Subject: [PATCH 0/8] RSS controller based on process containers (v2)
Posted by xemul on Mon, 09 Apr 2007 12:18:52 GMT
View Forum Message <> Reply to Message

Adds RSS accounting and control within a container.

Major change: current scanner code reuse.

Tasks and files accounting is not included as these containers
are simple enough to be implemented later.

Based on top of Paul Menage's container subsystem v8.
Note, that only first three patches from Paul's patchset should
be used as BeanCounters, CPU accounting etc are not needed for
RSS container.

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

Changes from previous version:
* reclamation code reuse. I.e. the scanner code used to reclaim
  pages on global memory shortage is used as-is in per-container
  page reclamation;
* active/inactive per-container lists (a bit reworked patch
  from Balbir);
* more elegant top container creation (thanks Paul for clarifying).

Subject: [PATCH 1/8] Resource counters
Posted by xemul on Mon, 09 Apr 2007 12:31:58 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-2.6.20-2/include/linux/res_counter.h
--- linux-2.6.20.orig/include/linux/res_counter.h 2007-04-09 11:26:12.000000000 +0400
+++ linux-2.6.20-2/include/linux/res_counter.h 2007-04-09 11:26:06.000000000 +0400
@@ -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-2.6.20-2/init/Kconfig
--- linux-2.6.20.orig/init/Kconfig 2007-04-09 11:26:06.000000000 +0400
+++ linux-2.6.20-2/init/Kconfig 2007-04-09 11:26:06.000000000 +0400
@@ -253,6 +253,10 @@ 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-2.6.20-2/kernel/Makefile
--- linux-2.6.20.orig/kernel/Makefile 2007-04-09 11:26:06.000000000 +0400
+++ linux-2.6.20-2/kernel/Makefile 2007-04-09 11:26:06.000000000 +0400
```

```
@@ -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-2.6.20-2/kernel/res_counter.c
--- linux-2.6.20.orig/kernel/res_counter.c 2007-04-09 11:26:12.000000000 +0400
+++ linux-2.6.20-2/kernel/res_counter.c 2007-04-09 11:26:06.000000000 +0400
@@ -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: [PATCH 2/8] Add container pointer on struct page
Posted by xemul on Mon, 09 Apr 2007 12:37:43 GMT
View Forum Message <> Reply to Message

Each page is supposed to have an owner - the container
that touched the page first. The owner stays alive during
the page lifetime even if the task that touched the page
dies or moves to another container.

This ownership is the forerunner for the "fair" page sharing

accounting, in which page has as may owners as it is really
used by.

```
diff -upr linux-2.6.20.orig/include/linux/mm.h linux-2.6.20-2/include/linux/mm.h
--- linux-2.6.20.orig/include/linux/mm.h 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/include/linux/mm.h 2007-04-09 11:26:06.000000000 +0400
@@ -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-2.6.20-2/include/linux/mm_types.h
--- linux-2.6.20.orig/include/linux/mm_types.h 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/include/linux/mm_types.h 2007-04-09 11:26:06.000000000 +0400
@@ -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 */
```

Subject: [PATCH 3/8] Add container pointer on mm_struct
Posted by xemul on Mon, 09 Apr 2007 12:43:17 GMT
View Forum Message <> Reply to Message

Naturally mm_struct determines the resource consumer in memory
accounting. So each mm_struct should have a pointer on container
it belongs to. When a new task is created its mm_struct is
assigned to the container this task belongs to.

```
diff -upr linux-2.6.20.orig/include/linux/sched.h linux-2.6.20-2/include/linux/sched.h
--- linux-2.6.20.orig/include/linux/sched.h 2007-04-09 11:26:06.000000000 +0400
+++ linux-2.6.20-2/include/linux/sched.h 2007-04-09 11:26:06.000000000 +0400
@@ -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-2.6.20-2/kernel/fork.c
--- linux-2.6.20.orig/kernel/fork.c 2007-04-09 11:26:06.000000000 +0400
+++ linux-2.6.20-2/kernel/fork.c 2007-04-09 11:26:06.000000000 +0400
@@ -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 +327,7 @@ 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);
@@ -340,11 +342,14 @@ static struct mm_struct * mm_init(struct
  mm->ioctx_list = NULL;
  mm->free_area_cache = TASK_UNMAPPED_BASE;
  mm->cached_hole_size = ~0UL;
+ mm_init_container(mm, tsk);

  if (likely(!mm_alloc_pgd(mm))) {
   mm->def_flags = 0;
   return mm;
  }
+
+ mm_free_container(mm);
  free_mm(mm);
  return NULL;
 }
@@ -359,7 +364,7 @@ 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 +378,7 @@ 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 +499,7 @@ 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 +526,7 @@ fail_nocontext:
   * because it calls destroy_context()
   */
  mm_free_pgd(mm);
+ mm_free_container(mm);
  free_mm(mm);
  return NULL;
 }
```

Subject: [PATCH 4/8] RSS container core
Posted by xemul on Mon, 09 Apr 2007 12:45:19 GMT
View Forum Message <> Reply to Message

This includes
* definition of rss_container as container subsystem combined
  with resource counter;
* registration of RSS container in generic containers;
* routines for pages tracking.

```
diff -upr linux-2.6.20.orig/include/linux/container_subsys.h
linux-2.6.20-2/include/linux/container_subsys.h
--- linux-2.6.20.orig/include/linux/container_subsys.h 2007-04-09 11:26:06.000000000 +0400
+++ linux-2.6.20-2/include/linux/container_subsys.h 2007-04-09 11:26:06.000000000 +0400
@@ -9,6 +9,10 @@
 SUBSYS(cpuset)
```

```
 #endif

+#ifdef CONFIG_RSS_CONTAINER
+SUBSYS(rss)
+#endif
+
 /* */

 /* */
diff -upr linux-2.6.20.orig/include/linux/rss_container.h linux-2.6.20-2/include/linux/rss_container.h
--- linux-2.6.20.orig/include/linux/rss_container.h 2007-04-09 11:26:12.000000000 +0400
+++ linux-2.6.20-2/include/linux/rss_container.h 2007-04-09 11:26:06.000000000 +0400
@@ -0,0 +1,55 @@
+#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 *);
+
+void mm_init_container(struct mm_struct *mm, struct task_struct *tsk);
+void mm_free_container(struct mm_struct *mm);
+
+#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 void mm_init_container(struct mm_struct *mm, struct task_struct *t)
+{
+}
+
+static inline void mm_free_container(struct mm_struct *mm)
+{
+}
+
+#endif
+#endif
diff -upr linux-2.6.20.orig/init/Kconfig linux-2.6.20-2/init/Kconfig
--- linux-2.6.20.orig/init/Kconfig 2007-04-09 11:26:06.000000000 +0400
+++ linux-2.6.20-2/init/Kconfig 2007-04-09 11:26:06.000000000 +0400
@@ -257,6 +257,17 @@ 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 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-2.6.20-2/mm/Makefile
--- linux-2.6.20.orig/mm/Makefile 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/mm/Makefile 2007-04-09 11:26:06.000000000 +0400
@@ -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-2.6.20-2/mm/rss_container.c
--- linux-2.6.20.orig/mm/rss_container.c 2007-04-09 11:26:12.000000000 +0400
+++ linux-2.6.20-2/mm/rss_container.c 2007-04-09 11:26:06.000000000 +0400
@@ -0,0 +1,274 @@
+/*
+ * 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/swap.h>
+#include <linux/res_counter.h>
+#include <linux/rss_container.h>
+
+struct rss_container {
+ struct res_counter res;
+ struct list_head inactive_list;
+ struct list_head active_list;
+ atomic_t rss_reclaimed;
+ 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_id),
+   struct rss_container, css);
+}
+
+void mm_init_container(struct mm_struct *mm, struct task_struct *tsk)
+{
+ struct rss_container *cnt;
+
+ cnt = rss_from_cont(task_container(tsk, rss_subsys_id));
+ css_get(&cnt->css);
```

```
+ mm->rss_container = cnt;
+}
+
+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(&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 (try_to_free_pages_in_container(rss)) {
+   atomic_inc(&rss->rss_reclaimed);
+   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_irq(&rss->res.lock);
+ list_add(&pc->list, &rss->active_list);
+ spin_unlock_irq(&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_irq(&rss->res.lock);
+ list_del(&pc->list);
+ res_counter_uncharge_locked(&rss->res, 1);
+ spin_unlock_irq(&rss->res.lock);
+
+ css_put(&rss->css);
+ kfree(pc);
+}
+
+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(&rss->css);
+ rcu_assign_pointer(mm->rss_container, rss);
+ css_put(&old_rss->css);
+
+out_put:
+ mmput(mm);
+out:
+ return;
+}
+
+static struct rss_container init_rss_container;
+
+static inline void rss_container_attach(struct rss_container *rss,
+  struct container *cont)
+{
+ cont->subsys[rss_subsys_id] = &rss->css;
+ rss->css.container = cont;
+}
+
+static int rss_create(struct container_subsys *ss, struct container *cont)
+{
+ struct rss_container *rss;
+
+ if (unlikely(cont->parent == NULL)) {
+  rss = &init_rss_container;
+  css_get(&rss->css);
+  init_mm.rss_container = rss;
+ } else
+  rss = kzalloc(sizeof(struct rss_container), GFP_KERNEL);
+
+ if (rss == NULL)
+  return -ENOMEM;
+
+ res_counter_init(&rss->res);
+ INIT_LIST_HEAD(&rss->inactive_list);
+ INIT_LIST_HEAD(&rss->active_list);
+ rss_container_attach(rss, cont);
+ 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 ssize_t rss_read_reclaimed(struct container *cont, struct cftype *cft,
+  struct file *file, char __user *userbuf,
+  size_t nbytes, loff_t *ppos)
+{
+ char buf[64], *s;
+
+ s = buf;
+ s += sprintf(s, "%d\n",
+   atomic_read(&rss_from_cont(cont)->rss_reclaimed));
+ return simple_read_from_buffer((void __user *)userbuf, nbytes,
+   ppos, buf, s - buf);
+}
+
+
+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 struct cftype rss_reclaimed = {
+ .name = "rss_reclaimed",
+ .read = rss_read_reclaimed,
+};
+
+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;
+ if ((rc = container_add_file(cont, &rss_reclaimed)) < 0)
+  return rc;
+
+ return 0;
+}
+
+struct container_subsys rss_subsys = {
+ .name = "rss",
+ .subsys_id = rss_subsys_id,
+ .create = rss_create,
+ .destroy = rss_destroy,
+ .populate = rss_populate,
+ .attach = rss_move_task,
+ .early_init = 1,
+};
```

Subject: [PATCH 5/8] RSS accounting hooks over the code
Posted by xemul on Mon, 09 Apr 2007 12:50:21 GMT
View Forum Message <> Reply to Message

As described above, pages are charged to their first touchers.
The first toucher is determined using pages' mapcount manipulations

in rmap calls.

Page is charged in two stages:
1. preparation, in which the resource availability is checked.
   This stage may lead to page reclamation, thus it is performed
   in a might-sleep places;
2. the container assignment to page. This is done in an atomic
   code that handles multiple touches.

```
diff -upr linux-2.6.20.orig/fs/exec.c linux-2.6.20-2/fs/exec.c
--- linux-2.6.20.orig/fs/exec.c 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/fs/exec.c 2007-04-09 11:26:06.000000000 +0400
@@ -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-2.6.20-2/include/linux/rmap.h
--- linux-2.6.20.orig/include/linux/rmap.h 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/include/linux/rmap.h 2007-04-09 11:26:06.000000000 +0400
@@ -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-2.6.20-2/mm/fremap.c
--- linux-2.6.20.orig/mm/fremap.c 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/mm/fremap.c 2007-04-09 11:26:06.000000000 +0400
@@ -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-2.6.20-2/mm/memory.c
--- linux-2.6.20.orig/mm/memory.c 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/mm/memory.c 2007-04-09 11:26:06.000000000 +0400
@@ -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-2.6.20-2/mm/migrate.c
--- linux-2.6.20.orig/mm/migrate.c 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/mm/migrate.c 2007-04-09 11:26:06.000000000 +0400
@@ -28,6 +28,7 @@
 #include <linux/mempolicy.h>
```

```
 #include <linux/vmalloc.h>
 #include <linux/security.h>
+#include <linux/rss_container.h>

 #include "internal.h"

@@ -134,6 +135,7 @@ 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 +159,11 @@ 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 +182,19 @@ 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-2.6.20-2/mm/rmap.c
--- linux-2.6.20.orig/mm/rmap.c 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/mm/rmap.c 2007-04-09 11:26:06.000000000 +0400
@@ -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-2.6.20-2/mm/swapfile.c
--- linux-2.6.20.orig/mm/swapfile.c 2007-03-06 19:09:50.000000000 +0300
```

```
+++ linux-2.6.20-2/mm/swapfile.c 2007-04-09 11:26:06.000000000 +0400
@@ -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: [PATCH 6/8] Per container OOM killer
Posted by xemul on Mon, 09 Apr 2007 12:52:19 GMT
View Forum Message <> Reply to Message

When container is completely out of memory some tasks should
die. This is unfair to kill the current task, so a task with
the largest RSS is chosen and killed. The code re-uses current
OOM killer select_bad_process() for task selection.

```
diff -upr linux-2.6.20.orig/include/linux/rss_container.h linux-2.6.20-2/include/linux/rss_container.h
--- linux-2.6.20.orig/include/linux/rss_container.h 2007-04-09 11:26:12.000000000 +0400
+++ linux-2.6.20-2/include/linux/rss_container.h 2007-04-09 11:26:06.000000000 +0400
@@ -19,6 +19,7 @@
 void container_rss_add(struct page_container *);
 void container_rss_del(struct page_container *);
 void container_rss_release(struct page_container *);
+void container_out_of_memory(struct rss_container *);

 void mm_init_container(struct mm_struct *mm, struct task_struct *tsk);
 void mm_free_container(struct mm_struct *mm);
diff -upr linux-2.6.20.orig/mm/oom_kill.c linux-2.6.20-2/mm/oom_kill.c
--- linux-2.6.20.orig/mm/oom_kill.c 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/mm/oom_kill.c 2007-04-09 11:26:06.000000000 +0400
@@ -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
```

Subject: [PATCH 7/8] Page scanner changes needed to implement per-container scanner
Posted by xemul on Mon, 09 Apr 2007 12:57:22 GMT
View Forum Message <> Reply to Message

Struct scan_control now carries:
* the RSS container to free pages in;
* pointer to an isolate_pages() function to isolate
  pages needed for the reclamation.

```
diff -upr linux-2.6.20.orig/mm/vmscan.c linux-2.6.20-2/mm/vmscan.c
--- linux-2.6.20.orig/mm/vmscan.c 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/mm/vmscan.c 2007-04-09 11:26:06.000000000 +0400
@@ -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;
@@ -66,6 +68,10 @@ struct scan_control {
  int swappiness;
```

```
  int all_unreclaimable;
+ struct rss_container *cnt;
+ unsigned long (*isolate_pages)(unsigned long nr, struct list_head *dst,
+   unsigned long *scanned, struct zone *zone,
+   struct rss_container *cont, int active);
 };

 /*
@@ -654,6 +660,17 @@ static unsigned long isolate_lru_pages(u
  return nr_taken;
 }

+static unsigned long isolate_pages_global(unsigned long nr,
+  struct list_head *dst, unsigned long *scanned,
+  struct zone *z, struct rss_container *cont,
+  int active)
+{
+ if (active)
+  return isolate_lru_pages(nr, &z->active_list, dst, scanned);
+ else
+  return isolate_lru_pages(nr, &z->inactive_list, dst, scanned);
+}
+
 /*
  * shrink_inactive_list() is a helper for shrink_zone().  It returns the number
  * of reclaimed pages
@@ -676,9 +693,9 @@ static unsigned long shrink_inactive_lis
   unsigned long nr_scan;
   unsigned long nr_freed;

-  nr_taken = isolate_lru_pages(sc->swap_cluster_max,
-          &zone->inactive_list,
-          &page_list, &nr_scan);
+  nr_taken = sc->isolate_pages(sc->swap_cluster_max, &page_list,
+    &nr_scan, zone, sc->cnt, 0);
+
   zone->nr_inactive -= nr_taken;
   zone->pages_scanned += nr_scan;
   spin_unlock_irq(&zone->lru_lock);
@@ -822,8 +839,8 @@ force_reclaim_mapped:

  lru_add_drain();
  spin_lock_irq(&zone->lru_lock);
- pgmoved = isolate_lru_pages(nr_pages, &zone->active_list,
-        &l_hold, &pgscanned);
+ pgmoved = sc->isolate_pages(nr_pages, &l_hold, &pgscanned, zone,
+   sc->cnt, 1);
```

```
  zone->pages_scanned += pgscanned;
  zone->nr_active -= pgmoved;
  spin_unlock_irq(&zone->lru_lock);
@@ -1012,7 +1031,9 @@ static unsigned long shrink_zones(int pr
  * holds filesystem locks which prevent writeout this might not work, and the
  * allocation attempt will fail.
  */
-unsigned long try_to_free_pages(struct zone **zones, gfp_t gfp_mask)
+
+static unsigned long do_try_to_free_pages(struct zone **zones, gfp_t gfp_mask,
+  struct scan_control *sc)
 {
  int priority;
  int ret = 0;
@@ -1021,13 +1042,6 @@ unsigned long try_to_free_pages(struct z
  struct reclaim_state *reclaim_state = current->reclaim_state;
  unsigned long lru_pages = 0;
  int i;
- struct scan_control sc = {
-  .gfp_mask = gfp_mask,
-  .may_writepage = !laptop_mode,
-  .swap_cluster_max = SWAP_CLUSTER_MAX,
-  .may_swap = 1,
-  .swappiness = vm_swappiness,
- };

  count_vm_event(ALLOCSTALL);

@@ -1041,17 +1055,18 @@ unsigned long try_to_free_pages(struct z
  }

  for (priority = DEF_PRIORITY; priority >= 0; priority--) {
-  sc.nr_scanned = 0;
+  sc->nr_scanned = 0;
   if (!priority)
    disable_swap_token();
-  nr_reclaimed += shrink_zones(priority, zones, &sc);
-  shrink_slab(sc.nr_scanned, gfp_mask, lru_pages);
+  nr_reclaimed += shrink_zones(priority, zones, sc);
+  if (sc->cnt == NULL)
+   shrink_slab(sc->nr_scanned, gfp_mask, lru_pages);
   if (reclaim_state) {
    nr_reclaimed += reclaim_state->reclaimed_slab;
    reclaim_state->reclaimed_slab = 0;
   }
-  total_scanned += sc.nr_scanned;
-  if (nr_reclaimed >= sc.swap_cluster_max) {
+  total_scanned += sc->nr_scanned;
```

```
+  if (nr_reclaimed >= sc->swap_cluster_max) {
    ret = 1;
    goto out;
   }
@@ -1063,18 +1078,18 @@ unsigned long try_to_free_pages(struct z
    * that's undesirable in laptop mode, where we *want* lumpy
    * writeout.  So in laptop mode, write out the whole world.
    */
-  if (total_scanned > sc.swap_cluster_max +
-     sc.swap_cluster_max / 2) {
+  if (total_scanned > sc->swap_cluster_max +
+     sc->swap_cluster_max / 2) {
    wakeup_pdflush(laptop_mode ? 0 : total_scanned);
-   sc.may_writepage = 1;
+   sc->may_writepage = 1;
   }

   /* Take a nap, wait for some writeback to complete */
-  if (sc.nr_scanned && priority < DEF_PRIORITY - 2)
+  if (sc->nr_scanned && priority < DEF_PRIORITY - 2)
    congestion_wait(WRITE, HZ/10);
  }
  /* top priority shrink_caches still had more to do? don't OOM, then */
- if (!sc.all_unreclaimable)
+ if (!sc->all_unreclaimable)
   ret = 1;
 out:
  /*
@@ -1097,6 +1112,21 @@ out:
  return ret;
 }

+unsigned long try_to_free_pages(struct zone **zones, gfp_t gfp_mask)
+{
+ struct scan_control sc = {
+  .gfp_mask = gfp_mask,
+  .may_writepage = !laptop_mode,
+  .swap_cluster_max = SWAP_CLUSTER_MAX,
+  .may_swap = 1,
+  .swappiness = vm_swappiness,
+  .cnt = NULL,
+  .isolate_pages = isolate_pages_global,
+ };
+
+ return do_try_to_free_pages(zones, gfp_mask, &sc);
+}
+
 /*
```

```
  * For kswapd, balance_pgdat() will work across all this node's zones until
  * they are all at pages_high.
@@ -1131,6 +1190,8 @@ static unsigned long balance_pgdat(pg_da
   .may_swap = 1,
   .swap_cluster_max = SWAP_CLUSTER_MAX,
   .swappiness = vm_swappiness,
+  .cnt = NULL,
+  .isolate_pages = isolate_pages_global,
  };
  /*
   * temp_priority is used to remember the scanning priority at which
@@ -1436,6 +1497,8 @@ unsigned long shrink_all_memory(unsigned
   .swap_cluster_max = nr_pages,
   .may_writepage = 1,
   .swappiness = vm_swappiness,
+  .cnt = NULL,
+  .isolate_pages = isolate_pages_global,
  };

  current->reclaim_state = &reclaim_state;
@@ -1619,6 +1682,8 @@ static int __zone_reclaim(struct zone *z
      SWAP_CLUSTER_MAX),
   .gfp_mask = gfp_mask,
   .swappiness = vm_swappiness,
+  .cnt = NULL,
+  .isolate_pages = isolate_pages_global,
  };
  unsigned long slab_reclaimable;
```

Subject: [PATCH 8/8] Per-container pages reclamation
Posted by xemul on Mon, 09 Apr 2007 12:59:50 GMT

View Forum Message <> Reply to Message

Implement try_to_free_pages_in_container() to free the
pages in container that has run out of memory.

The scan_control->isolate_pages() function isolates the
container pages only.

```
diff -upr linux-2.6.20.orig/include/linux/swap.h linux-2.6.20-2/include/linux/swap.h
--- linux-2.6.20.orig/include/linux/swap.h 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/include/linux/swap.h 2007-04-09 11:26:06.000000000 +0400
@@ -187,6 +187,9 @@ extern void swap_setup(void);

 /* linux/mm/vmscan.c */
 extern unsigned long try_to_free_pages(struct zone **, gfp_t);
+
```

```
+struct rss_container;
+extern unsigned long try_to_free_pages_in_container(struct rss_container *);
 extern unsigned long shrink_all_memory(unsigned long nr_pages);
 extern int vm_swappiness;
 extern int remove_mapping(struct address_space *mapping, struct page *page);
diff -upr linux-2.6.20.orig/mm/vmscan.c linux-2.6.20-2/mm/vmscan.c
--- linux-2.6.20.orig/mm/vmscan.c 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/mm/vmscan.c 2007-04-09 11:26:06.000000000 +0400
@@ -872,6 +872,7 @@ force_reclaim_mapped:
   ClearPageActive(page);

   list_move(&page->lru, &zone->inactive_list);
+  container_rss_move_lists(page, 0);
   pgmoved++;
   if (!pagevec_add(&pvec, page)) {
    zone->nr_inactive += pgmoved;
@@ -900,6 +901,7 @@ force_reclaim_mapped:
   SetPageLRU(page);
   VM_BUG_ON(!PageActive(page));
   list_move(&page->lru, &zone->active_list);
+  container_rss_move_lists(page, 1);
   pgmoved++;
   if (!pagevec_add(&pvec, page)) {
    zone->nr_active += pgmoved;
@@ -1110,6 +1112,35 @@ out:
  return ret;
 }

+#ifdef CONFIG_RSS_CONTAINER
+unsigned long try_to_free_pages_in_container(struct rss_container *cnt)
+{
+ struct scan_control sc = {
+  .gfp_mask = GFP_KERNEL,
+  .may_writepage = 1,
+  .swap_cluster_max = 1,
+  .may_swap = 1,
+  .swappiness = vm_swappiness,
+  .cnt = cnt,
+  .isolate_pages = isolate_pages_in_container,
+ };
+ int node;
+ struct zone **zones;
+
+ for_each_node(node) {
+#ifdef CONFIG_HIGHMEM
+  zones = NODE_DATA(node)->node_zonelists[ZONE_HIGHMEM].zones;
+  if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
+   return 1;
```

```
+#endif
+  zones = NODE_DATA(node)->node_zonelists[ZONE_NORMAL].zones;
+  if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
+   return 1;
+ }
+ return 0;
+}
+#endif
+
 unsigned long try_to_free_pages(struct zone **zones, gfp_t gfp_mask)
 {
  struct scan_control sc = {
diff -upr linux-2.6.20.orig/mm/swap.c linux-2.6.20-2/mm/swap.c
--- linux-2.6.20.orig/mm/swap.c 2007-03-06 19:09:50.000000000 +0300
+++ linux-2.6.20-2/mm/swap.c 2007-04-09 11:26:06.000000000 +0400
@@ -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;
@@ -147,6 +148,7 @@ void fastcall activate_page(struct page
   SetPageActive(page);
   add_page_to_active_list(zone, page);
   __count_vm_event(PGACTIVATE);
+  container_rss_move_lists(page, 1);
  }
  spin_unlock_irq(&zone->lru_lock);
 }

diff -upr linux-2.6.20.orig/mm/rss_container.c linux-2.6.20-2/mm/rss_container.c
--- linux-2.6.20.orig/mm/rss_container.c 2007-04-09 11:26:12.000000000 +0400
+++ linux-2.6.20-2/mm/rss_container.c 2007-04-09 11:26:06.000000000 +0400
@@ -96,6 +96,78 @@
  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);
```

```
+ if (pc == NULL)
+  return;
+
+ rss = pc->cnt;
+
+ spin_lock(&rss->res.lock);
+ if (active)
+  list_move(&pc->list, &rss->active_list);
+ else
+  list_move(&pc->list, &rss->inactive_list);
+ spin_unlock(&rss->res.lock);
+}
+
+static unsigned long isolate_container_pages(unsigned long nr_to_scan,
+  struct list_head *src, struct list_head *dst,
+  unsigned long *scanned, struct zone *zone)
+{
+ unsigned long nr_taken = 0;
+ struct page *page;
+ struct page_container *pc;
+ unsigned long scan;
+ LIST_HEAD(pc_list);
+
+ for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
+  pc = list_entry(src->prev, struct page_container, list);
+  page = pc->page;
+  if (page_zone(page) != zone)
+   continue;
+
+  list_move(&pc->list, &pc_list);
+
+  if (PageLRU(page)) {
+   if (likely(get_page_unless_zero(page))) {
+    ClearPageLRU(page);
+    nr_taken++;
+    list_move(&page->lru, dst);
+   }
+  }
+ }
+
+ list_splice(&pc_list, src);
+
+ *scanned = scan;
+ return nr_taken;
+}
+
+unsigned long isolate_pages_in_container(unsigned long nr_to_scan,
+  struct list_head *dst, unsigned long *scanned,
```

```
+  struct zone *zone, struct rss_container *rss, int active)
+{
+ unsigned long ret;
+
+ spin_lock(&rss->res.lock);
+ if (active)
+  ret = isolate_container_pages(nr_to_scan, &rss->active_list,
+    dst, scanned, zone);
+ else
+  ret = isolate_container_pages(nr_to_scan, &rss->inactive_list,
+    dst, scanned, zone);
+ spin_unlock(&rss->res.lock);
+ return ret;
+}
+
 void container_rss_add(struct page_container *pc)
 {
  struct page *pg;
diff -upr linux-2.6.20.orig/include/linux/rss_container.h linux-2.6.20-2/include/linux/rss_container.h
--- linux-2.6.20.orig/include/linux/rss_container.h 2007-04-09 11:26:12.000000000 +0400
+++ linux-2.6.20-2/include/linux/rss_container.h 2007-04-09 11:26:06.000000000 +0400
@@ -24,6 +24,10 @@
 void mm_init_container(struct mm_struct *mm, struct task_struct *tsk);
 void mm_free_container(struct mm_struct *mm);

+void container_rss_move_lists(struct page *pg, bool active);
+unsigned long isolate_pages_in_container(unsigned long nr_to_scan,
+  struct list_head *dst, unsigned long *scanned,
+  struct zone *zone, struct rss_container *, int active);
 #else
 static inline int container_rss_prepare(struct page *pg,
   struct vm_area_struct *vma, struct page_container **pc)
@@ -52,5 +56,7 @@
 {
 }

+#define isolate_container_pages(nr, dst, scanned, rss, act, zone) ({ BUG(); 0;})
+#define container_rss_move_lists(pg, active) do { } while (0)
 #endif
 #endif
```

Subject: Re: [PATCH 0/8] RSS controller based on process containers (v2)
Posted by Peter Zijlstra on Mon, 09 Apr 2007 15:54:50 GMT
View Forum Message <> Reply to Message

ugh /me no like.

The basic premises seems to be that we can track page owners perfectly
(although this patch set does not yet do so), through get/release
operations (on _mapcount).

This is simply not true for unmapped pagecache pages. Those receive no
'release' event; (the usage by find_get_page() could be seen as 'get').

Also, you don't seem to balance the active/inactive scanning on a per
container basis. This skews the per container working set logic.

Lastly, you don't call the slab shrinker for container reclaim; which
would leave slab reclaim only for those few non process specific
allocations, which would greatly skew the pagecache/slab balance.

Let us call

```
struct reclaim_struct {
 struct list_head active_list;
 struct list_head inactive_list;
 unsigned long nr_active;
 unsigned long nr_inactive;
}
```

Lets recognise three distinct page categories:
 - anonymous memory,
 - mapped pagecache, and
 - unmapped pagecache.

We then keep anonymous pages on a per container reclaim_struct, these
pages are fully accounted to each container.

We keep mapped pagecache pages on per inode reclaim_structs, these files
could be shared between containers and we could either just account all
pages belonging to each file proportional to the number of containers
involved, or do a more precise accounting.

We keep unmapped pagecache pages on a global reclaim_struct, these pages
can, in general, not be pinned to a specific container; all we can do is
keep a floating proportion relative to container 'get' events
(find_get_page() and perhaps add_to_page_cache()).

Reclaim will then have to fairly reclaim pages from all of these lists.
If we schedule such that it appears that these lists are parallel
instead of serial - that is a each tail is really a tail, not the head
of another list - the current reclaim semantics are preserved.

The slab shrinker should be called proportional to the containers size
relative to the machine.

Global reclaim will have to call each container reclaim in proportional
fashion.

The biggest problem with this approach is that there is no per zone
reclaim left, which is relied upon by the allocator to provide free
pages in a given physical address range. However there has been talk to
create a proper range allocator independent of zones.

Just my 0.02 euro..

Peter

Subject: Re: [PATCH 0/8] RSS controller based on process containers (v2)
Posted by xemul on Tue, 10 Apr 2007 08:27:19 GMT
View Forum Message <> Reply to Message

Peter Zijlstra wrote:
> *ugh* /me no like.
>
> The basic premises seems to be that we can track page owners perfectly
> (although this patch set does not yet do so), through get/release

It looks like you have examined the patches not very carefully
before concluding this. These patches DO track page owners.

I know that a page may be shared among several containers and
thus have many owners so we should track all of them. This is
exactly what we decided not to do half-a-year ago.

Page sharing accounting is performed in OpenVZ beancounters, and
this functionality will be pushed to mainline after this simple
container.

> operations (on _mapcount).
>
> This is simply not true for unmapped pagecache pages. Those receive no
> 'release' event; (the usage by find_get_page() could be seen as 'get').

These patches concern the mapped pagecache only. Unmapped pagecache
control is out of the scope of it since we do not want one container
to track all the resources.

> Also, you don't seem to balance the active/inactive scanning on a per

> container basis. This skews the per container working set logic.

This is not true. Balbir sent a patch to the first version of this
container that added active/inactive balancing to the container.
I have included this (a bit reworked) patch into this version and
pointed this fact in the zeroth letter.

> Lastly, you don't call the slab shrinker for container reclaim; which
> would leave slab reclaim only for those few non process specific
> allocations, which would greatly skew the pagecache/slab balance.

Of course I do not call the slab shrinker! We do not have the
kernel memory control yet. Thus we can not shrink arbitrary kernel
objects just because some container has run out of its *user* memory.

Kernel memory control will come later. We decided to start from
a simple RSS control. Please, refer to containers archives for
more details.

>
>
> Let us call
>
> struct reclaim_struct {
> struct list_head active_list;
> struct list_head inactive_list;
> unsigned long nr_active;
> unsigned long nr_inactive;
> }
>
> Lets recognise three distinct page categories:
> - anonymous memory,
> - mapped pagecache, and
> - unmapped pagecache.

We cannot split the user memory in parts. There must be some
overall parameter that will allow administrator to say "Well,
let us run this container in a 64Mb sandbox". With the anonymous
and mapped memory separated administrator will be a bit confused.

>
>
> We then keep anonymous pages on a per container reclaim_struct, these
> pages are fully accounted to each container.

Hmm... We do have such a construction. struct rss_container has
two lists and we shrink from them sequentially using an existing
scanner. Don't forget that this scanner has been evolving for

many years and writing a new scanner is just a waste of time.

> We keep mapped pagecache pages on per inode reclaim_structs, these files
> could be shared between containers and we could either just account all
> pages belonging to each file proportional to the number of containers
> involved, or do a more precise accounting.

What happens if one container fills the RAM with mapped pages from
a single file? Who will be the "owner" of this page set? Who will
expend its IO bandwidth to push these pages on disk? What if this
container will mlock() this set? Who will be killed?

> We keep unmapped pagecache pages on a global reclaim_struct, these pages
> can, in general, not be pinned to a specific container; all we can do is
> keep a floating proportion relative to container 'get' events
> (find_get_page() and perhaps add_to_page_cache()).
>
> Reclaim will then have to fairly reclaim pages from all of these lists.
> If we schedule such that it appears that these lists are parallel
> instead of serial - that is a each tail is really a tail, not the head
> of another list - the current reclaim semantics are preserved.

Yet again. The current scanner came out from the work of many people.
This is a very tricky place that is still evolving. Do you propose
to throw this out and write a new scanner?

> The slab shrinker should be called proportional to the containers size
> relative to the machine.

The slab shrinker must be called only if we do know what kernel
objects are used by this particular container. Otherwise we break
the idea of isolation. Generally speaking if some container runs
out of its resources we should reclaim pages, shrink objects, kill
tasks, etc from this container only.

> Global reclaim will have to call each container reclaim in proportional
> fashion.
>
> The biggest problem with this approach is that there is no per zone
> reclaim left, which is relied upon by the allocator to provide free
> pages in a given physical address range. However there has been talk to
> create a proper range allocator independent of zones.
>
> Just my 0.02 euro..
>
> Peter
>
>

> -
> 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: [PATCH 2/8] Add container pointer on struct page
Posted by Jean-Pierre Dion on Fri, 13 Apr 2007 13:56:37 GMT
View Forum Message <> Reply to Message

Hi Pavel,

I have been implied in the work for the
memory controller of res groups a few months ago.

I see that you propose to modify the struct
page to point to rss container struct.
This has made some debate because of the struct
page size increase, but this allows a quicker
scan to reclaim pages (I mean having per-container
lists of active/inactive pages).
We (here at Bull and others) proposed this implementation
for res groups and I am interested in knowing
if this has a chance of being accepted today (hope so).
I know this uses memory for internal management
and increases a lot the memory size used for
a large memory configuration, but in that case
we have lot of memory, so where is the issue ?
We tested this on a 28 GB server and it worked.
Also we can use larger page size to reduce
the overhead, and I believe this makes sense
on large servers with big memory.

So we balance between using more memory internally
and so getting faster access to pages for reclaim,
or do nothing. ;-)

jean-pierre

Subject: Re: [PATCH 2/8] Add container pointer on struct page
Posted by xemul on Fri, 13 Apr 2007 14:48:00 GMT
View Forum Message <> Reply to Message

Jean-Pierre Dion wrote:
> Hi Pavel,
>
> I have been implied in the work for the
> memory controller of res groups a few months ago.
>
> I see that you propose to modify the struct
> page to point to rss container struct.
> This has made some debate because of the struct
> page size increase, but this allows a quicker
> scan to reclaim pages (I mean having per-container
> lists of active/inactive pages).
> We (here at Bull and others) proposed this implementation
> for res groups and I am interested in knowing
> if this has a chance of being accepted today (hope so).

So do I :) I'm not the one who makes the final decision ;)

> I know this uses memory for internal management
> and increases a lot the memory size used for
> a large memory configuration, but in that case
> we have lot of memory, so where is the issue ?
> We tested this on a 28 GB server and it worked.

Thank you for additional testing on enterprise servers!
Hope this will be a good argument in favour of the patches.

> Also we can use larger page size to reduce
> the overhead, and I believe this makes sense
> on large servers with big memory.
>
> So we balance between using more memory internally
> and so getting faster access to pages for reclaim,
> or do nothing. ;-)

That's right. I made some small testing which showed
that moving this pointer in a mirrored array saves less
than 0.1% of performance on 4CPU i386 node. I don't know
how this will be on enterprise hardware, but I do believe
that the results will be the same (or even better).

>
> jean-pierre
>
>
> -
> 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: [PATCH 0/8] RSS controller based on process containers (v2)
Posted by Vaidyanathan Srinivas on Thu, 19 Apr 2007 05:37:14 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Peter Zijlstra wrote:
>> *ugh* /me no like.
>>
>> The basic premises seems to be that we can track page owners perfectly
>> (although this patch set does not yet do so), through get/release
>
> It looks like you have examined the patches not very carefully
> before concluding this. These patches DO track page owners.
>
> I know that a page may be shared among several containers and
> thus have many owners so we should track all of them. This is
> exactly what we decided not to do half-a-year ago.
>
> Page sharing accounting is performed in OpenVZ beancounters, and
> this functionality will be pushed to mainline after this simple
> container.
>
>> operations (on _mapcount).
>>
>> This is simply not true for unmapped pagecache pages. Those receive no
>> 'release' event; (the usage by find_get_page() could be seen as 'get').
>
> These patches concern the mapped pagecache only. Unmapped pagecache
> control is out of the scope of it since we do not want one container
> to track all the resources.

Unmapped pagecache control and swapcache control is part of
independent pagecache controller that is being developed. Initial
version was posted at http://lkml.org/lkml/2007/3/06/51
I plan to post a new version based on this patchset in a couple of days.

--Vaidy

>> Also, you don't seem to balance the active/inactive scanning on a per
>> container basis. This skews the per container working set logic.
>
> This is not true. Balbir sent a patch to the first version of this
> container that added active/inactive balancing to the container.

> I have included this (a bit reworked) patch into this version and
> pointed this fact in the zeroth letter.
>

[snip]

Subject: Re: [PATCH 8/8] Per-container pages reclamation
Posted by Balbir Singh on Tue, 24 Apr 2007 09:47:25 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Implement try_to_free_pages_in_container() to free the
> pages in container that has run out of memory.
>
> The scan_control->isolate_pages() function isolates the
> container pages only.
>

Pavel,

I've just started playing around with these patches, I preferred
the approach of v1. Please see below

> +static unsigned long isolate_container_pages(unsigned long nr_to_scan,
> + struct list_head *src, struct list_head *dst,
> + unsigned long *scanned, struct zone *zone)
> +{
> + unsigned long nr_taken = 0;
> + struct page *page;
> + struct page_container *pc;
> + unsigned long scan;
> + LIST_HEAD(pc_list);
> +
> + for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
> + pc = list_entry(src->prev, struct page_container, list);
> + page = pc->page;
> + if (page_zone(page) != zone)
> +   continue;

shrink_zone() will walk all pages looking for pages belonging to this
container and this slows down the reclaim quite a bit. Although we've
reused code, we've ended up walking the entire list of the zone to
find pages belonging to a particular container, this was the same
problem I had with my RSS controller patches.

> +
> + list_move(&pc->list, &pc_list);

> +

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

Subject: Re: [PATCH 8/8] Per-container pages reclamation
Posted by xemul on Tue, 24 Apr 2007 10:29:58 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelianov wrote:
>> Implement try_to_free_pages_in_container() to free the
>> pages in container that has run out of memory.
>>
>> The scan_control->isolate_pages() function isolates the
>> container pages only.
>>
>
> Pavel,
>
> I've just started playing around with these patches, I preferred
> the approach of v1. Please see below
>
>> +static unsigned long isolate_container_pages(unsigned long nr_to_scan,
>> +		struct list_head *src, struct list_head *dst,
>> +		unsigned long *scanned, struct zone *zone)
>> +{
>> +	unsigned long nr_taken = 0;
>> +	struct page *page;
>> +	struct page_container *pc;
>> +	unsigned long scan;
>> +	LIST_HEAD(pc_list);
>> +
>> +	for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
>> +		pc = list_entry(src->prev, struct page_container, list);
>> +		page = pc->page;
>> +		if (page_zone(page) != zone)
>> +			continue;
>
> shrink_zone() will walk all pages looking for pages belonging to this

No. shrink_zone() will walk container pages looking for pages in the desired zone.
Scann through the full zone is done on global memory shortage.

> container and this slows down the reclaim quite a bit. Although we've
> reused code, we've ended up walking the entire list of the zone to
> find pages belonging to a particular container, this was the same
> problem I had with my RSS controller patches.
>
>> +
>> +	list_move(&pc->list, &pc_list);
>> +
>
>

Subject: Re: [PATCH 8/8] Per-container pages reclamation
Posted by Balbir Singh on Tue, 24 Apr 2007 11:01:07 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Balbir Singh wrote:
>> Pavel Emelianov wrote:
>>> Implement try_to_free_pages_in_container() to free the
>>> pages in container that has run out of memory.
>>>
>>> The scan_control->isolate_pages() function isolates the
>>> container pages only.
>>>
>> Pavel,
>>
>> I've just started playing around with these patches, I preferred
>> the approach of v1. Please see below
>>
>>> +static unsigned long isolate_container_pages(unsigned long nr_to_scan,
>>> +		struct list_head *src, struct list_head *dst,
>>> +		unsigned long *scanned, struct zone *zone)
>>> +{
>>> +	unsigned long nr_taken = 0;
>>> +	struct page *page;
>>> +	struct page_container *pc;
>>> +	unsigned long scan;
>>> +	LIST_HEAD(pc_list);
>>> +
>>> +	for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
>>> +		pc = list_entry(src->prev, struct page_container, list);
>>> +		page = pc->page;
>>> +		if (page_zone(page) != zone)
>>> +			continue;
>> shrink_zone() will walk all pages looking for pages belonging to this
>

> No. shrink_zone() will walk container pages looking for pages in the desired zone.
> Scann through the full zone is done on global memory shortage.
>

Yes, I see that now. But for each zone in the system, we walk through the
containers list - right?

I have some more fixes, improvements that I want to send across.
I'll start sending them out to you as I test and verify them.

>> container and this slows down the reclaim quite a bit. Although we've
>> reused code, we've ended up walking the entire list of the zone to
>> find pages belonging to a particular container, this was the same
>> problem I had with my RSS controller patches.
>>
>>> +
>>> +	list_move(&pc->list, &pc_list);
>>> +
>>
>

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

Subject: Re: [PATCH 8/8] Per-container pages reclamation
Posted by xemul on Tue, 24 Apr 2007 11:37:38 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelianov wrote:
>> Balbir Singh wrote:
>>> Pavel Emelianov wrote:
>>>> Implement try_to_free_pages_in_container() to free the
>>>> pages in container that has run out of memory.
>>>>
>>>> The scan_control->isolate_pages() function isolates the
>>>> container pages only.
>>>>
>>> Pavel,
>>>
>>> I've just started playing around with these patches, I preferred
>>> the approach of v1. Please see below

>>>
>>>> +static unsigned long isolate_container_pages(unsigned long nr_to_scan,
>>>> +   struct list_head *src, struct list_head *dst,
>>>> +   unsigned long *scanned, struct zone *zone)
>>>> +{
>>>> +   unsigned long nr_taken = 0;
>>>> +   struct page *page;
>>>> +   struct page_container *pc;
>>>> +   unsigned long scan;
>>>> +   LIST_HEAD(pc_list);
>>>> +
>>>> +   for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
>>>> +   pc = list_entry(src->prev, struct page_container, list);
>>>> +   page = pc->page;
>>>> +   if (page_zone(page) != zone)
>>>> +   continue;
>>> shrink_zone() will walk all pages looking for pages belonging to this
>>
>> No. shrink_zone() will walk container pages looking for pages in the
>> desired zone.
>> Scann through the full zone is done on global memory shortage.
>>
>
> Yes, I see that now. But for each zone in the system, we walk through the
> containers list - right?

Right.

> I have some more fixes, improvements that I want to send across.
> I'll start sending them out to you as I test and verify them.

That's great! :) Thanks for participation.

>
>>> container and this slows down the reclaim quite a bit. Although we've
>>> reused code, we've ended up walking the entire list of the zone to
>>> find pages belonging to a particular container, this was the same
>>> problem I had with my RSS controller patches.
>>>
>>>> +
>>>> +   list_move(&pc->list, &pc_list);
>>>> +
>>>
>>
>
>

Subject: Re: [PATCH 8/8] Per-container pages reclamation
Posted by Balbir Singh on Wed, 02 May 2007 09:51:03 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Implement try_to_free_pages_in_container() to free the
> pages in container that has run out of memory.
>
> The scan_control->isolate_pages() function isolates the
> container pages only.
>
> +#ifdef CONFIG_RSS_CONTAINER
> +unsigned long try_to_free_pages_in_container(struct rss_container *cnt)
> +{
> + struct scan_control sc = {
> + .gfp_mask = GFP_KERNEL,
> + .may_writepage = 1,
> + .swap_cluster_max = 1,
> + .may_swap = 1,
> + .swappiness = vm_swappiness,
> + .cnt = cnt,
> + .isolate_pages = isolate_pages_in_container,
> + };
> + int node;
> + struct zone **zones;
> +
> + for_each_node(node) {
> +#ifdef CONFIG_HIGHMEM
> + zones = NODE_DATA(node)->node_zonelists[ZONE_HIGHMEM].zones;
> + if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
> + return 1;
> +#endif
> + zones = NODE_DATA(node)->node_zonelists[ZONE_NORMAL].zones;
> + if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
> + return 1;
> + }
> + return 0;
> +}
> +#endif
> +

Hi, Pavel,

This patch fixes a bug in the RSS controller, where we walk through all
nodes during reclaim in try_to_free_pages_in_container(). Instead of
for_each_node(), we now use for_each_online_node() so that we do not
end up with invalid zones from nodes that are not online.

Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>

Signed-off-by: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
---

 mm/vmscan.c | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

diff -puN mm/vmscan.c~rss-fix-nodescan mm/vmscan.c
--- linux-2.6.20/mm/vmscan.c~rss-fix-nodescan 2007-05-02 14:47:09.000000000
+0530
+++ linux-2.6.20-balbir/mm/vmscan.c 2007-05-02 14:47:25.000000000 +0530
@@ -1127,7 +1127,7 @@ unsigned long try_to_free_pages_in_conta
  int node;
  struct zone **zones;

- for_each_node(node) {
+ for_each_online_node(node) {
 #ifdef CONFIG_HIGHMEM
   zones = NODE_DATA(node)->node_zonelists[ZONE_HIGHMEM].zones;
   if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
_

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

Subject: Re: [PATCH 8/8] Per-container pages reclamation
Posted by Balbir Singh on Thu, 17 May 2007 11:31:26 GMT

View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Implement try_to_free_pages_in_container() to free the
> pages in container that has run out of memory.
>
> The scan_control->isolate_pages() function isolates the
> container pages only.
>
>

Hi, Pavel/Andrew,

I've started running some basic tests like lmbench and LTP vm stress
on the RSS controller.

With the controller rss_limit set to 256 MB, I saw the following panic
on a machine

Unable to handle kernel NULL pointer dereference at 000000000000001c RIP:
 [<ffffffff80328581>] _raw_spin_lock+0xd/0xf6
PGD 3c841067 PUD 5d5d067 PMD 0
Oops: 0000 [1] SMP
CPU 2
Modules linked in: ipv6 hidp rfcomm l2cap bluetooth sunrpc video button battery asus_acpi backlight ac lp parport_pc parport nvram pcspkr amd_rng rng_core i2c_amd756 i2c_core
Pid: 13581, comm: mtest01 Not tainted 2.6.20-autokern1 #1
RIP: 0010:[<ffffffff80328581>] [<ffffffff80328581>] _raw_spin_lock+0xd/0xf6
RSP: 0000:ffff81003e6c9ce8 EFLAGS: 00010096
RAX: ffffffff8087f720 RBX: 0000000000000018 RCX: ffff81003f36f9d0
RDX: ffff8100807bb040 RSI: 0000000000000001 RDI: 0000000000000018
RBP: 0000000000000000 R08: ffff81003e6c8000 R09: 0000000000000002
R10: ffff810001021da8 R11: ffffffff8044658f R12: ffff81000c861e01
R13: 0000000000000018 R14: ffff81000c861eb8 R15: ffff810032d34138
FS: 00002abf7a1961e0(0000) GS:ffff81003edb94c0(0000) knlGS:0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 000000008005003b
CR2: 000000000000001c CR3: 000000002ba6e000 CR4: 00000000000006e0
Process mtest01 (pid: 13581, threadinfo ffff81003e6c8000, task ffff81003d8ec040)
Stack: ffff810001003638 ffff810014a8c2c0 0000000000000000 ffff81000c861e01
 0000000000000018 ffffffff80287166 ffff81000c861eb8 ffff81000000bac0
 ffff81003f36f9a0 ffff81000c861e40 ffff81001d4b6a20 ffffffff8026a92e
Call Trace:
 [<ffffffff80287166>] container_rss_move_lists+0x3b/0xaf
 [<ffffffff8026a92e>] activate_page+0xc1/0xd0
 [<ffffffff80245f15>] wake_bit_function+0x0/0x23
 [<ffffffff8026ab34>] mark_page_accessed+0x1b/0x2f
 [<ffffffff80265d25>] filemap_nopage+0x180/0x338
 [<ffffffff80270474>] __handle_mm_fault+0x1f2/0xa81
 [<ffffffff804c58ef>] do_page_fault+0x42b/0x7b3
 [<ffffffff802484c4>] hrtimer_cancel+0xc/0x16
 [<ffffffff804c2a89>] do_nanosleep+0x47/0x70
 [<ffffffff802485f4>] hrtimer_nanosleep+0x58/0x119
 [<ffffffff8023bc1f>] sys_sysinfo+0x15b/0x173
 [<ffffffff804c3d3d>] error_exit+0x0/0x84

On analyzing the code, I found that the page is mapped (we have a page_mapped() check in container_rss_move_lists()), but the page_container is invalid. Please review the fix attached (we reset the page's container pointer to NULL when a page is completely unmapped)

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

Subject: Re: [PATCH 8/8] Per-container pages reclamation
Posted by xemul on Mon, 21 May 2007 15:15:32 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelianov wrote:
>> Implement try_to_free_pages_in_container() to free the
>> pages in container that has run out of memory.
>>
>> The scan_control->isolate_pages() function isolates the
>> container pages only.

Sorry for the late answer, but I have just managed to get
to the patches. One comment is below.

>>
>
> Hi, Pavel/Andrew,
>
> I've started running some basic tests like lmbench and LTP vm stress
> on the RSS controller.
>
> With the controller rss_limit set to 256 MB, I saw the following panic
> on a machine
>
> Unable to handle kernel NULL pointer dereference at 000000000000001c RIP:
> [<ffffffff80328581>] _raw_spin_lock+0xd/0xf6
> PGD 3c841067 PUD 5d5d067 PMD 0
> Oops: 0000 [1] SMP
> CPU 2
> Modules linked in: ipv6 hidp rfcomm l2cap bluetooth sunrpc video button battery asus_acpi
backlight ac lp parport_pc parport nvram pcspkr amd_rng rng_core i2c_amd756 i2c_core
> Pid: 13581, comm: mtest01 Not tainted 2.6.20-autokern1 #1
> RIP: 0010:[<ffffffff80328581>] [<ffffffff80328581>] _raw_spin_lock+0xd/0xf6
> RSP: 0000:ffff81003e6c9ce8 EFLAGS: 00010096
> RAX: ffffffff8087f720 RBX: 0000000000000018 RCX: ffff81003f36f9d0
> RDX: ffff8100807bb040 RSI: 0000000000000001 RDI: 0000000000000018
> RBP: 0000000000000000 R08: ffff81003e6c8000 R09: 0000000000000002
> R10: ffff810001021da8 R11: ffffffff8044658f R12: ffff81000c861e01
> R13: 0000000000000018 R14: ffff81000c861eb8 R15: ffff810032d34138
> FS: 00002abf7a1961e0(0000) GS:ffff81003edb94c0(0000) knlGS:0000000000000000
> CS: 0010 DS: 0000 ES: 0000 CR0: 000000008005003b
> CR2: 000000000000001c CR3: 000000002ba6e000 CR4: 00000000000006e0
> Process mtest01 (pid: 13581, threadinfo ffff81003e6c8000, task ffff81003d8ec040)
> Stack: ffff810001003638 ffff810014a8c2c0 0000000000000000 ffff81000c861e01
> 0000000000000018 ffffffff80287166 ffff81000c861eb8 ffff81000000bac0
> ffff81003f36f9a0 ffff81000c861e40 ffff81001d4b6a20 ffffffff8026a92e
> Call Trace:
> [<ffffffff80287166>] container_rss_move_lists+0x3b/0xaf

> [<ffffffff8026a92e>] activate_page+0xc1/0xd0
> [<ffffffff80245f15>] wake_bit_function+0x0/0x23
> [<ffffffff8026ab34>] mark_page_accessed+0x1b/0x2f
> [<ffffffff80265d25>] filemap_nopage+0x180/0x338
> [<ffffffff80270474>] __handle_mm_fault+0x1f2/0xa81
> [<ffffffff804c58ef>] do_page_fault+0x42b/0x7b3
> [<ffffffff802484c4>] hrtimer_cancel+0xc/0x16
> [<ffffffff804c2a89>] do_nanosleep+0x47/0x70
> [<ffffffff802485f4>] hrtimer_nanosleep+0x58/0x119
> [<ffffffff8023bc1f>] sys_sysinfo+0x15b/0x173
> [<ffffffff804c3d3d>] error_exit+0x0/0x84
>
> On analyzing the code, I found that the page is mapped (we have a page_mapped() check in
> container_rss_move_lists()), but the page_container is invalid. Please review the fix
> attached (we reset the page's container pointer to NULL when a page is completely unmapped)
>
>
>
> -- ------------
>
> Index: linux-2.6.20/mm/rss_container.c
> == =======
> --- linux-2.6.20.orig/mm/rss_container.c 2007-05-15 05:13:46.000000000 -0700
> +++ linux-2.6.20/mm/rss_container.c 2007-05-16 20:45:45.000000000 -0700
> @@ -212,6 +212,7 @@ void container_rss_del(struct page_conta
>
>   css_put(&rss->css);
>   kfree(pc);
> + init_page_container(page);

This hunk is bad.
See, when the page drops its mapcount to 0 it may be reused right
after this if it belongs to a file map - another CPU can touch it.
Thus you're risking to reset the wrong container.

The main idea if the accounting is that you cannot trust the
page_container(page) value after the page's mapcount became 0.

> }
>
> static void rss_move_task(struct container_subsys *ss,
> Index: linux-2.6.20/mm/page_alloc.c
> == =======
> --- linux-2.6.20.orig/mm/page_alloc.c 2007-05-16 10:30:10.000000000 -0700
> +++ linux-2.6.20/mm/page_alloc.c 2007-05-16 20:45:24.000000000 -0700
> @@ -41,6 +41,7 @@
> #include <linux/pfn.h>
> #include <linux/backing-dev.h>

> #include <linux/fault-inject.h>
> +#include <linux/rss_container.h>
>
> #include <asm/tlbflush.h>
> #include <asm/div64.h>
> @@ -1977,6 +1978,7 @@ void __meminit memmap_init_zone(unsigned
>   set_page_links(page, zone, nid, pfn);
>   init_page_count(page);
>   reset_page_mapcount(page);
> + init_page_container(page);
>   SetPageReserved(page);
>   INIT_LIST_HEAD(&page->lru);
> #ifdef WANT_PAGE_VIRTUAL
> Index: linux-2.6.20/include/linux/rss_container.h
> ===
> --- linux-2.6.20.orig/include/linux/rss_container.h 2007-05-16 10:31:04.000000000 -0700
> +++ linux-2.6.20/include/linux/rss_container.h 2007-05-16 10:32:14.000000000 -0700
> @@ -28,6 +28,11 @@ void container_rss_move_lists(struct pag
> unsigned long isolate_pages_in_container(unsigned long nr_to_scan,
>   struct list_head *dst, unsigned long *scanned,
>   struct zone *zone, struct rss_container *, int active);
> +static inline void init_page_container(struct page *page)
> +{
> + page_container(page) = NULL;
> +}
> +
> #else
> static inline int container_rss_prepare(struct page *pg,
>   struct vm_area_struct *vma, struct page_container **pc)
> @@ -56,6 +61,10 @@ static inline void mm_free_container(str
> {
> }
>
> +static inline void init_page_container(struct page *page)
> +{
> +}
> +
> #define isolate_container_pages(nr, dst, scanned, rss, act, zone) ({ BUG(); 0;})
> #define container_rss_move_lists(pg, active) do { } while (0)
> #endif

Subject: Re: [PATCH 8/8] Per-container pages reclamation
Posted by Balbir Singh on Thu, 24 May 2007 07:59:55 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
>> Index: linux-2.6.20/mm/rss_container.c

>> === =======
>> --- linux-2.6.20.orig/mm/rss_container.c 2007-05-15 05:13:46.000000000 -0700
>> +++ linux-2.6.20/mm/rss_container.c 2007-05-16 20:45:45.000000000 -0700
>> @@ -212,6 +212,7 @@ void container_rss_del(struct page_conta
>>
>> css_put(&rss->css);
>> kfree(pc);
>> + init_page_container(page);
>
> This hunk is bad.
> See, when the page drops its mapcount to 0 it may be reused right
> after this if it belongs to a file map - another CPU can touch it.
> Thus you're risking to reset the wrong container.
>
> The main idea if the accounting is that you cannot trust the
> page_container(page) value after the page's mapcount became 0.
>

Good catch, I'll move the initialization to free_hot_cold_page().
I'm attaching a new patch. I've also gotten rid of the unused
variable page in container_rss_del().

I've compile and boot tested the fix

--
 Thanks,
 Balbir Singh
 Linux Technology Center
 IBM, ISTL