Subject: [RFC][PATCH 0/3] Containers: Pagecache accounting and control subsystem (v3)
Posted by Vaidyanathan Srinivas on Wed, 23 May 2007 14:48:57 GMT
View Forum Message <> Reply to Message

Containers: Pagecache accounting and control subsystem (v3)

This patch extends the RSS controller to account and reclaim pagecache
and swapcache pages. This is a prototype to demonstrate that the existing
container infrastructure is useful to build different VM controller.

This patch is based on RSS Controller V2 by Pavel and Balbir. This patch
depends on
1. Paul Menage's Containers (V8): Generic Process Containers
http://lkml.org/lkml/2007/4/6/297
2. Pavel Emelianov's RSS controller based on process containers (v2)
http://lkml.org/lkml/2007/4/9/78
3. Balbir's fixes for RSS controller as mentioned in
http://lkml.org/lkml/2007/5/17/232

This is very much work-in-progress, you can certainly expect hangs/crash if
both pagecache and rss limits are set and the container is stressed.

Comments, suggestions and criticisms are welcome.

Thanks,
Vaidy

Features:
---
* No new subsystem is added. The RSS controller subsystem is extended
  since most of the code can be shared between pagecache control and
  RSS control.
* The accounting number include pages in swap cache and filesystem
  buffer pages apart from pagecache, basically everything under
  NR_FILE_PAGES is counted as pagecache.
* Limits on pagecache can be set by echo -n 100000 > pagecache_limit on
  the /container file system. The unit is in pages or 4 kilobytes
* If the pagecache utilisation limit is exceeded, the container reclaim
  code is invoked to recover pages from the container.

Advantages:
---
* Minimal code changes to RSS controller to include pagecache pages

Limitations:
---

* All limitation of RSS controller v2 applies to this code as well
* Page reclaim needs to be reworked to select correct pages when the
  respective limits are exceeded
* Concurrent and recursive triggering of reclaimer code is a mess leading
  to deadlocks. Reclaimer needs to be serialised and reworked to
  do the right job and also improve performance

Usage:
------
* Add all dependent patches before including this patch
* No new config settings apart from enabling CONFIG_RSS_CONTAINER
* Boot new kernel
* Mount container filesystem
 mount -t container none /container
 cd /container
* Create new container
 mkdir mybox
 cd /container/mybox
* Add current shell to container
 echo $$ > tasks
* There are two files pagecache_usage and pagecache_limit
* In order to set limit, echo value in pages (4KB) to pagecache_limit
 echo -n 100000 > pagecache_limit
 #This would set 409MB limit on pagecache usage
* Trash the system from current shell using scp/cp/dd/tar etc
* Watch pagecache_usage and /proc/meminfo to verify behavior

Tests:
------
* Simple dd/cat/cp test on pagecache limit
* rss_limit was tested with simple test application that would malloc
  predefined size of memory and touch them to allocate pages.

ToDo:
----
* Optimise the reclaim. Currently isolate_container_pages does not distinguish
  between whether pagecache limit is hit or rss limit is hit
* Prevent concurrent reclaim and recursive reclaim when both limits are set.

Patch Series:
----
pagecache-controller-v3-setup.patch
pagecache-controller-v3-acct.patch
pagecache-controller-v3-acct-hooks.patch

Subject: [RFC][PATCH 1/3] Containers: Pagecache accounting and control

Pagecache controller setup

This patch basically adds user interface files in container fs
similar to the rss control files.

pagecache_usage, pagecache_limit and pagecache_failcnt are added
to each container. All units are 'pages' as in rss controller.

pagecache usage is all file backed pages used by the container
which includes swapcache as well.

Separate res_counter for pagecache has been added.

Signed-off-by: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
---
 mm/rss_container.c | 42 ++
 1 file changed, 42 insertions(+)

--- linux-2.6.20.orig/mm/rss_container.c
+++ linux-2.6.20/mm/rss_container.c
@@ -16,6 +16,7 @@

 struct rss_container {
  struct res_counter res;
+ struct res_counter pagecache_res;
  struct list_head inactive_list;
  struct list_head active_list;
  atomic_t rss_reclaimed;
@@ -266,6 +267,7 @@ static int rss_create(struct container_s
   return -ENOMEM;

  res_counter_init(&rss->res);
+ res_counter_init(&rss->pagecache_res);
  INIT_LIST_HEAD(&rss->inactive_list);
  INIT_LIST_HEAD(&rss->active_list);
  rss_container_attach(rss, cont);
@@ -308,6 +310,21 @@ static ssize_t rss_read_reclaimed(struct
    ppos, buf, s - buf);
 }

+static ssize_t pagecache_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)->pagecache_res,
+   cft->private, userbuf, nbytes, ppos);
+}
+
+static ssize_t pagecache_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)->pagecache_res,
+   cft->private, userbuf, nbytes, ppos);
+}

 static struct cftype rss_usage = {
  .name = "rss_usage",
@@ -333,6 +350,25 @@ static struct cftype rss_reclaimed = {
  .read = rss_read_reclaimed,
 };

+static struct cftype pagecache_usage = {
+ .name = "pagecache_usage",
+ .private = RES_USAGE,
+ .read = pagecache_read,
+};
+
+static struct cftype pagecache_limit = {
+ .name = "pagecache_limit",
+ .private = RES_LIMIT,
+ .read = pagecache_read,
+ .write = pagecache_write,
+};
+
+static struct cftype pagecache_failcnt = {
+ .name = "pagecache_failcnt",
+ .private = RES_FAILCNT,
+ .read = pagecache_read,
+};
+
 static int rss_populate(struct container_subsys *ss,
   struct container *cont)
 {
@@ -346,6 +382,12 @@ static int rss_populate(struct container
   return rc;
  if ((rc = container_add_file(cont, &rss_reclaimed)) < 0)
   return rc;
+ if ((rc = container_add_file(cont, &pagecache_usage)) < 0)
+  return rc;
+ if ((rc = container_add_file(cont, &pagecache_failcnt)) < 0)
+  return rc;
```

```
+ if ((rc = container_add_file(cont, &pagecache_limit)) < 0)
+  return rc;

  return 0;
 }
```

Subject: [RFC][PATCH 2/3] Containers: Pagecache accounting and control subsystem (v3)
Posted by Vaidyanathan Srinivas on Wed, 23 May 2007 14:52:17 GMT
View Forum Message <> Reply to Message

Pagecache Accounting

The rss accounting hooks have been generalised to handle both anon pages
and file backed pages and charge the respective resource counters.

New flags and ref count has been added to page_container structure.
The ref count is used to ensure a page is added or removed from page_container
list only once independent of repeated calls from pagecache, swapcache and mmap
to RSS.

Flags in page_container is used to keep the accounting correct between RSS and
unmapped pagecache (includes swapcache) pages.

Signed-off-by: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
---
 include/linux/rss_container.h | 19 +++-
 mm/rss_container.c | 161 ++++++++++++++++++++++++++++++++-----------
 2 files changed, 127 insertions(+), 53 deletions(-)

```
--- linux-2.6.20.orig/include/linux/rss_container.h
+++ linux-2.6.20/include/linux/rss_container.h
@@ -12,13 +12,22 @@
 struct page_container;
 struct rss_container;

+/* Flags for container_page_xxx calls */
+#define PAGE_TYPE_RSS  0x00000001
+#define PAGE_TYPE_PAGECACHE 0x00000002
+#define PAGE_TYPE_SWAPCACHE 0x00000004
+
+#define PAGE_SUBTYPE_ANON 0x00010000
+#define PAGE_SUBTYPE_FILE 0x00020000
+
 #ifdef CONFIG_RSS_CONTAINER
-int container_rss_prepare(struct page *, struct vm_area_struct *vma,
```

```
-  struct page_container **);
+int container_page_prepare(struct page *page, struct mm_struct *mm,
+  struct page_container **ppc, unsigned int flags);
+
+void container_page_add(struct page_container *, unsigned int flags);
+void container_page_del(struct page_container *, unsigned int flags);
+void container_page_release(struct page_container *, unsigned int flags);

-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);
--- linux-2.6.20.orig/mm/rss_container.c
+++ linux-2.6.20/mm/rss_container.c
@@ -24,11 +24,18 @@ struct rss_container {
 };

 struct page_container {
+ unsigned long flags;
+ unsigned long ref_cnt;
  struct page *page;
  struct rss_container *cnt;
  struct list_head list;
 };

+/* Flags for Page Container */
+#define PAGE_IN_PAGECACHE  0x0001
+#define PAGE_IN_RSS   0x0002
+#define PAGE_IN_SWAPCACHE  0x0004
+
 static inline struct rss_container *rss_from_cont(struct container *cnt)
 {
  return container_of(container_subsys_state(cnt, rss_subsys_id),
@@ -49,52 +56,143 @@ void mm_free_container(struct mm_struct
  css_put(&mm->rss_container->css);
 }

-int container_rss_prepare(struct page *page, struct vm_area_struct *vma,
-  struct page_container **ppc)
+int container_page_prepare(struct page *page, struct mm_struct *mm,
+  struct page_container **ppc, unsigned int flags)
 {
  struct rss_container *rss;
  struct page_container *pc;
+ int rc = 1;
```

```
  rcu_read_lock();
- rss = rcu_dereference(vma->vm_mm->rss_container);
+ rss = rcu_dereference(mm->rss_container);
  css_get(&rss->css);
  rcu_read_unlock();

- pc = kmalloc(sizeof(struct page_container), GFP_KERNEL);
- if (pc == NULL)
-  goto out_nomem;
+ /* Charge the respective resource count */
+ while (rc) {
+  if (flags & PAGE_TYPE_RSS)
+   rc = res_counter_charge(&rss->res, 1);
+  else
+   rc = res_counter_charge(&rss->pagecache_res, 1);
+
+  if (!rc)
+   break; /* All well */

- while (res_counter_charge(&rss->res, 1)) {
   if (try_to_free_pages_in_container(rss)) {
    atomic_inc(&rss->rss_reclaimed);
-   continue;
+   continue; /* Try agin to charge container */
   }
-
+  /* Unable to free memory?? */
   container_out_of_memory(rss);
   if (test_thread_flag(TIF_MEMDIE))
    goto out_charge;
  }

- pc->page = page;
- pc->cnt = rss;
+  /* Page may have been added to container earlier */
+  pc = page_container(page);
+  if (!pc) {
+   pc = kzalloc(sizeof(struct page_container), GFP_KERNEL);
+   if (pc == NULL)
+    goto out_nomem;
+
+  pc->page = page;
+  pc->cnt = rss;
+  }
+
  *ppc = pc;
  return 0;
```

```
 out_charge:
- kfree(pc);
+ /* Need to zero page_container?? */
 out_nomem:
  css_put(&rss->css);
  return -ENOMEM;
 }

-void container_rss_release(struct page_container *pc)
+void container_page_release(struct page_container *pc, unsigned int flags)
+{
+ struct rss_container *rss;
+
+ rss = pc->cnt;
+ /* Setting the accounts right */
+ if (flags & PAGE_TYPE_RSS) {
+  res_counter_uncharge(&rss->res, 1);
+  if (pc->flags & PAGE_IN_PAGECACHE)
+   res_counter_charge(&rss->pagecache_res, 1);
+ } else {
+  res_counter_uncharge(&rss->pagecache_res, 1);
+ }
+
+ if (!(pc->flags)) {
+  page_container(pc->page) = 0;
+  kfree(pc);
+ }
+ css_put(&rss->css);
+}
+
+void container_page_add(struct page_container *pc, unsigned int flags)
+{
+ struct page *pg;
+ struct rss_container *rss;
+
+ pg = pc->page;
+
+ if (pg == ZERO_PAGE(0))
+  return;
+
+ /* Update flage in page container */
+ if (flags & PAGE_TYPE_RSS) {
+  pc->flags |= PAGE_IN_RSS;
+ } else {
+  pc->flags |= PAGE_IN_PAGECACHE;
+ }
+
+ rss = pc->cnt;
```

```
+ spin_lock_irq(&rss->res.lock);
+ if (!pc->ref_cnt)
+  list_add(&pc->list, &rss->inactive_list);
+ pc->ref_cnt++;
+ spin_unlock_irq(&rss->res.lock);
+
+ page_container(pg) = pc;
+}
+
+void container_page_del(struct page_container *pc, unsigned int flags)
 {
+ struct page *page;
  struct rss_container *rss;

+ page = pc->page;
  rss = pc->cnt;
- res_counter_uncharge(&rss->res, 1);
+
+ if (page == ZERO_PAGE(0))
+  return;
+ BUG_ON(pc->flags & ~3);
+ /* Setting the accounts right */
+ if (flags & PAGE_TYPE_RSS) {
+  res_counter_uncharge(&rss->res, 1);
+  pc->flags &= ~PAGE_IN_RSS;
+  /* If it is a pagecache page the move acct to pagecache */
+  if (pc->flags & PAGE_IN_PAGECACHE)
+   res_counter_charge(&rss->pagecache_res, 1);
+ } else {
+  res_counter_uncharge(&rss->pagecache_res, 1);
+  pc->flags &= ~PAGE_IN_PAGECACHE;
+  BUG_ON(pc->flags);
+ }
+ if(!(pc->flags)) {
+  spin_lock_irq(&rss->res.lock);
+  pc->ref_cnt--;
+  if (!pc->ref_cnt)
+   list_del(&pc->list);
+  spin_unlock_irq(&rss->res.lock);
+
+  if (!pc->ref_cnt) {
+   kfree(pc);
+    page_container(page) = 0;
+  }
+ }
  css_put(&rss->css);
- kfree(pc);
 }
```

```
 void container_rss_move_lists(struct page *pg, bool active)
@@ -183,39 +281,6 @@ unsigned long isolate_pages_in_container
  return ret;
 }

-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);
- init_page_container(page);
-}
-
 static void rss_move_task(struct container_subsys *ss,
   struct container *cont,
   struct container *old_cont,
```

Subject: [RFC][PATCH 3/3] Containers: Pagecache accounting and control subsystem (v3)
Posted by Vaidyanathan Srinivas on Wed, 23 May 2007 14:53:47 GMT

Pagecache and RSS accounting Hooks

New calls have been added from swap_state.c and filemap.c to track pagecache
and swapcache pages.

All existing RSS hooks have been generalised for pagecache accounting as well.

Most of these are function prototype changes.

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

```
 fs/exec.c       |    4 ++--
 mm/filemap.c    |   18 ++++++++++++++++++
 mm/fremap.c     |    4 ++--
 mm/memory.c     |   20 ++++++++++----------
 mm/migrate.c    |    4 ++--
 mm/rmap.c       |   12 ++++++------
 mm/swap_state.c |   16 ++++++++++++++++
 mm/swapfile.c   |    4 ++--
 8 files changed, 58 insertions(+), 24 deletions(-)

--- linux-2.6.20.orig/fs/exec.c
+++ linux-2.6.20/fs/exec.c
@@ -316,7 +316,7 @@ void install_arg_page(struct vm_area_str
  if (unlikely(anon_vma_prepare(vma)))
   goto out;

- if (container_rss_prepare(page, vma, &pcont))
+ if (container_page_prepare(page, vma->vm_mm, &pcont, PAGE_TYPE_RSS))
   goto out;

  flush_dcache_page(page);
@@ -338,7 +338,7 @@ void install_arg_page(struct vm_area_str
  return;

 out_release:
- container_rss_release(pcont);
+ container_page_release(pcont, PAGE_TYPE_RSS);
 out:
  __free_page(page);
  force_sig(SIGKILL, current);
--- linux-2.6.20.orig/mm/filemap.c
+++ linux-2.6.20/mm/filemap.c
@@ -30,6 +30,7 @@
 #include <linux/security.h>
 #include <linux/syscalls.h>
```

```
 #include <linux/cpuset.h>
+#include <linux/rss_container.h>
 #include "filemap.h"
 #include "internal.h"

@@ -117,6 +118,9 @@ void __remove_from_page_cache(struct pag
  struct address_space *mapping = page->mapping;

  radix_tree_delete(&mapping->page_tree, page->index);
+ /* Uncharge before the mapping is gone */
+ if (page_container(page))
+  container_page_del(page_container(page), PAGE_TYPE_PAGECACHE);
  page->mapping = NULL;
  mapping->nrpages--;
  __dec_zone_page_state(page, NR_FILE_PAGES);
@@ -438,6 +442,8 @@ int add_to_page_cache(struct page *page,
   pgoff_t offset, gfp_t gfp_mask)
 {
  int error = radix_tree_preload(gfp_mask & ~__GFP_HIGHMEM);
+ struct page_container *pc;
+ struct mm_struct *mm;

  if (error == 0) {
   write_lock_irq(&mapping->tree_lock);
@@ -451,6 +457,18 @@ int add_to_page_cache(struct page *page,
    __inc_zone_page_state(page, NR_FILE_PAGES);
   }
   write_unlock_irq(&mapping->tree_lock);
+  /* Unlock before charge, because we may reclaim this inline */
+  if(!error) {
+   if (current->mm)
+    mm = current->mm;
+   else
+    mm = &init_mm;
+   if (!container_page_prepare(page, mm, &pc, PAGE_TYPE_PAGECACHE))
+    container_page_add(pc, PAGE_TYPE_PAGECACHE);
+   else
+    BUG();
+  }
+
   radix_tree_preload_end();
  }
  return error;
--- linux-2.6.20.orig/mm/fremap.c
+++ linux-2.6.20/mm/fremap.c
@@ -61,7 +61,7 @@ int install_page(struct mm_struct *mm, s
  spinlock_t *ptl;
  struct page_container *pcont;
```

```
- if (container_rss_prepare(page, vma, &pcont))
+ if (container_page_prepare(page, vma->vm_mm, &pcont, PAGE_TYPE_RSS))
   goto out_release;

  pte = get_locked_pte(mm, addr, &ptl);
@@ -95,7 +95,7 @@ unlock:
  pte_unmap_unlock(pte, ptl);
 out:
  if (err != 0)
-  container_rss_release(pcont);
+  container_page_release(pcont, PAGE_TYPE_RSS);
 out_release:
  return err;
 }
--- linux-2.6.20.orig/mm/memory.c
+++ linux-2.6.20/mm/memory.c
@@ -1583,7 +1583,7 @@ gotten:
   cow_user_page(new_page, old_page, address, vma);
  }

- if (container_rss_prepare(new_page, vma, &pcont))
+ if (container_page_prepare(new_page, vma->vm_mm, &pcont, PAGE_TYPE_RSS))
   goto oom;

  /*
@@ -1619,7 +1619,7 @@ gotten:
   new_page = old_page;
   ret |= VM_FAULT_WRITE;
  } else
-  container_rss_release(pcont);
+  container_page_release(pcont, PAGE_TYPE_RSS);

  if (new_page)
   page_cache_release(new_page);
@@ -2029,7 +2029,7 @@ static int do_swap_page(struct mm_struct
   count_vm_event(PGMAJFAULT);
  }

- if (container_rss_prepare(page, vma, &pcont)) {
+ if (container_page_prepare(page, vma->vm_mm, &pcont, PAGE_TYPE_RSS)) {
   ret = VM_FAULT_OOM;
   goto out;
  }
@@ -2047,7 +2047,7 @@ static int do_swap_page(struct mm_struct

  if (unlikely(!PageUptodate(page))) {
   ret = VM_FAULT_SIGBUS;
```

```
-  container_rss_release(pcont);
+  container_page_release(pcont, PAGE_TYPE_RSS);
   goto out_nomap;
  }

@@ -2084,7 +2084,7 @@ unlock:
 out:
  return ret;
 out_nomap:
- container_rss_release(pcont);
+ container_page_release(pcont, PAGE_TYPE_RSS);
  pte_unmap_unlock(page_table, ptl);
  unlock_page(page);
  page_cache_release(page);
@@ -2115,7 +2115,7 @@ static int do_anonymous_page(struct mm_s
   if (!page)
    goto oom;

-  if (container_rss_prepare(page, vma, &pcont))
+  if (container_page_prepare(page, vma->vm_mm, &pcont, PAGE_TYPE_RSS))
    goto oom_release;

   entry = mk_pte(page, vma->vm_page_prot);
@@ -2151,7 +2151,7 @@ unlock:
  pte_unmap_unlock(page_table, ptl);
  return VM_FAULT_MINOR;
 release_container:
- container_rss_release(pcont);
+ container_page_release(pcont, PAGE_TYPE_RSS);
 release:
  page_cache_release(page);
  goto unlock;
@@ -2245,7 +2245,7 @@ retry:
   }
  }

- if (container_rss_prepare(new_page, vma, &pcont))
+ if (container_page_prepare(new_page, vma->vm_mm, &pcont, PAGE_TYPE_RSS))
   goto oom;

  page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
@@ -2256,7 +2256,7 @@ retry:
   */
  if (mapping && unlikely(sequence != mapping->truncate_count)) {
   pte_unmap_unlock(page_table, ptl);
-  container_rss_release(pcont);
+  container_page_release(pcont, PAGE_TYPE_RSS);
   page_cache_release(new_page);
```

```
   cond_resched();
   sequence = mapping->truncate_count;
@@ -2295,7 +2295,7 @@ retry:
    }
   } else {
    /* One of our sibling threads was faster, back out. */
-  container_rss_release(pcont);
+  container_page_release(pcont, PAGE_TYPE_RSS);
    page_cache_release(new_page);
    goto unlock;
   }
--- linux-2.6.20.orig/mm/migrate.c
+++ linux-2.6.20/mm/migrate.c
@@ -159,7 +159,7 @@ static void remove_migration_pte(struct
     return;
    }

- if (container_rss_prepare(new, vma, &pcont)) {
+ if (container_page_prepare(new, vma->vm_mm, &pcont, PAGE_TYPE_RSS)) {
    pte_unmap(ptep);
    return;
   }
@@ -194,7 +194,7 @@ static void remove_migration_pte(struct

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

 /*
--- linux-2.6.20.orig/mm/rmap.c
+++ linux-2.6.20/mm/rmap.c
@@ -551,10 +551,10 @@ void page_add_anon_rmap(struct page *pag
  struct page_container *pcont)
 {
  if (atomic_inc_and_test(&page->_mapcount)) {
-  container_rss_add(pcont);
+  container_page_add(pcont, PAGE_TYPE_RSS|PAGE_SUBTYPE_ANON);
    __page_set_anon_rmap(page, vma, address);
  } else
-  container_rss_release(pcont);
+  container_page_release(pcont, PAGE_TYPE_RSS|PAGE_SUBTYPE_ANON);
   /* else checking page index and mapping is racy */
 }

@@ -573,7 +573,7 @@ void page_add_new_anon_rmap(struct page
  struct page_container *pcont)
```

```
 {
  atomic_set(&page->_mapcount, 0); /* elevate count by 1 (starts at -1) */
- container_rss_add(pcont);
+ container_page_add(pcont, PAGE_TYPE_RSS|PAGE_SUBTYPE_ANON);
  __page_set_anon_rmap(page, vma, address);
 }

@@ -588,10 +588,10 @@ void page_add_file_rmap(struct page *pag
 {
  if (atomic_inc_and_test(&page->_mapcount)) {
   if (pcont)
-   container_rss_add(pcont);
+   container_page_add(pcont, PAGE_TYPE_RSS|PAGE_SUBTYPE_FILE);
   __inc_zone_page_state(page, NR_FILE_MAPPED);
  } else if (pcont)
-  container_rss_release(pcont);
+  container_page_release(pcont, PAGE_TYPE_RSS|PAGE_SUBTYPE_FILE);
 }

 /**
@@ -621,7 +621,7 @@ void page_remove_rmap(struct page *page,
   }

   if (pcont)
-   container_rss_del(pcont);
+   container_page_del(pcont, PAGE_TYPE_RSS);
   /*
    * It would be tidy to reset the PageAnon mapping here,
    * but that might overwrite a racing page_add_anon_rmap
--- linux-2.6.20.orig/mm/swap_state.c
+++ linux-2.6.20/mm/swap_state.c
@@ -16,6 +16,7 @@
 #include <linux/backing-dev.h>
 #include <linux/pagevec.h>
 #include <linux/migrate.h>
+#include <linux/rss_container.h>

 #include <asm/pgtable.h>

@@ -73,6 +74,8 @@ static int __add_to_swap_cache(struct pa
         gfp_t gfp_mask)
 {
  int error;
+ struct page_container *pc;
+ struct mm_struct *mm;

  BUG_ON(PageSwapCache(page));
  BUG_ON(PagePrivate(page));
```

```
@@ -91,6 +94,17 @@ static int __add_to_swap_cache(struct pa
   }
   write_unlock_irq(&swapper_space.tree_lock);
   radix_tree_preload_end();
+  /* Unlock before charge, because we may reclaim this inline */
+  if(!error) {
+   if (current->mm)
+    mm = current->mm;
+   else
+    mm = &init_mm;
+   if (!container_page_prepare(page, mm, &pc, PAGE_TYPE_PAGECACHE))
+    container_page_add(pc, PAGE_TYPE_PAGECACHE);
+   else
+    BUG();
+  }
  }
  return error;
 }
@@ -129,6 +143,8 @@ void __delete_from_swap_cache(struct pag
  BUG_ON(PagePrivate(page));

  radix_tree_delete(&swapper_space.page_tree, page_private(page));
+ if (page_container(page))
+  container_page_del(page_container(page), PAGE_TYPE_PAGECACHE);
  set_page_private(page, 0);
  ClearPageSwapCache(page);
  total_swapcache_pages--;
--- linux-2.6.20.orig/mm/swapfile.c
+++ linux-2.6.20/mm/swapfile.c
@@ -535,7 +535,7 @@ static int unuse_pte_range(struct vm_are
  int found = 0;
  struct page_container *pcont;

- if (container_rss_prepare(page, vma, &pcont))
+ if (container_page_prepare(page, vma->vm_mm, &pcont, PAGE_TYPE_RSS))
   return 0;

  pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
@@ -552,7 +552,7 @@ static int unuse_pte_range(struct vm_are
  } while (pte++, addr += PAGE_SIZE, addr != end);
  pte_unmap_unlock(pte - 1, ptl);
  if (!found)
-  container_rss_release(pcont);
+  container_page_release(pcont, PAGE_TYPE_RSS);
  return found;
 }
```