Subject: [PATCH 0/8] RSS controller based on process containers (v3.1)
Posted by Pavel Emelianov on Mon, 04 Jun 2007 13:21:01 GMT
View Forum Message <> Reply to Message

Adds RSS accounting and control within a container.

Changes from v3
 - comments across the code
 - git-bisect safe split
 - lost places to move the page between active/inactive lists

Ported above Paul's containers V10 with fixes from Balbir.

RSS container includes the per-container RSS accounting
and reclamation, and out-of-memory killer.

Each mapped page has an owning container and is linked into its
LRU lists just like in the global LRU ones. The owner of the page
is the container that touched the page first. As long as the page
stays mapped it holds the container, is accounted into its usage
and lives in its LRU list. When page is unmapped for the last time
it releases the container. The RSS usage is exactly the number of
pages in its booth LRU lists, i.e. the nu,ber of pages used by this
container. When this usage exceeds the limit set some pages are
reclaimed from the owning container. In case no reclamation possible
the OOM killer starts thinning out the container.

Thus 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.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

The testing scenario may look like this:

1. Prepare the containers
mkdir -p /containers/rss
mount -t container none /containers/rss -o rss

2. Make the new group and move bash into it
mkdir /containers/rss/0
echo $$ > /containers/rss/0/tasks

Since now we're in the 0 container.
We can alter the RSS limit
echo -n 6000 > /containers/rss/0/rss_limit

We can check the usage
cat /containers/rss/0/rss_usage
25

And do other stuff. To check the reclamation to work we need a
simple program that touches many pages of memory, like this:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>

#ifndef PGSIZE
#define PGSIZE  4096
#endif

int main(int argc, char **argv)
{
	unsigned long pages;
	int i;
	char *mem;

	if (argc < 2) {
		printf("Usage: %s <number_of_pages>\n", argv[0]);
		return 1;
	}

	pages = strtol(argv[1], &mem, 10);
	if (*mem != '\0') {
		printf("Bad number %s\n", argv[1]);
		return 1;
	}

	mem = mmap(NULL, pages * PGSIZE, PROT_READ | PROT_WRITE,
			MAP_PRIVATE | MAP_ANON, 0, 0);
	if (mem == MAP_FAILED) {
		perror("map");
		return 2;
	}

	for (i = 0; i < pages; i++)
		mem[i * PGSIZE] = 0;

	printf("OK\n");
	return 0;
}
```

Subject: [PATCH 1/8] Resource counters
Posted by Pavel Emelianov on Mon, 04 Jun 2007 13:24:37 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.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

---

diff -upr linux-2.6.22-rc2-mm1.orig/include/linux/res_counter.h
linux-2.6.22-rc2-mm1-2/include/linux/res_counter.h
--- linux-2.6.22-rc2-mm1.orig/include/linux/res_counter.h 2007-06-01 16:46:32.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/include/linux/res_counter.h 2007-06-01 16:35:15.000000000 +0400
@@ -0,0 +1,102 @@
+#ifndef __RES_COUNTER_H__
+#define __RES_COUNTER_H__
+
+/*
+ * resource counters
+ * contain common data types and routines for resource accounting
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+#include <linux/container.h>
+
+/*
+ * the core object. the container that wishes to account for some
+ * resource may include this counter into its structures and use
+ * the helpers described beyond
+ */
+
+struct res_counter {
+ /*
+ * the current resource consumption level
+ */
+ unsigned long usage;
+ /*
+ * the limit that usage cannot exceed
+ */
+ unsigned long limit;
+ /*

```
+  * the number of insuccessful attempts to consume the resource
+  */
+ unsigned long failcnt;
+ /*
+  * the lock to protect all of the above.
+  * the routines below consider this to be IRQ-safe
+  */
+ spinlock_t lock;
+};
+
+/*
+ * helpers to interact with userspace
+ * res_counter_read/_write - put/get the specified fields from the
+ * res_counter struct to/from the user
+ *
+ * @cnt:     the counter in question
+ * @member:  the field to work with (see RES_xxx below)
+ * @buf:     the buffer to opeate on,...
+ * @nbytes:  its size...
+ * @pos:     and the offset.
+ */
+
+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);
+
+/*
+ * the field descriptors. one for each member of res_counter
+ */
+
+enum {
+ RES_USAGE,
+ RES_LIMIT,
+ RES_FAILCNT,
+};
+
+/*
+ * helpers for accounting
+ */
+
+void res_counter_init(struct res_counter *cnt);
+
+/*
+ * charge - try to consume more resource.
+ *
+ * @cnt: the counter
+ * @val: the amount of the resource. each controller defines its own
```

```
+ *       units, e.g. numbers, bytes, Kbytes, etc
+ *
+ * returns 0 on success and <0 if the cnt->usage will exceed the cnt->limit
+ * _locked call expects the cnt->lock to be taken
+ */
+
+int res_counter_charge_locked(struct res_counter *cnt, unsigned long val);
+int res_counter_charge(struct res_counter *cnt, unsigned long val);
+
+/*
+ * uncharge - tell that some portion of the resource is released
+ *
+ * @cnt: the counter
+ * @val: the amount of the resource
+ *
+ * these calls check for usage underflow and show a warning on the console
+ * _locked call expects the cnt->lock to be taken
+ */
+
+void res_counter_uncharge_locked(struct res_counter *cnt, unsigned long val);
+void res_counter_uncharge(struct res_counter *cnt, unsigned long val);
+
+#endif
diff -upr linux-2.6.22-rc2-mm1.orig/init/Kconfig linux-2.6.22-rc2-mm1-2/init/Kconfig
--- linux-2.6.22-rc2-mm1.orig/init/Kconfig 2007-06-01 16:35:12.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/init/Kconfig 2007-06-01 16:35:13.000000000 +0400
@@ -328,6 +328,10 @@ 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-2.6.22-rc2-mm1.orig/kernel/Makefile linux-2.6.22-rc2-mm1-2/kernel/Makefile
--- linux-2.6.22-rc2-mm1.orig/kernel/Makefile 2007-06-01 16:35:12.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/kernel/Makefile 2007-06-01 16:35:13.000000000 +0400
@@ -56,6 +56,7 @@ obj-$(CONFIG_SYSCTL) += utsname_sysctl.o
 obj-$(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-2.6.22-rc2-mm1.orig/kernel/res_counter.c
linux-2.6.22-rc2-mm1-2/kernel/res_counter.c
--- linux-2.6.22-rc2-mm1.orig/kernel/res_counter.c 2007-06-01 16:46:32.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/kernel/res_counter.c 2007-06-01 16:35:15.000000000 +0400
@@ -0,0 +1,121 @@
+/*
+ * resource containers
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+#include <linux/types.h>
+#include <linux/parser.h>
+#include <linux/fs.h>
+#include <linux/res_counter.h>
+#include <linux/uaccess.h>
+
+void res_counter_init(struct res_counter *cnt)
+{
+ spin_lock_init(&cnt->lock);
+ cnt->limit = (unsigned long)LONG_MAX;
+}
+
+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;
+}
+
+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;
+}
+
```

```
+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;
+}
+
+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);
+}
+
+
+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 Pavel Emelianov on Mon, 04 Jun 2007 13:26:25 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.

Page ownership getter/setter are done as static inline function.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

```
---

diff -upr linux-2.6.22-rc2-mm1.orig/include/linux/mm.h linux-2.6.22-rc2-mm1-2/include/linux/mm.h
--- linux-2.6.22-rc2-mm1.orig/include/linux/mm.h 2007-05-30 12:32:35.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/include/linux/mm.h 2007-06-01 16:35:15.000000000 +0400
@@ -249,6 +249,30 @@ struct vm_operations_struct {

 struct mmu_gather;
 struct inode;
+struct page_container;
+
+#ifdef CONFIG_RSS_CONTAINER
+static inline struct page_container *page_container(struct page *pg)
+{
+ return pg->rss_container;
+}
+
+static inline void set_page_container(struct page *pg,
+  struct page_container *container)
+{
+ pg->rss_container = container;
+}
+#else
+static inline struct page_container *page_container(struct page *pg)
+{
+ return NULL;
+}
+
+static inline void set_page_container(struct page *pg,
+  struct page_container *container)
+{
+}
+#endif

 #define page_private(page)  ((page)->private)
 #define set_page_private(page, v) ((page)->private = (v))
diff -upr linux-2.6.22-rc2-mm1.orig/include/linux/mm_types.h
linux-2.6.22-rc2-mm1-2/include/linux/mm_types.h
--- linux-2.6.22-rc2-mm1.orig/include/linux/mm_types.h 2007-05-30 12:32:35.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/include/linux/mm_types.h 2007-06-01 16:35:13.000000000 +0400
@@ -83,6 +83,9 @@ struct page {
  unsigned int gfp_mask;
  unsigned long trace[8];
 #endif
+#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 Pavel Emelianov on Mon, 04 Jun 2007 13:29:49 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.

include/linux/rss_container.h is added in this patch to make
this valid apart from the whole set, but the content of it is
just a set of stubs.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

---

diff -upr linux-2.6.22-rc2-mm1.orig/include/linux/sched.h
linux-2.6.22-rc2-mm1-2/include/linux/sched.h
--- linux-2.6.22-rc2-mm1.orig/include/linux/sched.h 2007-06-01 16:35:12.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/include/linux/sched.h 2007-06-01 16:35:13.000000000 +0400
@@ -390,6 +390,9 @@ 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-2.6.22-rc2-mm1.orig/include/linux/rss_container.h
linux-2.6.22-rc2-mm1-2/include/linux/rss_container.h
--- linux-2.6.22-rc2-mm1.orig/include/linux/rss_container.h 2007-06-01 16:46:32.000000000
+0400
+++ linux-2.6.22-rc2-mm1-2/include/linux/rss_container.h 2007-06-01 16:35:15.000000000 +0400
@@ -0,0 +1,19 @@
+#ifndef __RSS_CONTAINER_H__
+#define __RSS_CONTAINER_H__
+/*
+ * RSS container
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc

```
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
+ */
+
+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
diff -upr linux-2.6.22-rc2-mm1.orig/kernel/fork.c linux-2.6.22-rc2-mm1-2/kernel/fork.c
--- linux-2.6.22-rc2-mm1.orig/kernel/fork.c 2007-06-01 16:35:12.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/kernel/fork.c 2007-06-01 16:35:13.000000000 +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)
  */
@@ -329,7 +331,7 @@ 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);
@@ -344,11 +346,14 @@ 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;
```

```
 }
@@ -363,7 +368,7 @@ 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;
 }
@@ -377,6 +382,7 @@ 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);
 }
@@ -497,7 +503,7 @@ 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))
@@ -524,6 +530,7 @@ fail_nocontext:
   * because it calls destroy_context()
   */
  mm_free_pgd(mm);
+ mm_free_container(mm);
  free_mm(mm);
  return NULL;
 }
```

Subject: [PATCH 4/8] Scanner changes needed to implement per-container scanner
Posted by Pavel Emelianov on Mon, 04 Jun 2007 13:31:37 GMT
View Forum Message <> Reply to Message

The core change is that the isolate_lru_pages() call is
replaced with struct scan_controll->isolate_pages() call.

Other changes include exporting __isolate_lru_page() for
per-container isolator and handling variable-to-pointer
changes in try_to_free_pages().

This makes possible to use different isolation routines
for per-container page reclamation. This will be used by
the following patch.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

```
diff -upr linux-2.6.22-rc2-mm1.orig/include/linux/swap.h
linux-2.6.22-rc2-mm1-2/include/linux/swap.h
--- linux-2.6.22-rc2-mm1.orig/include/linux/swap.h 2007-05-30 12:32:36.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/include/linux/swap.h 2007-06-01 16:35:13.000000000 +0400
@@ -192,6 +192,11 @@ extern void swap_setup(void);
 /* linux/mm/vmscan.c */
 extern unsigned long try_to_free_pages(struct zone **zones, int order,
      gfp_t gfp_mask);
+
+struct rss_container;
+extern unsigned long try_to_free_pages_in_container(struct rss_container *);
+int __isolate_lru_page(struct page *page, int mode);
+
 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-2.6.22-rc2-mm1.orig/mm/vmscan.c linux-2.6.22-rc2-mm1-2/mm/vmscan.c
--- linux-2.6.22-rc2-mm1.orig/mm/vmscan.c 2007-05-30 12:32:36.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/mm/vmscan.c 2007-06-01 16:35:15.000000000 +0400
@@ -47,6 +47,8 @@

 #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;
@@ -70,6 +72,13 @@ struct scan_control {
  int all_unreclaimable;

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

```
 #define lru_to_page(_head) (list_entry((_head)->prev, struct page, lru))
@@ -622,7 +631,7 @@ keep:
  *
  * returns 0 on success, -ve errno on failure.
  */
-static int __isolate_lru_page(struct page *page, int mode)
+int __isolate_lru_page(struct page *page, int mode)
 {
  int ret = -EINVAL;

@@ -774,6 +783,19 @@ static unsigned long clear_active_flags(
  return nr_active;
 }

+static unsigned long isolate_pages_global(unsigned long nr,
+  struct list_head *dst, unsigned long *scanned,
+  int order, int mode, struct zone *z,
+  struct rss_container *cont, int active)
+{
+ if (active)
+  return isolate_lru_pages(nr, &z->active_list,
+    dst, scanned, order, mode);
+ else
+  return isolate_lru_pages(nr, &z->inactive_list,
+    dst, scanned, order, mode);
+}
+
 /*
  * shrink_inactive_list() is a helper for shrink_zone().  It returns the number
  * of reclaimed pages
@@ -797,11 +819,11 @@ static unsigned long shrink_inactive_lis
   unsigned long nr_freed;
   unsigned long nr_active;

-  nr_taken = isolate_lru_pages(sc->swap_cluster_max,
-        &zone->inactive_list,
-        &page_list, &nr_scan, sc->order,
-        (sc->order > PAGE_ALLOC_COSTLY_ORDER)?
-         ISOLATE_BOTH : ISOLATE_INACTIVE);
+  nr_taken = sc->isolate_pages(sc->swap_cluster_max, &page_list,
+    &nr_scan, sc->order,
+    (sc->order > PAGE_ALLOC_COSTLY_ORDER) ?
+     ISOLATE_BOTH : ISOLATE_INACTIVE,
+    zone, sc->container, 0);
   nr_active = clear_active_flags(&page_list);

   __mod_zone_page_state(zone, NR_ACTIVE, -nr_active);
```

```
@@ -950,8 +975,8 @@ force_reclaim_mapped:

  lru_add_drain();
  spin_lock_irq(&zone->lru_lock);
- pgmoved = isolate_lru_pages(nr_pages, &zone->active_list,
-       &l_hold, &pgscanned, sc->order, ISOLATE_ACTIVE);
+ pgmoved = sc->isolate_pages(nr_pages, &l_hold, &pgscanned,
+   sc->order, ISOLATE_ACTIVE, zone, sc->container, 1);
  zone->pages_scanned += pgscanned;
  __mod_zone_page_state(zone, NR_ACTIVE, -pgmoved);
  spin_unlock_irq(&zone->lru_lock);
@@ -1142,7 +1169,8 @@ 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, int order, gfp_t gfp_mask)
+unsigned long do_try_to_free_pages(struct zone **zones, gfp_t gfp_mask,
+  struct scan_control *sc)
 {
  int priority;
  int ret = 0;
@@ -1151,14 +1179,6 @@ 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,
-  .order = order,
- };

  delay_swap_prefetch();
  count_vm_event(ALLOCSTALL);
@@ -1174,17 +1194,23 @@ 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->container == NULL)
```

```
+   /*
+    * we do not shrink slabs as this won't unmap any
+    * pages from user and thus won't help him to charge
+    * more pages  --xemul
+    */
+   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;
    }
@@ -1196,18 +1222,23 @@ 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)
+ /*
+  * top priority shrink_caches still had more to do? don't OOM, then
+  *
+  * we cannot hope shrink_caches to help us in case we are trying
+  * to shrink container pages  --xemul
+  */
+ if (!sc->all_unreclaimable && sc->container == NULL)
    ret = 1;
  out:
   /*
@@ -1230,6 +1261,22 @@ out:
```

```
  return ret;
 }

+unsigned long try_to_free_pages(struct zone **zones, int order, 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,
+  .order = order,
+  .container = 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.
@@ -1265,6 +1347,8 @@ static unsigned long balance_pgdat(pg_da
   .swap_cluster_max = SWAP_CLUSTER_MAX,
   .swappiness = vm_swappiness,
   .order = order,
+  .container = NULL,
+  .isolate_pages = isolate_pages_global,
  };
  /*
   * temp_priority is used to remember the scanning priority at which
@@ -1604,6 +1688,8 @@ unsigned long shrink_all_memory(unsigned
   .swap_cluster_max = nr_pages,
   .may_writepage = 1,
   .swappiness = vm_swappiness,
+  .container = NULL,
+  .isolate_pages = isolate_pages_global,
  };

  delay_swap_prefetch();
@@ -1789,6 +1875,8 @@ static int __zone_reclaim(struct zone *z
      SWAP_CLUSTER_MAX),
   .gfp_mask = gfp_mask,
   .swappiness = vm_swappiness,
+  .container = NULL,
+  .isolate_pages = isolate_pages_global,
  };
  unsigned long slab_reclaimable;
```

Subject: [PATCH 5/8] RSS container core
Posted by Pavel Emelianov on Mon, 04 Jun 2007 13:35:07 GMT
View Forum Message <> Reply to Message

The core routines for tracking the page ownership, RSS subsystem
registration in the containers and the definition of the
rss_container struct as container subsystem combined with the
resource counter structure.

To make the whole set look more consistent the calls to the
reclamation code and oom killer are removed from this patch,
so if someone has the kernel snapshot stopped at this patch
and calls the accounting routines and the limit will be hit,
the current task will be killed immediately.

Includes fixes from Balbir Singh <balbir@in.ibm.com>

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

```
--- ./include/linux/container_subsys.h.rsscore 2007-06-04 12:05:30.000000000 +0400
+++ ./include/linux/container_subsys.h 2007-06-04 12:06:27.000000000 +0400
@@ -11,6 +11,10 @@
 SUBSYS(cpuacct)
 #endif

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

 #ifdef CONFIG_CPUSETS
--- ./include/linux/rss_container.h.rsscore 2007-06-04 12:06:06.000000000 +0400
+++ ./include/linux/rss_container.h 2007-06-04 12:06:27.000000000 +0400
@@ -9,6 +9,93 @@
  *
  */

+struct page_container;
+struct rss_container;
+
+#ifdef CONFIG_RSS_CONTAINER
+/*
+ * This is how the RSS accounting works.
+ *
+ * Abstract:
+ *  Each mapped page has an owning container and is linked into its LRU lists
```

```
+ *  just like in the global LRU ones. The owner of the page is the container
+ *  that touched the page first. As long as the page stays mapped it holds
+ *  the container, is accounted into its usage and lives in its LRU list.
+ *  When page is unmapped for the last time it releases the container. The
+ *  RSS usage is exactly the number of pages in its booth LRU lists. When
+ *  this usage exceeds the limit set some pages are reclaimed from the owning
+ *  container. In case no reclamation possible the OOM killer starts thinning
+ *  out the container.
+ *
+ *
+ * The details:
+ *  The mapping-to-user process is splitted into 3 stages:
+ *  1. The page allocation and charging. This stage is might-sleep one and
+ *     in case the container has run out of resources the reclamation code
+ *     starts.
+ *
+ *    container_rss_prepare() fultion prepares the page container to add it
+ *    to the rss container and charges the resource counter.
+ *
+ *  2. The page tables tuning. This stage is atomic (as it is always done under
+ *     the ptl lock) and handles the race between multiple mappers.
+ *
+ *    container_rss_add() assigns the container to the page and adds it to
+ *    the container's LRU list.
+ *
+ *  3. The release/error path. In case of any error after prepare or race
+ *     with another toucher the page container must be released.
+ *
+ *    container_rss_release() frees the preallocated container and uncharges
+ *    the now-unneded amount from the counter
+ *
+ *  The unmapping process is simpe:
+ *     When page is unmapped from the last address space it releases the
+ *     container and is removed from its LRU lists. This is importaint that
+ *     the page stays in the global LRU list till it is completely freed
+ *
+ *    container_rss_del() removes the container from the page
+ *
+ *
+ * The "mapped for the first time" and "unmapped from the last user" conditions
+ * are checkin in rmap calls.
+ *   - If mapcount changes form 0 to 1 the page is first touched and is added
+ *     to the container;
+ *   - If mapcount changes from 1 to 0 the page is unmapped for the last time
+ *     and is removed from the container.
+ *
+ * See comments in mm/rmap.c for more details.
+ *
```

```
+ */
+
+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)
 {
 }
@@ -17,3 +103,4 @@ static inline void mm_free_container(str
 {
 }
 #endif
+#endif
--- ./init/Kconfig.rsscore 2007-06-04 12:05:48.000000000 +0400
+++ ./init/Kconfig 2007-06-04 12:06:27.000000000 +0400
@@ -332,6 +332,17 @@ config RESOURCE_COUNTERS
  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
--- ./mm/Makefile.rsscore 2007-06-04 12:05:26.000000000 +0400
+++ ./mm/Makefile 2007-06-04 12:06:27.000000000 +0400
@@ -30,4 +30,5 @@ obj-$(CONFIG_FS_XIP) += filemap_xip.o
 obj-$(CONFIG_MIGRATION) += migrate.o
 obj-$(CONFIG_SMP) += allocpercpu.o
 obj-$(CONFIG_QUICKLIST) += quicklist.o
+obj-$(CONFIG_RSS_CONTAINER) += rss_container.o

--- /dev/null 2007-04-11 18:31:29.344197500 +0400
+++ ./mm/rss_container.c 2007-06-04 12:06:27.000000000 +0400
@@ -0,0 +1,323 @@
+/*
+ * RSS accounting container
+ *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ *
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ * Fixes from: Balbir Singh <balbir@in.ibm.com>
+ *
+ */
+
+#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>
+
+/*
+ * the rss container description
+ */
+
+struct rss_container {
+ /*
+  * the counter to account for RSS
+  */
+ struct res_counter res;
+ /*
```

```
+  * the lists of pages within the container.
+  * actually these lists store the page containers (see below), not
+  * pages. they live just like the global LRU lists do. i.e. the
+  * page containers migrate from one list to another as soon as the
+  * according page does. see container_rss_move_lists and comment
+  * in include/linux/rss_container.h for more details
+  *
+  * these lists are protected with res_counter's lock and this lock
+  * may be taken under zone->lru_lock
+  */
+ struct list_head inactive_list;
+ struct list_head active_list;
+ /*
+  * the number of successful reclaims from the container. reclamation
+  * happens each time the counter reports that no resource available,
+  * so low value in comparison to the failcnt will indicate that the
+  * container is experiencing difficulties during its lifetime and is
+  * probably misconfigured
+  */
+ atomic_t rss_reclaimed;
+
+ struct container_subsys_state css;
+};
+
+/*
+ * page container ties together a page and a container making a one-to-one
+ * relations between them. see comment in inlude/linux/rss_container.h on
+ * how this relation is set
+ */
+
+struct page_container {
+ struct page *page;
+ struct rss_container *cnt;
+ struct list_head list; /* this is the element of (int)active_list of
+    * the container.
+    */
+};
+
+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);
+}
+
+/*
+ * mm_struct represents the page consumer, so we store the container pointer
+ * on them as well and use it in accounting
+ */
```

```
+
+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);
+}
+
+/*
+ * ownership tracking.
+ * I bet you have already read the comment in include/linux/rss_container.h :)
+ */
+
+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;
+
+ if (res_counter_charge(&rss->res, 1))
+  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);
+
+ set_page_container(pg, pc);
+}
+
+void container_rss_del(struct page_container *pc)
+{
+ struct rss_container *rss;
+
+ 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);
+}
+
+/*
+ * interaction with the containers subsystem
+ */
+
+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);
+
+ /*
+  * tasks may share one mm_struct with each other. on the other hand
+  * tasks may belong to different containers. keep mm_structs tied
+  * to the according task and migrate them together
+  */
+
+ 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);
+}
```

```
+
+/*
+ * container files to export RSS container's counter state to userspace
+ * one file for each field
+ */
+
+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 6/8] Per container OOM killer
Posted by Pavel Emelianov on Mon, 04 Jun 2007 13:40:17 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.

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

```
--- ./include/linux/rss_container.h.rssoom 2007-06-04 12:06:27.000000000 +0400
+++ ./include/linux/rss_container.h 2007-06-04 12:21:11.000000000 +0400
@@ -73,6 +73,8 @@ void container_rss_add(struct page_conta
 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);
 #else
--- ./mm/oom_kill.c.rssoom 2007-06-04 12:05:26.000000000 +0400
+++ ./mm/oom_kill.c 2007-06-04 12:21:11.000000000 +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.
   */
@@ -204,7 +213,8 @@ 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;
@@ -258,7 +268,7 @@ 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;
@@ -444,7 +454,7 @@ 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;
```

```
@@ -473,3 +483,27 @@ 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] Per-container pages reclamation
Posted by Pavel Emelianov on Mon, 04 Jun 2007 13:41:58 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 is set to
isolate_pages_in_container() that isolates the container
pages only. The exported __isolate_lru_page() call
makes things look simpler than in the previous version.

Includes fix from Balbir Singh <balbir@in.ibm.com>

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

```
--- ./include/linux/rss_container.h.rssreclaim 2007-06-04 12:21:11.000000000 +0400
+++ ./include/linux/rss_container.h 2007-06-04 12:22:38.000000000 +0400
@@ -77,6 +77,12 @@ void container_out_of_memory(struct rss_

 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,
+  int order, int mode, 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)
@@ -104,5 +110,8 @@ static inline void mm_init_container(str
 static inline void mm_free_container(struct mm_struct *mm)
 {
 }
+
+#define isolate_container_pages(nr, dst, scanned, rss, act, zone) ({ BUG(); 0;})
+#define container_rss_move_lists(pg, active) do { } while (0)
 #endif
 #endif
--- ./mm/rss_container.c.rssreclaim 2007-06-04 12:06:27.000000000 +0400
+++ ./mm/rss_container.c 2007-06-04 12:22:38.000000000 +0400
@@ -108,8 +108,16 @@ int container_rss_prepare(struct page *p
  if (pc == NULL)
   goto out_nomem;

- if (res_counter_charge(&rss->res, 1))
-  goto out_charge;
+ 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;
@@ -163,6 +171,95 @@ void container_rss_del(struct page_conta
 }
```

```
 /*
+ * reclamation code helpers
+ */
+
+/*
+ * Move the page across the active/inactive lists.
+ * It is called when we want to move a page in the LRU list. This could happen
+ * when a page is activated or when reclaim finds that a particular page cannot
+ * be reclaimed right now.  --balbir
+ */
+
+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;
+
+ /* we are called with zone->lru_lock held with irqs disabled */
+ 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, int mode)
+{
+ 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;
+  /*
```

```
+   * TODO: now we hold all the pages in one... ok, two lists
+   * and skip the pages from another zones with the check
+   * below.  this in not very good - try to make these lists
+   * per-zone to optimize this loop
+   */
+  if (page_zone(page) != zone)
+   continue;
+
+  list_move(&pc->list, &pc_list);
+
+  if (__isolate_lru_page(page, mode) == 0) {
+   list_move(&page->lru, dst);
+   nr_taken++;
+  }
+ }
+
+ 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,
+  int order, int mode, struct zone *zone,
+  struct rss_container *rss, int active)
+{
+ unsigned long ret;
+
+ /* we are called with zone->lru_lock held with irqs disabled */
+ spin_lock(&rss->res.lock);
+ if (active)
+  ret = isolate_container_pages(nr_to_scan, &rss->active_list,
+    dst, scanned, zone, mode);
+ else
+  ret = isolate_container_pages(nr_to_scan, &rss->inactive_list,
+    dst, scanned, zone, mode);
+ spin_unlock(&rss->res.lock);
+ return ret;
+}
+
+/*
  * interaction with the containers subsystem
  */

--- ./mm/swap.c.rssreclaim 2007-06-04 12:05:26.000000000 +0400
+++ ./mm/swap.c 2007-06-04 12:22:38.000000000 +0400
@@ -31,6 +31,7 @@
```

```
 #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;
@@ -128,6 +129,7 @@ int rotate_reclaimable_page(struct page
  if (PageLRU(page) && !PageActive(page)) {
   list_move_tail(&page->lru, &zone->inactive_list);
   __count_vm_event(PGROTATED);
+  container_rss_move_lists(page, 0);
  }
  if (!test_clear_page_writeback(page))
   BUG();
@@ -148,6 +150,7 @@ 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);
 }
--- ./mm/vmscan.c.rssreclaim 2007-06-04 12:06:15.000000000 +0400
+++ ./mm/vmscan.c 2007-06-04 12:22:38.000000000 +0400
@@ -855,10 +855,13 @@ static unsigned long shrink_inactive_lis
    VM_BUG_ON(PageLRU(page));
    SetPageLRU(page);
    list_del(&page->lru);
-   if (PageActive(page))
+   if (PageActive(page)) {
     add_page_to_active_list(zone, page);
-   else
+    container_rss_move_lists(page, 1);
+   } else {
     add_page_to_inactive_list(zone, page);
+    container_rss_move_lists(page, 0);
+   }
    if (!pagevec_add(&pvec, page)) {
     spin_unlock_irq(&zone->lru_lock);
     __pagevec_release(&pvec);
@@ -1005,6 +1008,7 @@ force_reclaim_mapped:
   ClearPageActive(page);

   list_move(&page->lru, &zone->inactive_list);
+  container_rss_move_lists(page, 0);
   pgmoved++;
   if (!pagevec_add(&pvec, page)) {
```

```
    __mod_zone_page_state(zone, NR_INACTIVE, pgmoved);
@@ -1033,6 +1037,7 @@ 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)) {
    __mod_zone_page_state(zone, NR_ACTIVE, pgmoved);
@@ -1272,6 +1277,41 @@ unsigned long try_to_free_pages(struct z
  return do_try_to_free_pages(zones, gfp_mask, &sc);
 }

+#ifdef CONFIG_RSS_CONTAINER
+/*
+ * user pages can be allocated from any zone starting from ZONE_HIGHMEM
+ * so try to shrink pages from them all
+ */
+#ifdef CONFIG_HIGHMEM
+#define ZONE_USERPAGES ZONE_HIGHMEM
+#else
+#define ZONE_USERPAGES ZONE_NORMAL
+#endif
+
+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,
+  .order = 0, /* in this case we wanted one page only */
+  .container = cnt,
+  .isolate_pages = isolate_pages_in_container,
+ };
+ int node;
+ struct zone **zones;
+
+ for_each_online_node(node) {
+  zones = NODE_DATA(node)->node_zonelists[ZONE_USERPAGES].zones;
+  if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
+   return 1;
+ }
+ return 0;
+}
+#endif
+
```

```
 /*
  * For kswapd, balance_pgdat() will work across all this node's zones until
  * they are all at pages_high.
```

Subject: [PATCH 8/8] RSS accounting hooks over the code
Posted by Pavel Emelianov on Mon, 04 Jun 2007 13:45:40 GMT
View Forum Message <> Reply to Message

As described, pages are charged to their first touchers.
The first toucher is determined using pages' _mapcount
manipulations in rmap calls.

A 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 races between multiple touchers.

The uncharge process is tricky. After the _mapcount is changed
and happens to hit zero we cannot trust the page->container
pointer any longer. So we get the pointer before the decrement.

Includes fixes from Balbir Singh <balbir@in.ibm.com>

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

```
diff -upr linux-2.6.22-rc2-mm1.orig/fs/exec.c linux-2.6.22-rc2-mm1-2/fs/exec.c
--- linux-2.6.22-rc2-mm1.orig/fs/exec.c 2007-05-30 12:32:34.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/fs/exec.c 2007-06-01 16:35:13.000000000 +0400
@@ -59,6 +59,8 @@
 #include <linux/kmod.h>
 #endif

+#include <linux/rss_container.h>
+
 int core_uses_pid;
 char core_pattern[CORENAME_MAX_SIZE] = "core";
 int suid_dumpable = 0;
@@ -314,27 +316,34 @@ 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-2.6.22-rc2-mm1.orig/include/linux/rmap.h
linux-2.6.22-rc2-mm1-2/include/linux/rmap.h
--- linux-2.6.22-rc2-mm1.orig/include/linux/rmap.h 2007-05-29 18:01:57.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/include/linux/rmap.h 2007-06-01 16:35:13.000000000 +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 *);

 #ifdef CONFIG_DEBUG_VM
diff -upr linux-2.6.22-rc2-mm1.orig/mm/memory.c linux-2.6.22-rc2-mm1-2/mm/memory.c
--- linux-2.6.22-rc2-mm1.orig/mm/memory.c 2007-05-30 12:32:36.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/mm/memory.c 2007-06-01 16:35:13.000000000 +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;
@@ -1155,7 +1157,7 @@ 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);
@@ -1263,7 +1265,7 @@ 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;
@@ -1668,6 +1670,7 @@ 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)
@@ -1752,6 +1755,9 @@ 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
    */
@@ -1779,12 +1785,14 @@ 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)
@@ -2176,6 +2184,7 @@ 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;
@@ -2208,6 +2217,11 @@ 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);
@@ -2221,6 +2235,7 @@ static int do_swap_page(struct mm_struct

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

```
 }

@@ -2235,7 +2250,7 @@ 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())
@@ -2256,6 +2271,7 @@ unlock:
 out:
  return ret;
 out_nomap:
+ container_rss_release(pcont);
  pte_unmap_unlock(page_table, ptl);
  unlock_page(page);
  page_cache_release(page);
@@ -2274,6 +2290,7 @@ 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. */
@@ -2285,15 +2302,19 @@ 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);
@@ -2305,7 +2326,7 @@ 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);
@@ -2316,9 +2337,14 @@ 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;
 }
@@ -2346,6 +2372,7 @@ static int __do_fault(struct mm_struct *
  int anon = 0;
  struct page *dirty_page = NULL;
  struct fault_data fdata;
+ struct page_container *pcont;

  fdata.address = address & PAGE_MASK;
  fdata.pgoff = pgoff;
@@ -2415,6 +2442,11 @@ static int __do_fault(struct mm_struct *

  }

+ if (container_rss_prepare(page, vma, &pcont)) {
+  fdata.type = VM_FAULT_OOM;
+  goto out;
+ }
+
  page_table = pte_offset_map_lock(mm, pmd, address, &ptl);

  /*
@@ -2437,10 +2469,10 @@ static int __do_fault(struct mm_struct *
   if (anon) {
                        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 {
    inc_mm_counter(mm, file_rss);
-   page_add_file_rmap(page);
+   page_add_file_rmap(page, pcont);
    if (flags & FAULT_FLAG_WRITE) {
     dirty_page = page;
     get_page(dirty_page);
@@ -2451,6 +2483,7 @@ static int __do_fault(struct mm_struct *
   update_mmu_cache(vma, address, entry);
   lazy_mmu_prot_update(entry);
  } else {
+  container_rss_release(pcont);
   if (anon)
    page_cache_release(page);
   else
diff -upr linux-2.6.22-rc2-mm1.orig/mm/migrate.c linux-2.6.22-rc2-mm1-2/mm/migrate.c
--- linux-2.6.22-rc2-mm1.orig/mm/migrate.c 2007-05-30 12:32:36.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/mm/migrate.c 2007-06-01 16:35:13.000000000 +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-2.6.22-rc2-mm1.orig/mm/page_alloc.c linux-2.6.22-rc2-mm1-2/mm/page_alloc.c
--- linux-2.6.22-rc2-mm1.orig/mm/page_alloc.c 2007-05-30 12:32:36.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/mm/page_alloc.c 2007-06-01 16:35:15.000000000 +0400
@@ -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>
@@ -1042,6 +1043,7 @@ static void fastcall free_hot_cold_page(

  if (!PageHighMem(page))
   debug_check_no_locks_freed(page_address(page), PAGE_SIZE);
+ set_page_container(page, NULL);
  arch_free_page(page, 0);
  kernel_map_pages(page, 1, 0);

@@ -2592,6 +2594,7 @@ void __meminit memmap_init_zone(unsigned
   set_page_links(page, zone, nid, pfn);
   init_page_count(page);
   reset_page_mapcount(page);
+  set_page_container(page, NULL);
   SetPageReserved(page);
```

```
   /*
diff -upr linux-2.6.22-rc2-mm1.orig/mm/rmap.c linux-2.6.22-rc2-mm1-2/mm/rmap.c
--- linux-2.6.22-rc2-mm1.orig/mm/rmap.c 2007-05-30 12:32:36.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/mm/rmap.c 2007-06-01 16:35:15.000000000 +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)
@@ -563,18 +565,28 @@ static void __page_check_anon_rmap(struc
  * @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 and the page must be locked.
  */
 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)
 {
  VM_BUG_ON(!PageLocked(page));
  VM_BUG_ON(address < vma->vm_start || address >= vma->vm_end);
- if (atomic_inc_and_test(&page->_mapcount))
+ if (atomic_inc_and_test(&page->_mapcount)) {
   __page_set_anon_rmap(page, vma, address);
- else
+  /* 0 -> 1 state change */
+  container_rss_add(pcont);
+ } else {
   __page_check_anon_rmap(page, vma, address);
+  /*
+   * we raced with another touch or just mapped the page
+   * for the N-th time
+   */
+  container_rss_release(pcont);
+ }
 }

 /*
@@ -582,29 +594,44 @@ 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.
  * Page does not have to be locked.
  */
 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)
 {
  BUG_ON(address < vma->vm_start || address >= vma->vm_end);
  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)
+   /*
+    * 0 -> 1 state change. add the page
+    *
+    * some pages like ZERO_PAGE() or driver specific pages
+    * are not added to the container as they do not imply
+    * RSS consumption  --xemul
+    */
+   container_rss_add(pcont);
   __inc_zone_page_state(page, NR_FILE_MAPPED);
+ } else if (pcont)
+  container_rss_release(pcont);
 }

 #ifdef CONFIG_DEBUG_VM
@@ -635,6 +662,15 @@ void page_dup_rmap(struct page *page, st
```

```
 */
 void page_remove_rmap(struct page *page, struct vm_area_struct *vma)
 {
+ struct page_container *pcont;
+
+ /*
+  * this place is tricky. we get the container pointer *before* the
+  * mapcount change. this is done so sice we cannot trust this pointer
+  * *after* mapcount drops to zero - the page can be re-mapped right
+  * after this so we risk getting the wrong container  --xemul
+  */
+ 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));
@@ -652,6 +688,8 @@ 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-2.6.22-rc2-mm1.orig/mm/swapfile.c linux-2.6.22-rc2-mm1-2/mm/swapfile.c
--- linux-2.6.22-rc2-mm1.orig/mm/swapfile.c 2007-05-11 16:36:58.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/mm/swapfile.c 2007-06-01 16:35:13.000000000 +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: Re: [PATCH 0/8] RSS controller based on process containers (v3.1)
Posted by Herbert Poetzl on Fri, 08 Jun 2007 12:03:43 GMT
View Forum Message <> Reply to Message

On Mon, Jun 04, 2007 at 05:25:25PM +0400, Pavel Emelianov wrote:
> Adds RSS accounting and control within a container.
>
> Changes from v3
> - comments across the code
> - git-bisect safe split
> - lost places to move the page between active/inactive lists
>
> Ported above Paul's containers V10 with fixes from Balbir.
>

> RSS container includes the per-container RSS accounting
> and reclamation, and out-of-memory killer.
>
>
> Each mapped page has an owning container and is linked into its
> LRU lists just like in the global LRU ones. The owner of the page
> is the container that touched the page first.

> As long as the page stays mapped it holds the container, is accounted
> into its usage and lives in its LRU list. When page is unmapped for
> the last time it releases the container.

> The RSS usage is exactly the number of pages in its booth LRU lists,
> i.e. the nu,ber of pages used by this container.

so there could be two guests, unified (i.e. sharing
most of the files as hardlinks), where the first one
holds 80% of the resulting pages, and the second one
20%, and thus shows much lower 'RSS' usage as the
other one, although it is running the very same
processes and providing identical services?

> When this usage exceeds the limit set some pages are reclaimed from
> the owning container. In case no reclamation possible the OOM killer
> starts thinning out the container.

so the system (physical machine) starts reclaiming
and probably swapping even when there is no need
to do so?

e.g. a system with a single guest, limited to 10k
pages, with a working set of 15k pages in different
apps would continuously swap (trash?) on an otherwise
unused (100k+ pages) system?

> Thus 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.

is that really what we want?
I think we can do _better_ than a standalone machine
and in many cases we really should ...

best,
Herbert

> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
>

> The testing scenario may look like this:
>
> 1. Prepare the containers
> # mkdir -p /containers/rss
> # mount -t container none /containers/rss -o rss
>
> 2. Make the new group and move bash into it
> # mkdir /containers/rss/0
> # echo $$ > /containers/rss/0/tasks
>
> Since now we're in the 0 container.
> We can alter the RSS limit
> # echo -n 6000 > /containers/rss/0/rss_limit
>
> We can check the usage
> # cat /containers/rss/0/rss_usage
> 25
>
> And do other stuff. To check the reclamation to work we need a
> simple program that touches many pages of memory, like this:
>
> #include <stdio.h>
> #include <unistd.h>
> #include <sys/mman.h>
> #include <fcntl.h>
>
> #ifndef PGSIZE
> #define PGSIZE 4096
> #endif
>
> int main(int argc, char **argv)
> {
>   unsigned long pages;
>   int i;
>   char *mem;
>
>   if (argc < 2) {
>   printf("Usage: %s <number_of_pages>\n", argv[0]);
>   return 1;
>   }
>
>   pages = strtol(argv[1], &mem, 10);
>   if (*mem != '\0') {
>   printf("Bad number %s\n", argv[1]);
>   return 1;
>   }
>
>   mem = mmap(NULL, pages * PGSIZE, PROT_READ | PROT_WRITE,

```
>                       MAP_PRIVATE | MAP_ANON, 0, 0);
>       if (mem == MAP_FAILED) {
>               perror("map");
>               return 2;
>       }
>
>       for (i = 0; i < pages; i++)
>               mem[i * PGSIZE] = 0;
>
>       printf("OK\n");
>       return 0;
> }
> _____
> 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: [PATCH 0/8] RSS controller based on process containers (v3.1)
Posted by Pavel Emelianov on Fri, 08 Jun 2007 12:39:28 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Mon, Jun 04, 2007 at 05:25:25PM +0400, Pavel Emelianov wrote:
>> Adds RSS accounting and control within a container.
>>
>> Changes from v3
>> - comments across the code
>> - git-bisect safe split
>> - lost places to move the page between active/inactive lists
>>
>> Ported above Paul's containers V10 with fixes from Balbir.
>>
>> RSS container includes the per-container RSS accounting
>> and reclamation, and out-of-memory killer.
>>
>>
>> Each mapped page has an owning container and is linked into its
>> LRU lists just like in the global LRU ones. The owner of the page
>> is the container that touched the page first.
>
>> As long as the page stays mapped it holds the container, is accounted
>> into its usage and lives in its LRU list. When page is unmapped for
>> the last time it releases the container.

>
>> The RSS usage is exactly the number of pages in its booth LRU lists,
>> i.e. the nu,ber of pages used by this container.
>
> so there could be two guests, unified (i.e. sharing
> most of the files as hardlinks), where the first one
> holds 80% of the resulting pages, and the second one
> 20%, and thus shows much lower 'RSS' usage as the
> other one, although it is running the very same
> processes and providing identical services?

Herbert!!! Where have you been so long?

You must have missed that we've decided not to account pages
sharing right now, but start with that model. Later we'll make
sharing accountable.

(There's something bad with my memory. I have a feeling that I
  have already told that... many times...)

>> When this usage exceeds the limit set some pages are reclaimed from
>> the owning container. In case no reclamation possible the OOM killer
>> starts thinning out the container.
>
> so the system (physical machine) starts reclaiming
> and probably swapping even when there is no need
> to do so?

Good catch! The system will start reclaiming right when the
container hits the limit to expend its IO bandwidth. Not some
other's one that hit the global limit due to some bad container
was allowed to go above it.

> e.g. a system with a single guest, limited to 10k
> pages, with a working set of 15k pages in different
> apps would continuously swap (trash?) on an otherwise
> unused (100k+ pages) system?
>
>> Thus 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.
>
> is that really what we want?

A kind of ;)

> I think we can do _better_ than a standalone machine
> and in many cases we really should ...

That's it! You are right - this is our ultimate goal. And we
plan to get there step by step. And we will appreciate your
patches fixing BUGS, improving the performance, extending the
functionality, etc.

> best,
> Herbert

Thanks for your attention,
Pavel

>> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
>>
>> The testing scenario may look like this:
>>
>> 1. Prepare the containers
>> # mkdir -p /containers/rss
>> # mount -t container none /containers/rss -o rss
>>
>> 2. Make the new group and move bash into it
>> # mkdir /containers/rss/0
>> # echo $$ > /containers/rss/0/tasks
>>
>> Since now we're in the 0 container.
>> We can alter the RSS limit
>> # echo -n 6000 > /containers/rss/0/rss_limit
>>
>> We can check the usage
>> # cat /containers/rss/0/rss_usage
>> 25
>>
>> And do other stuff. To check the reclamation to work we need a
>> simple program that touches many pages of memory, like this:
>>
>> #include <stdio.h>
>> #include <unistd.h>
>> #include <sys/mman.h>
>> #include <fcntl.h>
>>
>> #ifndef PGSIZE
>> #define PGSIZE 4096
>> #endif
>>
>> int main(int argc, char **argv)
>> {
>>   unsigned long pages;
>>   int i;

```
>>      char *mem;
>>
>>      if (argc < 2) {
>>              printf("Usage: %s <number_of_pages>\n", argv[0]);
>>              return 1;
>>      }
>>
>>      pages = strtol(argv[1], &mem, 10);
>>      if (*mem != '\0') {
>>              printf("Bad number %s\n", argv[1]);
>>              return 1;
>>      }
>>
>>      mem = mmap(NULL, pages * PGSIZE, PROT_READ | PROT_WRITE,
>>                      MAP_PRIVATE | MAP_ANON, 0, 0);
>>      if (mem == MAP_FAILED) {
>>              perror("map");
>>              return 2;
>>      }
>>
>>      for (i = 0; i < pages; i++)
>>              mem[i * PGSIZE] = 0;
>>
>>      printf("OK\n");
>>      return 0;
>> }
>> _____________________________________________________
>> Containers mailing list
>> Containers@lists.linux-foundation.org
>> https://lists.linux-foundation.org/mailman/listinfo/containers
>
```

Subject: Re: [PATCH 0/8] RSS controller based on process containers (v3.1)
Posted by Herbert Poetzl on Fri, 08 Jun 2007 15:37:00 GMT
View Forum Message <> Reply to Message

On Fri, Jun 08, 2007 at 04:39:28PM +0400, Pavel Emelianov wrote:
> Herbert Poetzl wrote:
> > On Mon, Jun 04, 2007 at 05:25:25PM +0400, Pavel Emelianov wrote:
> >> Adds RSS accounting and control within a container.
> >>
> >> Changes from v3

> >> - comments across the code
> >> - git-bisect safe split
> >> - lost places to move the page between active/inactive lists
> >>
> >> Ported above Paul's containers V10 with fixes from Balbir.
> >>
> >> RSS container includes the per-container RSS accounting
> >> and reclamation, and out-of-memory killer.
> >>
> >>
> >> Each mapped page has an owning container and is linked into its
> >> LRU lists just like in the global LRU ones. The owner of the page
> >> is the container that touched the page first.
> >
> >> As long as the page stays mapped it holds the container, is accounted
> >> into its usage and lives in its LRU list. When page is unmapped for
> >> the last time it releases the container.
> >
> >> The RSS usage is exactly the number of pages in its booth LRU lists,
> >> i.e. the nu,ber of pages used by this container.
> >
> > so there could be two guests, unified (i.e. sharing
> > most of the files as hardlinks), where the first one
> > holds 80% of the resulting pages, and the second one
> > 20%, and thus shows much lower 'RSS' usage as the
> > other one, although it is running the very same
> > processes and providing identical services?
>
> Herbert!!! Where have you been so long?

I was on vacation in april, and it took almost the
entire may to process the backlog ...

> You must have missed that we've decided not to account pages
> sharing right now, but start with that model. Later we'll make
> sharing accountable.

well, there are two ways not to account sharing:

 1) account it to the first user
 2) account it to every user

while the first one typically causes quite an
imbalance when applied to shared resources (as
Linux-VServer uses them), the latter one creates
a new, but fair, metric for the usage

to make that clear: we definitely prefer the

latter one over the former, because we do not
want to bring that imbalance to our shared guests
(besides that, the second one doesn't add any
overhead compared to the first one, more than that
it probably simplifies the entire design)

> (There's something bad with my memory. I have a feeling that I
> have already told that... many times...)

> >> When this usage exceeds the limit set some pages are reclaimed
> >> from the owning container. In case no reclamation possible the OOM
> >> killer starts thinning out the container.
> >
> > so the system (physical machine) starts reclaiming
> > and probably swapping even when there is no need
> > to do so?
>
> Good catch! The system will start reclaiming right when the
> container hits the limit to expend its IO bandwidth. Not some
> other's one that hit the global limit due to some bad container
> was allowed to go above it.

well, from the system PoV, a constantly swapping
guest (on an otherwise unused host) is definitely
something you do not really want, not to talk
about a tightly packed host system, where guests
start hogging the I/O with _unnecessary_ swapping

> > e.g. a system with a single guest, limited to 10k
> > pages, with a working set of 15k pages in different
> > apps would continuously swap (trash?) on an otherwise
> > unused (100k+ pages) system?
> >
> >> Thus 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.
> >
> > is that really what we want?
>
> A kind of ;)

okay, to clarify, we (Linux-VServer) do not want
that behavior ...

> > I think we can do _better_ than a standalone machine
> > and in many cases we really should ...

>
> That's it! You are right - this is our ultimate goal. And we
> plan to get there step by step. And we will appreciate your
> patches fixing BUGS, improving the performance, extending the
> functionality, etc.

well, I would rip out the entire accounting and
add a summation accounting as we do it right now,
no problem with keeping the reclaim mechanisms
though ... but I doubt that this is what you have
in mind?

> > best,
> > Herbert
>
> Thanks for your attention,
> Pavel

just to make it clear, I don't want any limits
in mainline which penalize the first started
guest in a shared guest scenario .. or to rephrase,
such a limit would be useless for our purpose

best,
Herbert

> >> Signed-off-by: Pavel Emelianov <xemul@openvz.org>
> >>
> >> The testing scenario may look like this:
> >>
> >> 1. Prepare the containers
> >> # mkdir -p /containers/rss
> >> # mount -t container none /containers/rss -o rss
> >>
> >> 2. Make the new group and move bash into it
> >> # mkdir /containers/rss/0
> >> # echo $$ > /containers/rss/0/tasks
> >>
> >> Since now we're in the 0 container.
> >> We can alter the RSS limit
> >> # echo -n 6000 > /containers/rss/0/rss_limit
> >>
> >> We can check the usage
> >> # cat /containers/rss/0/rss_usage
> >> 25
> >>
> >> And do other stuff. To check the reclamation to work we need a
> >> simple program that touches many pages of memory, like this:

```
> >>
> >> #include <stdio.h>
> >> #include <unistd.h>
> >> #include <sys/mman.h>
> >> #include <fcntl.h>
> >>
> >> #ifndef PGSIZE
> >> #define PGSIZE  4096
> >> #endif
> >>
> >> int main(int argc, char **argv)
> >> {
> >>         unsigned long pages;
> >>         int i;
> >>         char *mem;
> >>
> >>         if (argc < 2) {
> >>                 printf("Usage: %s <number_of_pages>\n", argv[0]);
> >>                 return 1;
> >>         }
> >>
> >>         pages = strtol(argv[1], &mem, 10);
> >>         if (*mem != '\0') {
> >>                 printf("Bad number %s\n", argv[1]);
> >>                 return 1;
> >>         }
> >>
> >>         mem = mmap(NULL, pages * PGSIZE, PROT_READ | PROT_WRITE,
> >>                         MAP_PRIVATE | MAP_ANON, 0, 0);
> >>         if (mem == MAP_FAILED) {
> >>                 perror("map");
> >>                 return 2;
> >>         }
> >>
> >>         for (i = 0; i < pages; i++)
> >>                 mem[i * PGSIZE] = 0;
> >>
> >>         printf("OK\n");
> >>         return 0;
> >> }
> >> _____
> >> Containers mailing list
> >> Containers@lists.linux-foundation.org
> >> https://lists.linux-foundation.org/mailman/listinfo/containers
> >
```

_____

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

Subject: Re: [PATCH 0/8] RSS controller based on process containers (v3.1)
Posted by Balbir Singh on Fri, 08 Jun 2007 17:07:56 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Fri, Jun 08, 2007 at 04:39:28PM +0400, Pavel Emelianov wrote:
>> Herbert Poetzl wrote:
>>> On Mon, Jun 04, 2007 at 05:25:25PM +0400, Pavel Emelianov wrote:
>>>> Adds RSS accounting and control within a container.
>>>>
>>>> Changes from v3
>>>> - comments across the code
>>>> - git-bisect safe split
>>>> - lost places to move the page between active/inactive lists
>>>>
>>>> Ported above Paul's containers V10 with fixes from Balbir.
>>>>
>>>> RSS container includes the per-container RSS accounting
>>>> and reclamation, and out-of-memory killer.
>>>>
>>>>
>>>> Each mapped page has an owning container and is linked into its
>>>> LRU lists just like in the global LRU ones. The owner of the page
>>>> is the container that touched the page first.
>>>> As long as the page stays mapped it holds the container, is accounted
>>>> into its usage and lives in its LRU list. When page is unmapped for
>>>> the last time it releases the container.
>>>> The RSS usage is exactly the number of pages in its booth LRU lists,
>>>> i.e. the nu,ber of pages used by this container.
>>> so there could be two guests, unified (i.e. sharing
>>> most of the files as hardlinks), where the first one
>>> holds 80% of the resulting pages, and the second one
>>> 20%, and thus shows much lower 'RSS' usage as the
>>> other one, although it is running the very same
>>> processes and providing identical services?

Hi, Herbert,

For page reclaim one page can belong to only container LRU and only
each physical page is accounted for. Consider what happens when we
equally charge each container

1. Lets say you have containers A and B, both showing that they
   are charged 45% of memory usage
2. The sum of these charges is equal to 90%, but the real memory

used is just 70%, since 20% of the charges are shared.

The system administrator will find this confusing while assigning resources

>> Herbert!!! Where have you been so long?
>
> I was on vacation in april, and it took almost the
> entire may to process the backlog ...
>
>> You must have missed that we've decided not to account pages
>> sharing right now, but start with that model. Later we'll make
>> sharing accountable.
>
> well, there are two ways not to account sharing:
>
> 1) account it to the first user
> 2) account it to every user
>
> while the first one typically causes quite an
> imbalance when applied to shared resources (as
> Linux-VServer uses them), the latter one creates
> a new, but fair, metric for the usage
>
> to make that clear: we definitely prefer the
> latter one over the former, because we do not
> want to bring that imbalance to our shared guests
> (besides that, the second one doesn't add any
> overhead compared to the first one, more than that
> it probably simplifies the entire design)
>

Please see comments by Nick at

http://lkml.org/lkml/2007/3/14/40

The imbalance is only temporary in the long run, the container
that uses a page more aggressively will get charged for it.
If the container that first brought the pages in is actively
using it, then it's only fair to charge it.

>> (There's something bad with my memory. I have a feeling that I
>>   have already told that... many times...)
>

>
>>>> When this usage exceeds the limit set some pages are reclaimed
>>>> from the owning container. In case no reclamation possible the OOM

>>>> killer starts thinning out the container.
>>> so the system (physical machine) starts reclaiming
>>> and probably swapping even when there is no need
>>> to do so?
>> Good catch! The system will start reclaiming right when the
>> container hits the limit to expend its IO bandwidth. Not some
>> other's one that hit the global limit due to some bad container
>> was allowed to go above it.
>
> well, from the system PoV, a constantly swapping
> guest (on an otherwise unused host) is definitely
> something you do not really want, not to talk
> about a tightly packed host system, where guests
> start hogging the I/O with _unnecessary_ swapping
>

Why do you call the swapping unnecessary? We defined a limit for
the containers, so that other containers are not impacted by
the memory usage of our container. We go overboard, it's either

1. We get penalized
2. We are not well configured, reconfigure

>>> e.g. a system with a single guest, limited to 10k
>>> pages, with a working set of 15k pages in different
>>> apps would continuously swap (trash?) on an otherwise
>>> unused (100k+ pages) system?
>>>

What to do with the free memory is an open question, we've
even discussed implementing soft limits (may be if required
sometime in the future).

>>>> Thus 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.
>>> is that really what we want?
>> A kind of ;)
>
> okay, to clarify, we (Linux-VServer) do not want
> that behavior ...
>

Hmmm... could you define the behaviour you expect?

>>> I think we can do _better_ than a standalone machine
>>> and in many cases we really should ...

>> That's it! You are right - this is our ultimate goal. And we
>> plan to get there step by step. And we will appreciate your
>> patches fixing BUGS, improving the performance, extending the
>> functionality, etc.
>
> well, I would rip out the entire accounting and
> add a summation accounting as we do it right now,
> no problem with keeping the reclaim mechanisms
> though ... but I doubt that this is what you have
> in mind?
>

Patches are always welcome!

>>> best,
>>> Herbert
>> Thanks for your attention,
>> Pavel
>
> just to make it clear, I don't want any limits
> in mainline which penalize the first started
> guest in a shared guest scenario .. or to rephrase,
> such a limit would be useless for our purpose
>

In summary, I would like to emphasize on the following points

1. Let's not build an ideal system, lets build something use-able, review-able and understand-able
2. Let's add more features when we hit a bottleneck/problem, code development process is always iterative
3. Please try our patches, test them, provide numbers, feedback, new patches to improve upon the existing code (in a reasonable amount of time, to keep the development on track), you never know you might end up liking what you use :-)

> best,
> Herbert
>

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

Subject: Re: [PATCH 0/8] RSS controller based on process containers (v3.1)
Posted by Vaidyanathan Srinivas on Fri, 08 Jun 2007 17:44:36 GMT
View Forum Message <> Reply to Message

Herbert Poetzl wrote:
> On Fri, Jun 08, 2007 at 04:39:28PM +0400, Pavel Emelianov wrote:
>> Herbert Poetzl wrote:
>>> On Mon, Jun 04, 2007 at 05:25:25PM +0400, Pavel Emelianov wrote:
[snip]
>>>> When this usage exceeds the limit set some pages are reclaimed
>>>> from the owning container. In case no reclamation possible the OOM
>>>> killer starts thinning out the container.
>>> so the system (physical machine) starts reclaiming
>>> and probably swapping even when there is no need
>>> to do so?
>> Good catch! The system will start reclaiming right when the
>> container hits the limit to expend its IO bandwidth. Not some
>> other's one that hit the global limit due to some bad container
>> was allowed to go above it.
>
> well, from the system PoV, a constantly swapping
> guest (on an otherwise unused host) is definitely
> something you do not really want, not to talk
> about a tightly packed host system, where guests
> start hogging the I/O with _unnecessary_ swapping
>
>>> e.g. a system with a single guest, limited to 10k
>>> pages, with a working set of 15k pages in different
>>> apps would continuously swap (trash?) on an otherwise
>>> unused (100k+ pages) system?
>>>

Hi Herbert,

When the reclaim process started, swappable pages are unmapped and
moved to swapcache. The RSS accounting treats the page as dropped,
but however the page will remain in memory until there is enough
global pressure to push it out to disk. When a page is faulted-in,
the page is just remapped from the swapcache.

In effect container 'trashing' is not as bad in an otherwise unused
system. Well, certainly we pay a penalty to move around the pages
instead of keeping them mapped in RSS as long as free memory was

available.

The pagecache controller is suppose to track the pagecache and swapcache pages and push out pages to disk. As Balbir and Pavel have mentioned, we will be building the features in stages.

--Vaidy

[snip]

_____

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