Subject: [RFC][PATCH 0/3] Containers: Integrated RSS and pagecache control v5
Posted by Vaidyanathan Srinivas on Fri, 29 Jun 2007 06:17:23 GMT
View Forum Message <> Reply to Message

Containers: Integrated RSS and pagecache accounting and control v5
 --- ------

Based on the discussions at OLS yesterday, the consensus was to try an
integrated pagecache controller along with RSS controller under the
same usage limit.

This patch extends the RSS controller to account and reclaim pagecache
and swapcache pages. The same 'rss_limit' now applies to both RSS pages
and pagecache pages. When the limit is reached, both pagecache and RSS
pages are reclaimed in LRU order as per the normal system wide reclaim
policy.

This patch is based on RSS Controller V3.1 by Pavel and Balbir. This patch
depends on

1. Paul Menage's Containers(V10): Generic Process Containers
http://lwn.net/Articles/236032/
2. Pavel Emelianov's RSS controller based on process containers (v3.1)
http://lwn.net/Articles/236817/
3. Balbir's fixes for RSS controller as mentioned in
http://lkml.org/lkml/2007/6/04/185

This is very much work-in-progress and it have been posted for comments
after some basic testing with kernbench.

Comments, suggestions and criticisms are welcome.

--Vaidy

Features:
----
* Single limit for both RSS and pagecache/swapcache pages
* 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 under rss_usage.
* The usage limit set in rss_limit applies to sum of both RSS and
  pagecache pages
* Limits on pagecache can be set by echo -n 100000 > rss_limit on
  the /container file system. The unit is in pages or 4 kilobytes
* If the pagecache+RSS utilisation exceed the limit, 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 applies to this code as well
* Per-container reclaim knobs like dirty ratio, vm_swappiness may
  provide better control

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
```

* In order to set limit, echo value in pages (4KB) to rss_limit

```
 echo -n 100000 > rss_limit
 #This would set 409MB limit on pagecache+rss usage
```

* Trash the system from current shell using scp/cp/dd/tar etc
* Watch rss_usage and /proc/meminfo to verify behavior

Tests:
------
* Simple dd/cat/cp test on pagecache limit/reclaim
* rss_limit was tested with simple test application that would malloc
  predefined size of memory and touch them to allocate pages.
* kernbench was run under container with 400MB memory limit

ToDo:
----
* Optimise the reclaim.
* Per-container VM stats and knobs

Patch Series:
----
pagecache-controller-v5-acct.patch
pagecache-controller-v5-acct-hooks.patch
pagecache-controller-v5-reclaim.patch

ChangeLog:
----

v5: Integrated pagecache + rss controller

* No separate pagecache_limit
* pagecache and rss pages accounted in rss_usage and governed by rss_limit
* Each page counted only once in rss_usage. Mapped or unmapped
  pagecache pages are counted alike in rss_usage

v4:
* Patch remerged to Container v10 and RSS v3.1
* Bug fixes
* Tested with kernbench

v3:
* Patch merged with Containers v8 and RSS v2

Subject: [RFC][PATCH 1/3] Pagecache accounting
Posted by Vaidyanathan Srinivas on Fri, 29 Jun 2007 06:19:37 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 resource counter.

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.

No setup patch is required since rss_limit and rss_usage has been generalised
as the resource counter for pagecache as well.

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

```
---
 include/linux/rss_container.h |   18 ++---
 mm/rss_container.c            |  134 +++++++++++++++++++++++++++-------------
 2 files changed, 99 insertions(+), 53 deletions(-)

--- linux-2.6.22-rc2-mm1.orig/include/linux/rss_container.h
+++ linux-2.6.22-rc2-mm1/include/linux/rss_container.h
@@ -68,11 +68,11 @@ struct rss_container;
  *
  */
```

```
-int container_rss_prepare(struct page *, struct vm_area_struct *vma,
+int container_page_prepare(struct page *, struct mm_struct *mm,
   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 container_page_add(struct page_container *);
+void container_page_del(struct page_container *);
+void container_page_release(struct page_container *);

 void container_out_of_memory(struct rss_container *);

@@ -85,22 +85,22 @@ unsigned long isolate_pages_in_container
   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)
+static inline int container_page_prepare(struct page *pg,
+  struct mm_struct *mm, 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_page_add(struct page_container *pc)
 {
 }

-static inline void container_rss_del(struct page_container *pc)
+static inline void container_page_del(struct page_container *pc)
 {
 }

-static inline void container_rss_release(struct page_container *pc)
+static inline void container_page_release(struct page_container *pc)
 {
 }

--- linux-2.6.22-rc2-mm1.orig/mm/rss_container.c
+++ linux-2.6.22-rc2-mm1/mm/rss_container.c
@@ -56,6 +56,9 @@ struct rss_container {
  */

 struct page_container {
+ unsigned long ref_cnt;  /* Ref cnt to keep track of
+    * multiple additions of same page
+    */
  struct page *page;
  struct rss_container *cnt;
  struct list_head list; /* this is the element of (int)active_list of
@@ -93,26 +96,36 @@ void mm_free_container(struct mm_struct
  * 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,
+int container_page_prepare(struct page *page, struct mm_struct *mm,
   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;
+   struct rss_container *rss;
+   struct page_container *pc;
+  int rc = 1;
+
+   /* Page may have been added to container earlier */
+   pc = page_container(page);
+ /* Check if this is fist time addition or not */
+ if (!pc) {
+  rcu_read_lock();
+  rss = rcu_dereference(mm->rss_container);
+  css_get(&rss->css);
+  rcu_read_unlock();
+ } else {
+  rss = pc->cnt;
+ }

- while (res_counter_charge(&rss->res, 1)) {
-  if (try_to_free_pages_in_container(rss)) {
-   atomic_inc(&rss->rss_reclaimed);
-   continue;
-  }
+  /* Charge the respective resource count first time only */
+  while (rc && !pc) {
+  rc = res_counter_charge(&rss->res, 1);
+
+   if (!rc)
+    break; /* All well */
+
+    if (try_to_free_pages_in_container(rss)) {
+     atomic_inc(&rss->rss_reclaimed);
+    continue; /* Try agin to charge container */
+    }

   /*
     * try_to_free_pages() might not give us a full picture
@@ -125,60 +138,93 @@ int container_rss_prepare(struct page *p
   if (res_counter_check_under_limit(&rss->res))
    continue;

-  container_out_of_memory(rss);
-  if (test_thread_flag(TIF_MEMDIE))
-   goto out_charge;
- }
+   /* Unable to free memory?? */
+    container_out_of_memory(rss);
+    if (test_thread_flag(TIF_MEMDIE))
+     goto out_charge;
+   }
+
+   /* First time addition to container: Create new page_container */
+   if (!pc) {
+    pc = kzalloc(sizeof(struct page_container), GFP_KERNEL);
+    if (pc == NULL)
+     goto out_nomem;
+
+   pc->page = page;
+   pc->cnt = rss;
+   }

- pc->page = page;
- pc->cnt = rss;
- *ppc = pc;
- return 0;
+   *ppc = pc;
+   return 0;

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

```
+   css_put(&rss->css);
+   return -ENOMEM;
+
 }

-void container_rss_release(struct page_container *pc)
+void container_page_release(struct page_container *pc)
 {
  struct rss_container *rss;

  rss = pc->cnt;
- res_counter_uncharge(&rss->res, 1);
- css_put(&rss->css);
- kfree(pc);
+ /* Setting the accounts right */
+ if (!pc->ref_cnt) {
+  res_counter_uncharge(&rss->res, 1);
+  set_page_container(pc->page, NULL);
+  kfree(pc);
+  css_put(&rss->css);
+ }
 }

-void container_rss_add(struct page_container *pc)
+void container_page_add(struct page_container *pc)
 {
  struct page *pg;
  struct rss_container *rss;
+ unsigned long irqflags;

  pg = pc->page;
- rss = pc->cnt;
+ if (pg == ZERO_PAGE(0))
+  return;

- spin_lock_irq(&rss->res.lock);
- list_add(&pc->list, &rss->active_list);
- spin_unlock_irq(&rss->res.lock);
+ rss = pc->cnt;
+ spin_lock_irqsave(&rss->res.lock, irqflags);
+ if (!pc->ref_cnt)
+  list_add(&pc->list, &rss->inactive_list);
+ pc->ref_cnt++;
+ spin_unlock_irqrestore(&rss->res.lock, irqflags);

  set_page_container(pg, pc);
 }
```

```
-void container_rss_del(struct page_container *pc)
+void container_page_del(struct page_container *pc)
 {
+ struct page *page;
  struct rss_container *rss;
+ unsigned long irqflags;

+ 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);
+ if (page == ZERO_PAGE(0))
+  return;
+
+ spin_lock_irqsave(&rss->res.lock, irqflags);
+ pc->ref_cnt--;
+ if (!pc->ref_cnt) {
+  list_del(&pc->list);
+  set_page_container(page, NULL);
+ }
+ spin_unlock_irqrestore(&rss->res.lock, irqflags);
+
+ if (!pc->ref_cnt) {
+  res_counter_uncharge(&rss->res, 1);
+  kfree(pc);
+  css_put(&rss->css);
+ }
+
 }

 /*
```

Subject: [RFC][PATCH 2/3] Pagecache and RSS accounting hooks
Posted by Vaidyanathan Srinivas on Fri, 29 Jun 2007 06:21:01 GMT

View Forum Message <> Reply to Message

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     |   17 +++++++++++++++++
 mm/memory.c      |   18 +++++++++---------
 mm/migrate.c     |    4 ++--
 mm/rmap.c        |   12 ++++++------
 mm/swap_state.c  |   16 ++++++++++++++++
 mm/swapfile.c    |    4 ++--
 7 files changed, 54 insertions(+), 21 deletions(-)

--- linux-2.6.22-rc2-mm1.orig/fs/exec.c
+++ linux-2.6.22-rc2-mm1/fs/exec.c
@@ -321,7 +321,7 @@ 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))
   goto out;

  flush_dcache_page(page);
@@ -343,7 +343,7 @@ void install_arg_page(struct vm_area_str
  return;

 out_release:
- container_rss_release(pcont);
+ container_page_release(pcont);
 out:
  __free_page(page);
  force_sig(SIGKILL, current);
--- linux-2.6.22-rc2-mm1.orig/mm/filemap.c
+++ linux-2.6.22-rc2-mm1/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->mapping = NULL;
  mapping->nrpages--;
  __dec_zone_page_state(page, NR_FILE_PAGES);
@@ -440,6 +444,8 @@ 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);
@@ -453,6 +459,17 @@ 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))
+    container_page_add(pc);
+   else
+    BUG();
+  }
   radix_tree_preload_end();
  }
  return error;
--- linux-2.6.22-rc2-mm1.orig/mm/memory.c
+++ linux-2.6.22-rc2-mm1/mm/memory.c
@@ -1755,7 +1755,7 @@ 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))
   goto oom;

  /*
@@ -1791,7 +1791,7 @@ gotten:
   new_page = old_page;
   ret |= VM_FAULT_WRITE;
  } else
-  container_rss_release(pcont);
+  container_page_release(pcont);

  if (new_page)
   page_cache_release(new_page);
@@ -2217,7 +2217,7 @@ 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)) {
   ret = VM_FAULT_OOM;
   goto out;
  }
@@ -2235,7 +2235,7 @@ static int do_swap_page(struct mm_struct

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

@@ -2271,7 +2271,7 @@ unlock:
 out:
  return ret;
 out_nomap:
- container_rss_release(pcont);
+ container_page_release(pcont);
  pte_unmap_unlock(page_table, ptl);
  unlock_page(page);
  page_cache_release(page);
@@ -2302,7 +2302,7 @@ 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))
    goto oom_release;

   entry = mk_pte(page, vma->vm_page_prot);
@@ -2338,7 +2338,7 @@ unlock:
  pte_unmap_unlock(page_table, ptl);
  return VM_FAULT_MINOR;
 release_container:
- container_rss_release(pcont);
+ container_page_release(pcont);
```

```
 release:
  page_cache_release(page);
  goto unlock;
@@ -2442,7 +2442,7 @@ static int __do_fault(struct mm_struct *

  }

- if (container_rss_prepare(page, vma, &pcont)) {
+ if (container_page_prepare(page, vma->vm_mm, &pcont)) {
   fdata.type = VM_FAULT_OOM;
   goto out;
  }
@@ -2483,7 +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);
+  container_page_release(pcont);
   if (anon)
    page_cache_release(page);
   else
--- linux-2.6.22-rc2-mm1.orig/mm/migrate.c
+++ linux-2.6.22-rc2-mm1/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)) {
   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);
 }

 /*
--- linux-2.6.22-rc2-mm1.orig/mm/rmap.c
+++ linux-2.6.22-rc2-mm1/mm/rmap.c
@@ -578,14 +578,14 @@ void page_add_anon_rmap(struct page *pag
  if (atomic_inc_and_test(&page->_mapcount)) {
   __page_set_anon_rmap(page, vma, address);
   /* 0 -> 1 state change */
-  container_rss_add(pcont);
+  container_page_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);
+  container_page_release(pcont);
  }
 }

@@ -606,7 +606,7 @@ void page_add_new_anon_rmap(struct page
 {
  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);
+ container_page_add(pcont);
  __page_set_anon_rmap(page, vma, address);
 }

@@ -628,10 +628,10 @@ void page_add_file_rmap(struct page *pag
     * are not added to the container as they do not imply
     * RSS consumption  --xemul
     */
-   container_rss_add(pcont);
+   container_page_add(pcont);
   __inc_zone_page_state(page, NR_FILE_MAPPED);
  } else if (pcont)
-  container_rss_release(pcont);
+  container_page_release(pcont);
 }

 #ifdef CONFIG_DEBUG_VM
@@ -689,7 +689,7 @@ void page_remove_rmap(struct page *page,
   }

   if (pcont)
-   container_rss_del(pcont);
+   container_page_del(pcont);
   /*
    * It would be tidy to reset the PageAnon mapping here,
    * but that might overwrite a racing page_add_anon_rmap
--- linux-2.6.22-rc2-mm1.orig/mm/swap_state.c
+++ linux-2.6.22-rc2-mm1/mm/swap_state.c
@@ -17,6 +17,7 @@
 #include <linux/backing-dev.h>
 #include <linux/pagevec.h>
 #include <linux/migrate.h>
+#include <linux/rss_container.h>

 #include <asm/pgtable.h>

@@ -74,6 +75,8 @@ 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));
@@ -92,6 +95,17 @@ 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))
+    container_page_add(pc);
+   else
+    BUG();
+  }
  }
  return error;
 }
@@ -130,6 +144,8 @@ 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));
  set_page_private(page, 0);
  ClearPageSwapCache(page);
  total_swapcache_pages--;
--- linux-2.6.22-rc2-mm1.orig/mm/swapfile.c
+++ linux-2.6.22-rc2-mm1/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))
   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);
  return found;
 }
```

Subject: [RFC][PATCH 3/3] Pagecache reclaim
Posted by Vaidyanathan Srinivas on Fri, 29 Jun 2007 06:22:27 GMT

View Forum Message <> Reply to Message

Pagecache controller reclaim changes

Reclaim path needs performance improvement.
For now it is minor changes to include unmapped
pages in our list of page_container.

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

```
---
 mm/rss_container.c |    3 ---
 1 file changed, 3 deletions(-)

--- linux-2.6.22-rc2-mm1.orig/mm/rss_container.c
+++ linux-2.6.22-rc2-mm1/mm/rss_container.c
@@ -243,9 +243,6 @@ void container_rss_move_lists(struct pag
  struct rss_container *rss;
  struct page_container *pc;

- if (!page_mapped(pg))
-  return;
-
  pc = page_container(pg);
  if (pc == NULL)
   return;
```