the general principle of the entire
system doing better if you can share the page cache is very real.

That the page cache isn't accounted for here isn't terribly important we still
get the global benefit.

> I don't doubt this, but doing this two-level page-out thing for
> containers/vservers over their limits is surely something that we should
> consider farther down the road, right?

It is what the current VM of linux does. There is removing a page from
processes and then there is writing it out to disk. I think the normal
term is second chance replacement. The idea is that once you remove
a page from being mapped you let it age a little before it is paged
back in. This allows pages in high demand to avoid being written
to disk, all they incur are minor not major fault costs.

> It's important to you, but you're obviously not doing any of the
> mainline coding, right?

Tread carefully here. Herbert may not be doing a lot of mainline coding
or extremely careful review of potential patches but he does seem to have
a decent grasp of the basic issues. In addition to a reasonable amount
of experience so it is worth listening to what he says.

In addition Herbert does seem to be doing some testing of the mainline
code as we get it going. So he is contributing.

>> > What are the consequences if this isn't done? Doesn't
>> > a loaded system eventually have all of its pages used
>> > anyway, so won't this always be a temporary situation?
>>
>> let's consider a quite limited guest (or several
>> of them) which have a 'RAM' limit of 64MB and
>> additional 64MB of 'virtual swap' assigned ...
>>
>> if they use roughly 96MB (memory footprint) then
>> having this 'fluffy' optimization will keep them
>> running without any effect on the host side, but
>> without, they will continously swap in and out
>> which will affect not only the host, but also the
>> other guests ...

Ugh. You really want swap > RAM here. Because there are real
cases when you are swapping when all of your pages in RAM can
be cached in the page cache. 96MB with 64MB RSS and 64MB swap is
almost a sure way to hit your swap page limit and die.

> All workloads that use $limit+1 pages of memory will always pay the
> price, right? :)

They should. When you remove an anonymous page from the pages tables it
needs to be allocated and placed in the swap cache. Once you do that
it can sit in the page cache like any file backed page. So the
container that hits $limit+1 should get the paging pressure and a lot
more minor faults. However we still want to globally write thing to
disk and optimize that as we do right now.

Eric

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

---

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by ebiederm on Sun, 18 Mar 2007 17:42:15 GMT
View Forum Message <> Reply to Message

Dave Hansen <hansendc@us.ibm.com> writes:

> On Fri, 2007-03-16 at 12:54 -0600, Eric W. Biederman wrote:
>> Dave Hansen <hansendc@us.ibm.com> writes:

>
>> - Why do limits have to apply to the unmapped page cache?
>
> To me, it is just because it consumes memory. Unmapped cache is, of
> couse, much more easily reclaimed than mapped files, but it still
> fundamentally causes pressure on the VM.
>
> To me, a process sitting there doing constant reads of 10 pages has the
> same overhead to the VM as a process sitting there with a 10 page file
> mmaped, and reading that.

I can see temporarily accounting for pages in use for such a
read/write and possibly during things such as read ahead.

However I doubt it is enough memory to be significant, and as
such is probably a waste of time accounting for it.

A memory limit is not about accounting for memory pressure, so I think
the reasoning for wanting to account for unmapped pages as a hard
requirement is still suspect. A memory limit is to prevent one container
from hogging all of the memory in the system, and denying it to other
containers.

The page cache by definition is a global resource that facilitates
global kernel optimizations. If we kill those optimizations we
are on the wrong track. By requiring limits there I think we are
very likely to kill our very important global optimizations, and bring
the performance of the entire system down.

>> - Could you mention proper multi process RSS limits.
>>   (I.e. we count the number of pages each group of processes have mapped
>>   and limit that).
>>   It is the same basic idea as partial page ownership, but instead of
>>   page ownership you just count how many pages each group is using and
>>   strictly limit that. There is no page owner ship or partial charges.
>>   The overhead is just walking the rmap list at map and unmap time to
>>   see if this is the first users in the container. No additional kernel
>>   data structures are needed.
>
> I've tried to capture this. Let me know what else you think it
> needs.

Requirements:
- The current kernel global optimizations are preserved and useful.

  This does mean one container can affect another when the
  optimizations go awry but on average it means much better
  performance. For many the global optimizations are what make

the in-kernel approach attractive over paravirtualization.

Very nice to have:
- Limits should be on things user space have control of.

  Saying you can only have X bytes of kernel memory for file
  descriptors and the like is very hard to work with. Saying you
  can have only N file descriptors open is much easier to deal with.

- SMP Scalability.

  The final implementation should have per cpu counters or per task
  reservations so in most instances we don't need to bounce a global
  cache line around to perform the accounting.

Nice to have:

- Perfect precision.

  Having every last byte always accounted for is nice but a
  little bit of bounded fuzziness in the accounting is acceptable
  if it that make the accounting problem more tractable.

We need several more limits in this discussion to get a full picture,
otherwise we may to try and build the all singing all dancing limit.
- A limit on the number of anonymous pages.
  (Pages that are or may be in the swap cache).
- Filesystem per container quotas.
  (Only applicable in some contexts but you get the idea).
- Inode, file descriptor, and similar limits.
- I/O limits.

Eric

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

---

Subject: Re: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Paul Menage on Sun, 18 Mar 2007 22:44:24 GMT
View Forum Message <> Reply to Message

On 3/13/07, Dave Hansen <hansendc@us.ibm.com> wrote:
> How do we determine what is shared, and goes into the shared zones?
> Once we've allocated a page, it's too late because we already picked.
> Do we just assume all page cache is shared? Base it on filesystem,

> mount, ...? Mount seems the most logical to me, that a sysadmin would
> have to set up a container's fs, anyway, and will likely be doing
> special things to shared data, anyway (r/o bind mounts :).

I played with an approach where you can bind a dentry to a set of
memory zones, and any children of that dentry would inherit the
mempolicy; I was envisaging that most data wouldn't be shared between
different containers/jobs, and that userspace would set up "shared"
zones for big shared regions such as /lib, /usr, /bin, and for
specially-known cases of sharing.

> If we really do bind a set of processes strongly to a set of memory on a
> set of nodes, then those really do become its home NUMA nodes. If the
> CPUs there get overloaded, running it elsewhere will continue to grab
> pages from the home. Would this basically keep us from ever being able
> to move tasks around a NUMA system?

move_pages() will let you shuffle tasks from one node to another
without too much intrusion.

Paul

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

---

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by Herbert Poetzl on Mon, 19 Mar 2007 15:48:08 GMT
View Forum Message <> Reply to Message

On Sun, Mar 18, 2007 at 11:42:15AM -0600, Eric W. Biederman wrote:
> Dave Hansen <hansendc@us.ibm.com> writes:
>
> > On Fri, 2007-03-16 at 12:54 -0600, Eric W. Biederman wrote:
> >> Dave Hansen <hansendc@us.ibm.com> writes:
> >
> >> - Why do limits have to apply to the unmapped page cache?
> >
> > To me, it is just because it consumes memory. Unmapped cache is, of
> > couse, much more easily reclaimed than mapped files, but it still
> > fundamentally causes pressure on the VM.
> >
> > To me, a process sitting there doing constant reads of 10 pages has the
> > same overhead to the VM as a process sitting there with a 10 page file
> > mmaped, and reading that.
>
> I can see temporarily accounting for pages in use for such a

> read/write and possibly during things such as read ahead.
>
> However I doubt it is enough memory to be significant, and as
> such is probably a waste of time accounting for it.
>
> A memory limit is not about accounting for memory pressure, so I think
> the reasoning for wanting to account for unmapped pages as a hard
> requirement is still suspect.

> A memory limit is to prevent one container from hogging all of the
> memory in the system, and denying it to other containers.

exactly!

nevertheless, you might want to extend that to swapping
and to the very expensive page in/out operations too

> The page cache by definition is a global resource that facilitates
> global kernel optimizations. If we kill those optimizations we
> are on the wrong track. By requiring limits there I think we are
> very likely to kill our very important global optimizations, and bring
> the performance of the entire system down.

that is my major concern for most of the 'straight forward'
virtualizations proposed (see Xen comment)

> >> - Could you mention proper multi process RSS limits.
> >>   (I.e. we count the number of pages each group of processes have mapped
> >>   and limit that).
> >>   It is the same basic idea as partial page ownership, but instead of
> >>   page ownership you just count how many pages each group is using and
> >>   strictly limit that. There is no page owner ship or partial charges.
> >>   The overhead is just walking the rmap list at map and unmap time to
> >>   see if this is the first users in the container. No additional kernel
> >>   data structures are needed.
> >
> > I've tried to capture this. Let me know what else you think it
> > needs.
>
> Requirements:
> - The current kernel global optimizations are preserved and useful.
>
>   This does mean one container can affect another when the
>   optimizations go awry but on average it means much better
>   performance. For many the global optimizations are what make
>   the in-kernel approach attractive over paravirtualization.

total agreement here

> Very nice to have:
> - Limits should be on things user space have control of.
>
> Saying you can only have X bytes of kernel memory for file
> descriptors and the like is very hard to work with. Saying you
> can have only N file descriptors open is much easier to deal with.

yep, and IMHO more natural ...

> - SMP Scalability.
>
> The final implementation should have per cpu counters or per task
> reservations so in most instances we don't need to bounce a global
> cache line around to perform the accounting.

agreed, we want to optimize for small systems
as well as for large ones, and SMP/NUMA is quite
common in the server area (even for small servers)

> Nice to have:
>
> - Perfect precision.
>
> Having every last byte always accounted for is nice but a
> little bit of bounded fuzziness in the accounting is acceptable
> if it that make the accounting problem more tractable.

as long as the accounting is consistant, i.e.
you do not lose resources by repetitive operations
inside the guest (or through guest-guest interaction)
as this could be used for DoS and intentional unfairness

> We need several more limits in this discussion to get a full picture,
> otherwise we may to try and build the all singing all dancing limit.

> - A limit on the number of anonymous pages.
> (Pages that are or may be in the swap cache).

> - Filesystem per container quotas.
> (Only applicable in some contexts but you get the idea).

with shared files, otherwise an lvm partition does
a good job for that already ...

> - Inode, file descriptor, and similar limits.

> - I/O limits.

I/O and CPU limits are special, as they have the temporal
component, i.e. you are not interested in 10s CPU time,
instead you want 0.5s/s CPU (same for I/O)

note: this is probably also true for page in/out

- sockets
- locks
- dentries

HTH,
Herbert

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

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

---

Subject: Re: Re: [RFC][PATCH 2/7] RSS controller core
Posted by ebiederm on Mon, 19 Mar 2007 17:41:23 GMT
View Forum Message <> Reply to Message

"Paul Menage" <menage@google.com> writes:

> On 3/13/07, Dave Hansen <hansendc@us.ibm.com> wrote:
>> How do we determine what is shared, and goes into the shared zones?
>> Once we've allocated a page, it's too late because we already picked.
>> Do we just assume all page cache is shared? Base it on filesystem,
>> mount, ...? Mount seems the most logical to me, that a sysadmin would
>> have to set up a container's fs, anyway, and will likely be doing
>> special things to shared data, anyway (r/o bind mounts :).
>
> I played with an approach where you can bind a dentry to a set of
> memory zones, and any children of that dentry would inherit the
> mempolicy; I was envisaging that most data wouldn't be shared between
> different containers/jobs, and that userspace would set up "shared"
> zones for big shared regions such as /lib, /usr, /bin, and for
> specially-known cases of sharing.

Here is a wacky one.

Suppose there is some NFS server that exports something that most machines
want to mount like company home directories.

Suppose multiple containers mount that NFS server based on local policy.
(If we can allow non-root users to mount filesystems a slightly more trusted
 guest admin certainly will be able to).

The NFS code as current written (unless I am confused) will do
everything in it's power to share the filesystem cache between the
different mounts (including the dentry tree).

How do we handle bit shared areas like that.

Dynamic programming solutions where we discovery the areas of sharing
at runtime seem a lot more general then a priori solutions where you
have to predict what will come next.

If a priori planning and knowledge about sharing is the best we can do
it is the best we can do and we will have to live with the limits that
imposes. Given the inflexibility in use and setup I'm not yet ready
to concede that this is the best we can do.

Eric

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

---

Subject: controlling mmap()'d vs read/write() pages
Posted by Dave Hansen on Tue, 20 Mar 2007 16:15:34 GMT
View Forum Message <> Reply to Message

On Sun, 2007-03-18 at 11:42 -0600, Eric W. Biederman wrote:
> Dave Hansen <hansendc@us.ibm.com> writes:
> > To me, a process sitting there doing constant reads of 10 pages has the
> > same overhead to the VM as a process sitting there with a 10 page file
> > mmaped, and reading that.
>
> I can see temporarily accounting for pages in use for such a
> read/write and possibly during things such as read ahead.
>
> However I doubt it is enough memory to be significant, and as
> such is probably a waste of time accounting for it.
>
> A memory limit is not about accounting for memory pressure, so I think
> the reasoning for wanting to account for unmapped pages as a hard

> requirement is still suspect. A memory limit is to prevent one container
> from hogging all of the memory in the system, and denying it to other
> containers.
>
> The page cache by definition is a global resource that facilitates
> global kernel optimizations. If we kill those optimizations we
> are on the wrong track. By requiring limits there I think we are
> very likely to kill our very important global optimizations, and bring
> the performance of the entire system down.

Let's say you have an mmap'd file. It has zero pages brought in right
now. You do a write to it. It is well within the kernel's rights to
let you write one word to an mmap'd file, then unmap it, write it to
disk, and free the page.

To me, mmap() is an interface, not a directive to tell the kernel to
keep things in memory. The fact that two reads of a bytes from an
mmap()'d file tends to not go to disk or even cause a fault for the
second read is because the page is in the page cache. The fact that two
consecutive read()s of the same disk page tend to not cause two trips to
the disk is because the page is in the page cache.

Anybody who wants to get data in and out of a file can choose to use
either of these interfaces. A page being brought into the system for
either a read or touch of an mmap()'d area causes the same kind of
memory pressure.

So, I think we have a difference of opinion. I think it's _all_ about
memory pressure, and you think it is _not_ about accounting for memory
pressure. :) Perhaps we mean different things, but we appear to
disagree greatly on the surface.

Can we agree that there must be _some_ way to control the amounts of
unmapped page cache? Whether that's related somehow to the same way we
control RSS or done somehow at the I/O level, there must be some way to
control it. Agree?

> >> - Could you mention proper multi process RSS limits.
> >>   (I.e. we count the number of pages each group of processes have mapped
> >>   and limit that).
> >>   It is the same basic idea as partial page ownership, but instead of
> >>   page ownership you just count how many pages each group is using and
> >>   strictly limit that. There is no page owner ship or partial charges.
> >>   The overhead is just walking the rmap list at map and unmap time to
> >>   see if this is the first users in the container. No additional kernel
> >>   data structures are needed.
> >
> > I've tried to capture this. Let me know what else you think it

> > needs.
>
> Requirements:
> - The current kernel global optimizations are preserved and useful.
>
> This does mean one container can affect another when the
> optimizations go awry but on average it means much better
> performance. For many the global optimizations are what make
> the in-kernel approach attractive over paravirtualization.
>
> Very nice to have:
> - Limits should be on things user space have control of.
...
> - SMP Scalability.

> - Perfect precision.
...

I've tried to capture this:

http://linux-mm.org/SoftwareZones

> We need several more limits in this discussion to get a full picture,
> otherwise we may to try and build the all singing all dancing limit.
> - A limit on the number of anonymous pages.
> (Pages that are or may be in the swap cache).
> - Filesystem per container quotas.
> (Only applicable in some contexts but you get the idea).
> - Inode, file descriptor, and similar limits.
> - I/O limits.

Definitely. I think we've all agreed that memory is the hard one,
though. If we can make progress on this one, we're set! :)

-- Dave

Subject: Re: [RFC][PATCH 2/7] RSS controller core
Posted by mel on Tue, 20 Mar 2007 18:57:39 GMT
View Forum Message <> Reply to Message

On (14/03/07 13:42), Dave Hansen didst pronounce:
> On Wed, 2007-03-14 at 15:38 +0000, Mel Gorman wrote:

> > On (13/03/07 10:05), Dave Hansen didst pronounce:
> > > How do we determine what is shared, and goes into the shared zones?
> >
> > Assuming we had a means of creating a zone that was assigned to a container,
> > a second zone for shared data between a set of containers. For shared data,
> > the time the pages are being allocated is at page fault time. At that point,
> > the faulting VMA is known and you also know if it's MAP_SHARED or not.
>
> Well, but MAP_SHARED does not necessarily mean shared outside of the
> container, right?

Well, the data could also be shared outside of the container. I would see
that happening for library text sections for example.

> Somebody wishing to get around resource limits could
> just MAP_SHARED any data they wished to use, and get it into the shared
> area before their initial use, right?
>

They would only be able to impact other containers in a limited sense.
Specifically, if 5 containers have one shared area, then any process in
those 5 containers could exceed their container limits at the expense of
the shared area.

> How do normal read/write()s fit into this?
>

A normal read/write if it's the first reader of a file would get charged to the
container, not to the shared area. It is less likely that a file that is read()
is expected to be shared where as mapping MAP_SHARED is relatively explicit.

> > > There's a conflict between the resize granularity of the zones, and the
> > > storage space their lookup consumes. We'd want a container to have a
> > > limited ability to fill up memory with stuff like the dcache, so we'd
> > > appear to need to put the dentries inside the software zone. But, that
> > > gets us to our inability to evict arbitrary dentries.
> >
> > Stuff like shrinking dentry caches is already pretty course-grained.
> > Last I looked, we couldn't even shrink within a specific node, let alone
> > a zone or a specific dentry. This is a separate problem.
>
> I shouldn't have used dentries as an example. I'm just saying that if
> we end up (or can end up with) with a whole ton of these software zones,
> we might have troubles storing them. I would imagine the issue would
> come immediately from lack of page->flags to address lots of them.
>

That is an immediate problem. There needs to be a way of mapping an arbitrary

page to a software zone. page_zone() as it is could only resolve the "main" zone. If additional bits were used in page->flags, there would be very hard limits on the number of containers that can exist.

If zones were physically contiguous to MAX_ORDER, pageblock flags from the anti-fragmentation could be used to record that a block of pages was in a container and what the ID is. If non-contiguous software zones were required, page->zone could be reintroduced for software zones to be used when a page belongs to a container. It's not ideal the proper way of mapping pages to software zones might be more obvious then when we'd see where page->zone was used.

With either approach, the important thing that occured to me is be to be sure that pages only came from the same hardware zone. For example, do not mix HIGHMEM pages with DMA pages because it'll fail miserably. For RSS accounting, this is not much of a restriction but it does have an impact on keeping kernel allocations within a container on systems with HighMemory.

> > > After a while,
> > > would containers tend to pin an otherwise empty zone into place? We
> > > could resize it, but what is the cost of keeping zones that can be
> > > resized down to a small enough size that we don't mind keeping it there?
> > > We could merge those "orphaned" zones back into the shared zone.
> >
> > Merging "orphaned" zones back into the "main" zone would seem a sensible
> > choice.
>
> OK, but merging wouldn't be possible if they're not physically
> contiguous. I guess this could be worked around by just calling it a
> shared zone, no matter where it is physically.
>

More than likely, yes.

> > > Were there any requirements about physical contiguity?
> >
> > For the lookup to software zone to be efficient, it would be easiest to have
> > them as MAX_ORDER_NR_PAGES contiguous. This would avoid having to break the
> > existing assumptions in the buddy allocator about MAX_ORDER_NR_PAGES
> > always being in the same zone.
>
> I was mostly wondering about zones spanning other zones. We _do_
> support this today

In practice, overlapping zones never happen today so a few new bugs based on assumptions about MAX_ORDER_NR_PAGES being aligned in a zone may crop up.

>, and it might make quite a bit more merging possible.
>
> > > If we really do bind a set of processes strongly to a set of memory on a
> > > set of nodes, then those really do become its home NUMA nodes. If the
> > > CPUs there get overloaded, running it elsewhere will continue to grab
> > > pages from the home. Would this basically keep us from ever being able
> > > to move tasks around a NUMA system?
> >
> > Moving the tasks around would not be easy. It would require a new zone
> > to be created based on the new NUMA node and all the data migrated. hmm
>
> I know we _try_ to avoid this these days, but I'm not sure how taking it
> away as an option will affect anything.
>

--
Mel Gorman
Part-time Phd Student Linux Technology Center
University of Limerick IBM Dublin Software Lab

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

---

Subject: Re: controlling mmap()'d vs read/write() pages
Posted by ebiederm on Tue, 20 Mar 2007 21:19:16 GMT
View Forum Message <> Reply to Message

Dave Hansen <hansendc@us.ibm.com> writes:

>
> So, I think we have a difference of opinion. I think it's _all_ about
> memory pressure, and you think it is _not_ about accounting for memory
> pressure. :) Perhaps we mean different things, but we appear to
> disagree greatly on the surface.

I think it is about preventing a badly behaved container from having a
significant effect on the rest of the system, and in particular other
containers on the system.

See below. I think to reach agreement we should start by discussing
the algorithm that we see being used to keep the system function well
and the theory behind that algorithm. Simply limiting memory is not
enough to understand why it works....

> Can we agree that there must be _some_ way to control the amounts of
> unmapped page cache? Whether that's related somehow to the same way we

> control RSS or done somehow at the I/O level, there must be some way to
> control it. Agree?

At lot depends on what we measure and what we try and control.
Currently what we have been measuring are amounts of RAM, and thus
what we are trying to control is the amount of RAM. If we want to
control memory pressure we need a definition and a way to measure it.
I think there may be potential if we did that but we would still need
a memory limit to keep things like mlock in check.

So starting with a some definitions and theory.
RSS is short for resident set size. The resident set being how many
of pages are current in memory and not on disk and used by the
application. This includes the memory in page tables, but can
reasonably be extended to include any memory a process can be shown to
be using.

In theory there is some minimal RSS that you can give an application
at which it will get productive work done. Below the minimal RSS
the application will spend the majority of real time waiting for
pages to come in from disk, so it can execute the next instruction.
The ultimate worst case here is a read instruction appearing on one
page and it's datum on another. You have to have both pages in memory
at the same time for the read to complete. If you set the RSS hard
limit to one page the problem will be continually restarting either
because the page it is on is not in memory or the page it is reading
from is not in memory.

What we want to accomplish is to have a system that runs multiple
containers without problems. As a general memory management policy
we can accomplish this by ensuring each container has at least
it's minimal RSS quota of pages. By watching the paging activity
of a container it is possible to detect when that container has
to few pages and is spend all of it's time I/O bound, and thus
has slipped below it's minimal RSS.

As such it is possible for the memory management system if we have
container RSS accounting to dynamically figure out how much memory
each container needs and to keep everyone above their minimal RSS
most of the time when that is possible. Basically to do this the
memory manage code would need to keep dynamic RSS limits, and
adjust them based upon need.

There is still the case when not all containers can have their
minimal RSS, there is simply not enough memory in the system.

That is where having a hard settable RSS limit comes in. With this

we communicate to the application and the users beyond which point we
consider their application to be abusing the system.

There is a lot of history with RSS limits showing their limitations
and how they work. It is fundamentally a dynamic policy instead of
a static set of guarantees which allows for applications with a
diverse set of memory requirements to work in harmony.

One of the very neat things about a hard RSS limit is that if there
are extra resources on the system you can improve overall system
performance by cache pages in the page cache instead writing them
to disk.

> http://linux-mm.org/SoftwareZones

I will try and take a look in a bit.

Eric

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

---

Subject: Re: controlling mmap()'d vs read/write() pages
Posted by Herbert Poetzl on Fri, 23 Mar 2007 00:51:36 GMT
View Forum Message <> Reply to Message

On Tue, Mar 20, 2007 at 03:19:16PM -0600, Eric W. Biederman wrote:
> Dave Hansen <hansendc@us.ibm.com> writes:
>
> > > So, I think we have a difference of opinion. I think it's _all_
> > > about memory pressure, and you think it is _not_ about accounting
> > > for memory pressure. :) Perhaps we mean different things, but we
> > > appear to disagree greatly on the surface.
>
> I think it is about preventing a badly behaved container from having a
> significant effect on the rest of the system, and in particular other
> containers on the system.
>
> See below. I think to reach agreement we should start by discussing
> the algorithm that we see being used to keep the system function well
> and the theory behind that algorithm. Simply limiting memory is not
> enough to understand why it works....

>
> > Can we agree that there must be _some_ way to control the amounts of
> > unmapped page cache? Whether that's related somehow to the same way
> > we control RSS or done somehow at the I/O level, there must be some
> > way to control it. Agree?
>
> At lot depends on what we measure and what we try and control.
> Currently what we have been measuring are amounts of RAM, and thus
> what we are trying to control is the amount of RAM. If we want to
> control memory pressure we need a definition and a way to measure it.
> I think there may be potential if we did that but we would still need
> a memory limit to keep things like mlock in check.
>
> So starting with a some definitions and theory.
> RSS is short for resident set size. The resident set being how many
> of pages are current in memory and not on disk and used by the
> application. This includes the memory in page tables, but can
> reasonably be extended to include any memory a process can be shown to
> be using.
>
> In theory there is some minimal RSS that you can give an application
> at which it will get productive work done. Below the minimal RSS
> the application will spend the majority of real time waiting for
> pages to come in from disk, so it can execute the next instruction.
> The ultimate worst case here is a read instruction appearing on one
> page and it's datum on another. You have to have both pages in memory
> at the same time for the read to complete. If you set the RSS hard
> limit to one page the problem will be continually restarting either
> because the page it is on is not in memory or the page it is reading
> from is not in memory.
>
> What we want to accomplish is to have a system that runs multiple
> containers without problems. As a general memory management policy
> we can accomplish this by ensuring each container has at least
> it's minimal RSS quota of pages. By watching the paging activity
> of a container it is possible to detect when that container has
> to few pages and is spend all of it's time I/O bound, and thus
> has slipped below it's minimal RSS.
>
> As such it is possible for the memory management system if we have
> container RSS accounting to dynamically figure out how much memory
> each container needs and to keep everyone above their minimal RSS
> most of the time when that is possible. Basically to do this the
> memory manage code would need to keep dynamic RSS limits, and
> adjust them based upon need.
>
> There is still the case when not all containers can have their
> minimal RSS, there is simply not enough memory in the system.

>
> That is where having a hard settable RSS limit comes in. With this
> we communicate to the application and the users beyond which point we
> consider their application to be abusing the system.
>
> There is a lot of history with RSS limits showing their limitations
> and how they work. It is fundamentally a dynamic policy instead of
> a static set of guarantees which allows for applications with a
> diverse set of memory requirements to work in harmony.
>
> One of the very neat things about a hard RSS limit is that if there
> are extra resources on the system you can improve overall system
> performance by cache pages in the page cache instead writing them
> to disk.

that is exactly what we (Linux-VServer) want ...
(sounds good to me, please keep up the good work in
this direction)

there is nothing wrong with hard limits if somebody
really wants them, even if they hurt the sysstem as
whole, but those limits shouldn't be the default ..

best,
Herbert

> > http://linux-mm.org/SoftwareZones
>
> I will try and take a look in a bit.
>
>
>
> Eric
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers

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

---

Subject: Re: controlling mmap()'d vs read/write() pages
Posted by Nick Piggin on Fri, 23 Mar 2007 05:57:40 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Dave Hansen <hansendc@us.ibm.com> writes:

>
>
>>So, I think we have a difference of opinion. I think it's _all_ about
>>memory pressure, and you think it is _not_ about accounting for memory
>>pressure. :) Perhaps we mean different things, but we appear to
>>disagree greatly on the surface.
>
>
> I think it is about preventing a badly behaved container from having a
> significant effect on the rest of the system, and in particular other
> containers on the system.

That's Dave's point, I believe. Limiting mapped memory may be
mostly OK for well behaved applications, but it doesn't do anything
to stop bad ones from effectively DoSing the system or ruining any
guarantees you might proclaim (not that hard guarantees are always
possible without using virtualisation anyway).

This is why I'm surprised at efforts that go to such great lengths
to get accounting "just right" (but only for mmaped memory). You
may as well not even bother, IMO.

Give me an RSS limit big enough to run a couple of system calls and
a loop...

--
SUSE Labs, Novell Inc.
Send instant messages to your online friends http://au.messenger.yahoo.com

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

---

Subject: Re: controlling mmap()'d vs read/write() pages
Posted by ebiederm on Fri, 23 Mar 2007 10:12:09 GMT
View Forum Message <> Reply to Message

Nick Piggin <nickpiggin@yahoo.com.au> writes:

> Eric W. Biederman wrote:
>> Dave Hansen <hansendc@us.ibm.com> writes:
>>
>>
>>>So, I think we have a difference of opinion. I think it's _all_ about
>>>memory pressure, and you think it is _not_ about accounting for memory
>>>pressure. :) Perhaps we mean different things, but we appear to

>>>disagree greatly on the surface.
>>
>>
>> I think it is about preventing a badly behaved container from having a
>> significant effect on the rest of the system, and in particular other
>> containers on the system.
>
> That's Dave's point, I believe. Limiting mapped memory may be
> mostly OK for well behaved applications, but it doesn't do anything
> to stop bad ones from effectively DoSing the system or ruining any
> guarantees you might proclaim (not that hard guarantees are always
> possible without using virtualisation anyway).
>
> This is why I'm surprised at efforts that go to such great lengths
> to get accounting "just right" (but only for mmaped memory). You
> may as well not even bother, IMO.
>
> Give me an RSS limit big enough to run a couple of system calls and
> a loop...

Would any of them work on a system on which every filesystem was on ramfs, and there was no swap? If not then they are not memory attacks but I/O attacks.

I completely concede that you can DOS the system with I/O if that is not limited as well.

My point is that is not a memory problem but a disk I/O problem which is much easier to and cheaper to solve. Disk I/O is fundamentally a slow path which makes it hard to modify it in a way that negatively affects system performance.

I don't think with a memory RSS limit you can DOS the system in a way that is purely about memory. You have to pick a different kind of DOS attack.

As for virtualization that is what a kernel is about virtualizing it's resources so you can have multiple users accessing them at the same time. You don't need some hypervisor or virtual machine to give you that. That is where we start. However it was found long ago that global optimizations give better system through put then the rigid systems you can get with hypervisors. Although things are not quite as deterministic when you optimize globally. They should be sufficiently deterministic you can avoid the worst of the DOS attacks.

The real practical problem with the current system is that nearly all of our limits are per process and applications now span more than

one process so the limits provided by linux are generally useless
to limit real world applications. This isn't generally a problem
until we start trying to run multiple applications on the same system
because the hardware is so powerful. Which the namespace work which
will allow you to run several different instances of user space
simultaneously is likely to allow.

At the moment I very much in a position of doing review not
implementing this part of it. I'm trying to get the people doing the
implementation to make certain they have actually been paying
attention to how their proposed limits will interact with the rest of
the system. So far generally the conversation has centered on memory
limits because it seems that is where people have decided the
conversation should focus. What I haven't seen is people with the
limitations coming back to me tearing my arguments apart and showing
or telling me where I'm confused. In general I can challenge even the
simplest things and not get a good response. All of which tells me
the implementations are not ready.

I do have some practical use cases and I have some clue how these
subsystems work, and I do care. Which puts in a decent position to
at least to high level design review.

My biggest disappointment is that none of this is new, and that we
seem to have forgotten a lot of the lessons of the past.

Eric

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

---

Subject: Re: controlling mmap()'d vs read/write() pages
Posted by Nick Piggin on Fri, 23 Mar 2007 10:47:45 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Nick Piggin <nickpiggin@yahoo.com.au> writes:
>
>
>>Eric W. Biederman wrote:
>>
>>>Dave Hansen <hansendc@us.ibm.com> writes:
>>>
>>>
>>>
>>>So, I think we have a difference of opinion. I think it's _all_ about

>>>>memory pressure, and you think it is _not_ about accounting for memory
>>>>pressure. :) Perhaps we mean different things, but we appear to
>>>>disagree greatly on the surface.
>>>
>>>
>>>I think it is about preventing a badly behaved container from having a
>>>significant effect on the rest of the system, and in particular other
>>>containers on the system.
>>
>>That's Dave's point, I believe. Limiting mapped memory may be
>>mostly OK for well behaved applications, but it doesn't do anything
>>to stop bad ones from effectively DoSing the system or ruining any
>>guarantees you might proclaim (not that hard guarantees are always
>>possible without using virtualisation anyway).
>>
>>This is why I'm surprised at efforts that go to such great lengths
>>to get accounting "just right" (but only for mmaped memory). You
>>may as well not even bother, IMO.
>>
>>Give me an RSS limit big enough to run a couple of system calls and
>>a loop...
>
>
> Would any of them work on a system on which every filesystem was on
> ramfs, and there was no swap? If not then they are not memory attacks
> but I/O attacks.
>
> I completely concede that you can DOS the system with I/O if that is
> not limited as well.
>
> My point is that is not a memory problem but a disk I/O problem which is
> much easier to and cheaper to solve. Disk I/O is fundamentally a slow
> path which makes it hard to modify it in a way that negatively affects
> system performance.
>
> I don't think with a memory RSS limit you can DOS the system in a way
> that is purely about memory. You have to pick a different kind of DOS
> attack.

It can be done trivially without performing any IO or swap, yes.

--
SUSE Labs, Novell Inc.
Send instant messages to your online friends http://au.messenger.yahoo.com

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

Subject: Re: controlling mmap()'d vs read/write() pages
Posted by ebiederm on Fri, 23 Mar 2007 12:21:01 GMT
View Forum Message <> Reply to Message

Nick Piggin <nickpiggin@yahoo.com.au> writes:

>> Would any of them work on a system on which every filesystem was on
>> ramfs, and there was no swap? If not then they are not memory attacks
>> but I/O attacks.
>>
>> I completely concede that you can DOS the system with I/O if that is
>> not limited as well.
>>
>> My point is that is not a memory problem but a disk I/O problem which is
>> much easier to and cheaper to solve. Disk I/O is fundamentally a slow
>> path which makes it hard to modify it in a way that negatively affects
>> system performance.
>>
>> I don't think with a memory RSS limit you can DOS the system in a way
>> that is purely about memory. You have to pick a different kind of DOS
>> attack.
>
> It can be done trivially without performing any IO or swap, yes.

Please give me a rough sketch of how to do so.

Or is this about DOS'ing the system by getting the kernel to allocate
a large number of data structures (struct file, struct inode, or the like)?

Eric

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

Subject: Re: controlling mmap()'d vs read/write() pages
Posted by Dave Hansen on Fri, 23 Mar 2007 16:41:12 GMT
View Forum Message <> Reply to Message

On Fri, 2007-03-23 at 04:12 -0600, Eric W. Biederman wrote:
> Would any of them work on a system on which every filesystem was on
> ramfs, and there was no swap? If not then they are not memory attacks
> but I/O attacks.

I truly understand your point here. But, I don't think this thought exercise is really helpful here. In a pure sense, nothing is keeping an unmapped page cache file in memory, other than the user's prayers. But, please don't discount their prayers, it's what they want!

I seem to remember a quote attributed to Alan Cox around OLS time last year, something about any memory controller being able to be fair, fast, and accurate. Please pick any two, but only two. Alan, did I get close?

To me, one of the keys of Linux's "global optimizations" is being able to use any memory globally for its most effective purpose, globally (please ignore highmem :). Let's say I have a 1GB container on a machine that is at least 100% committed. I mmap() a 1GB file and touch the entire thing (I never touch it again). I then go open another 1GB file and r/w to it until the end of time. I'm at or below my RSS limit, but that 1GB of RAM could surely be better used for the second file. How do we do this if we only account for a user's RSS? Does this fit into Alan's unfair bucket? ;)

Also, in a practical sense, it is also a *LOT* easier to describe to a customer that they're getting 1GB of RAM than >=20GB/hr of bandwidth from the disk.

-- Dave

P.S. Do we have an quotas on ramfs? If we have an ramfs filesystems, what keeps the containerized users from just filling up RAM?

Subject: Re: controlling mmap()'d vs read/write() pages
Posted by Herbert Poetzl on Fri, 23 Mar 2007 18:16:26 GMT
View Forum Message <> Reply to Message

On Fri, Mar 23, 2007 at 09:41:12AM -0700, Dave Hansen wrote:
> On Fri, 2007-03-23 at 04:12 -0600, Eric W. Biederman wrote:
> > Would any of them work on a system on which every filesystem was on
> > ramfs, and there was no swap? If not then they are not memory attacks
> > but I/O attacks.
>
> I truly understand your point here. But, I don't think this thought
> exercise is really helpful here. In a pure sense, nothing is keeping

> an unmapped page cache file in memory, other than the user's prayers.
> But, please don't discount their prayers, it's what they want!
>
> I seem to remember a quote attributed to Alan Cox around OLS time last
> year, something about any memory controller being able to be fair,
> fast, and accurate. Please pick any two, but only two. Alan, did I get
> close?

so we would pick fair and fast then :)

> To me, one of the keys of Linux's "global optimizations" is being able
> to use any memory globally for its most effective purpose, globally
> (please ignore highmem :). Let's say I have a 1GB container on a
> machine that is at least 100% committed. I mmap() a 1GB file and touch
> the entire thing (I never touch it again). I then go open another 1GB
> file and r/w to it until the end of time. I'm at or below my RSS limit,
> but that 1GB of RAM could surely be better used for the second file.
> How do we do this if we only account for a user's RSS? Does this fit
> into Alan's unfair bucket? ;)

what's the difference to a normal Linux system here?
when low on memory, the system will reclaim pages, and
guess what pages will be reclaimed first ...

> Also, in a practical sense, it is also a *LOT* easier to describe to a
> customer that they're getting 1GB of RAM than >=20GB/hr of bandwidth
> from the disk.

if you want something which is easy to describe for the
'customer', then a VM is what you are looking for, it has
a perfectly well defined amount of resources which will
not be shared or used by other machines ...

> -- Dave
>
> P.S. Do we have an quotas on ramfs? If we have an ramfs filesystems,
> what keeps the containerized users from just filling up RAM?

tmpfs has hard limits, you simply specify it on mount

```
 none /tmp  tmpfs size=16m,mode=1777 0 0
```

best,
Herbert

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

> https://lists.linux-foundation.org/mailman/listinfo/containers

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

---

Subject: Re: controlling mmap()'d vs read/write() pages
Posted by Nick Piggin on Wed, 28 Mar 2007 07:33:37 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> Nick Piggin <nickpiggin@yahoo.com.au> writes:

>>It can be done trivially without performing any IO or swap, yes.
>
>
> Please give me a rough sketch of how to do so.

Reading sparse files is just one I had in mind. But I'm not very
creative compared to university students doing their assignments.

> Or is this about DOS'ing the system by getting the kernel to allocate
> a large number of data structures (struct file, struct inode, or the like)?

That works too. And I don't believe hand-accounting and limiting
all these things individually as a means to limit RAM usage is sane,
when you have a much more comprehensive and relatively unintrusive
page level scheme.

--
SUSE Labs, Novell Inc.
Send instant messages to your online friends http://au.messenger.yahoo.com

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

---

Subject: Re: controlling mmap()'d vs read/write() pages
Posted by Balbir Singh on Wed, 28 Mar 2007 09:18:21 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
>> To me, one of the keys of Linux's "global optimizations" is being able

>> to use any memory globally for its most effective purpose, globally
>> (please ignore highmem :). Let's say I have a 1GB container on a
>> machine that is at least 100% committed. I mmap() a 1GB file and touch
>> the entire thing (I never touch it again). I then go open another 1GB
>> file and r/w to it until the end of time. I'm at or below my RSS limit,
>> but that 1GB of RAM could surely be better used for the second file.
>> How do we do this if we only account for a user's RSS? Does this fit
>> into Alan's unfair bucket? ;)
>
> what's the difference to a normal Linux system here?
> when low on memory, the system will reclaim pages, and
> guess what pages will be reclaimed first ...
>

But would it not bias application writers towards using read()/write()
calls over mmap()? They know that their calls are likely to be faster
when the application is run in a container. Without page cache control
we'll end up creating an asymmetrical container, where certain usage is
charged and some usage is not.

Also, please note that when a page is unmapped and moved to swap cache;
the swap cache uses the page cache. Without page cache control, we could
end up with too many pages moving over to the swap cache and still
occupying memory, while the original intension was to avoid this
scenario.

--
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: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by Ethan Solomita on Wed, 28 Mar 2007 20:15:20 GMT
View Forum Message <> Reply to Message

Nick Piggin wrote:
> Eric W. Biederman wrote:
>> First touch page ownership does not guarantee give me anything useful
>> for knowing if I can run my application or not. Because of page
>> sharing my application might run inside the rss limit only because
>> I got lucky and happened to share a lot of pages with another running
>> application. If the next I run and it isn't running my application

>> will fail. That is ridiculous.
>
> Let's be practical here, what you're asking is basically impossible.
>
> Unless by deterministic you mean that it never enters the a non
> trivial syscall, in which case, you just want to know about maximum
> RSS of the process, which we already account).

If we used Beancounters as Pavel and Kirill mentioned, that would keep track of each container that has referenced a page, not just the first container. It sounds like beancounters can return a usage count where each page is divided by the number of referencing containers (e.g. 1/3rd if 3 containers share a page). Presumably it could also return a full count of 1 to each container.

If we look at data in the latter form, i.e. each container must pay fully for each page used, then Eric could use that to determine real usage needs of the container. However we could also use the fractional count in order to do things such as charging the container for its actual usage. i.e. full count for setting guarantees, fractional for actual usage.

-- Ethan

_____

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