Subject: [PATCH 1/3][RFC] Containers: Pagecache controller setup
Posted by Vaidyanathan Srinivas on Mon, 05 Mar 2007 14:52:38 GMT
View Forum Message <> Reply to Message

This patch adds basic infrastructure for pagecache accounting and
control subsystem within the container framework.

pagecache_usage and pagecache_limit files are created along with
every new container. Routines to extract subsystem pointers and
container pointers have been added.

Compile time kernel options and have been added along with Makefile
changes.

Signed-off-by: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
---
 include/linux/pagecache_acct.h | 53 +++++
 init/Kconfig | 7
 mm/Makefile | 1
 mm/pagecache_acct.c | 368 ++
 4 files changed, 429 insertions(+)

--- /dev/null
+++ linux-2.6.20/include/linux/pagecache_acct.h
@@ -0,0 +1,53 @@
+/*
+ * Pagecache controller - "Account and control pagecache usage"
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+ * GNU General Public License for more details.
+ *
+ * You should have received a copy of the GNU General Public License
+ * along with this program; if not, write to the Free Software
+ * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
+ *
+ * Copyright IBM Corporation, 2007
+ *
+ * Author: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
+ *
+ */
+

```
+#ifndef _LINUX_PAGECACHE_ACCT_H
+#define _LINUX_PAGECACHE_ACCT_H
+
+#include <linux/container.h>
+
+#ifdef CONFIG_CONTAINER_PAGECACHE_ACCT
+extern void pagecache_acct_init_page_ptr(struct page *page);
+extern void pagecache_acct_charge(struct page *page);
+extern void pagecache_acct_uncharge(struct page *page);
+extern int pagecache_acct_page_overlimit(struct page *page);
+extern int pagecache_acct_mapping_overlimit(struct address_space *mapping);
+extern int pagecache_acct_cont_overlimit(struct container *cont);
+extern int pagecache_acct_shrink_used(unsigned long nr_pages);
+#else
+static inline void pagecache_acct_init_page_ptr(struct page *page) {}
+static inline void pagecache_acct_charge(struct page *page) {}
+static inline void pagecache_acct_uncharge(struct page *page) {}
+static inline int pagecache_acct_page_overlimit(
+  struct page *page) { return 0; }
+static inline int pagecache_acct_mapping_overlimit(
+  struct address_space *mapping) { return 0; }
+static inline int pagecache_acct_cont_overlimit(
+  struct container *cont) { return 0; }
+static inline int pagecache_acct_shrink_used(
+  unsigned long nr_pages) { return 0; }
+#endif /* CONFIG_CONTAINER_PAGECACHE_ACCT */
+
+#endif /* _LINUX_PAGECACHE_ACCT_H */
+
+
--- linux-2.6.20.orig/init/Kconfig
+++ linux-2.6.20/init/Kconfig
@@ -313,6 +313,13 @@ config CONTAINER_MEMCONTROL
    Provides a simple Resource Controller for monitoring and
    controlling the total Resident Set Size of the tasks in a container

+config CONTAINER_PAGECACHE_ACCT
+ bool "Simple PageCache accounting & control container subsystem"
+ select CONTAINERS
+ help
+   Provides a simple Resource Controller for monitoring the
+   total pagecache memory consumed by the tasks in a container
+
 config RELAY
  bool "Kernel->user space relay support (formerly relayfs)"
  help
--- linux-2.6.20.orig/mm/Makefile
+++ linux-2.6.20/mm/Makefile
```

```
@@ -30,3 +30,4 @@ obj-$(CONFIG_FS_XIP) += filemap_xip.o
 obj-$(CONFIG_MIGRATION) += migrate.o
 obj-$(CONFIG_SMP) += allocpercpu.o
 obj-$(CONFIG_CONTAINER_MEMCONTROL) += memcontrol.o
+obj-$(CONFIG_CONTAINER_PAGECACHE_ACCT) += pagecache_acct.o
--- /dev/null
+++ linux-2.6.20/mm/pagecache_acct.c
@@ -0,0 +1,368 @@
+/*
+ * Pagecache controller - "Account and control pagecache usage"
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ * GNU General Public License for more details.
+ *
+ * You should have received a copy of the GNU General Public License
+ * along with this program; if not, write to the Free Software
+ * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
+ *
+ * Copyright IBM Corporation, 2007
+ *
+ * Author: Vaidyanathan Srinivasan <svaidy@linux.vnet.ibm.com>
+ *
+ */
+
+#include <linux/module.h>
+#include <linux/container.h>
+#include <linux/fs.h>
+#include <linux/mm.h>
+#include <linux/mm_types.h>
+#include <linux/uaccess.h>
+#include <asm/div64.h>
+#include <linux/pagecache_acct.h>
+
+/*
+ * Convert unit from pages to kilobytes
+ */
+#define K(x) ((x) << (PAGE_SHIFT - 10))
+/*
+ * Convert unit from kilobytes to pages
+ */
+#define K_to_pages(x) ((x) >> (PAGE_SHIFT -10))
```

```
+
+/* Limits for user string */
+
+#define MAX_LIMIT_STRING  25
+
+/* nr_pages above limit to start reclaim */
+
+#define NR_PAGES_RECLAIM_THRESHOLD 64
+
+struct pagecache_acct {
+ struct container_subsys_state css;
+ spinlock_t lock;
+ atomic_t count; /*Pagecache pages added*/
+ atomic_t removed_count; /*Pagecache pages removed*/
+ unsigned int limit; /* Pagecache usage limit in kiB */
+};
+
+/*Failure counters for debugging*/
+static atomic_t failed_count; /*Page charge failures?*/
+static atomic_t failed_removed_count; /*Page uncharge failure?*/
+static atomic_t reclaim_count; /*Overlimit direct page reclaim run count */
+
+static struct container_subsys pagecache_acct_subsys;
+
+static inline struct pagecache_acct *container_pca(struct container *cont)
+{
+ return container_of(
+   container_subsys_state(cont, &pagecache_acct_subsys),
+       struct pagecache_acct, css);
+}
+
+static int pagecache_acct_create(struct container_subsys *ss,
+    struct container *cont)
+{
+ struct pagecache_acct *pca = kzalloc(sizeof(*pca), GFP_KERNEL);
+ if (!pca)
+  return -ENOMEM;
+ spin_lock_init(&pca->lock);
+ cont->subsys[pagecache_acct_subsys.subsys_id] = &pca->css;
+ return 0;
+}
+
+static void pagecache_acct_destroy(struct container_subsys *ss,
+     struct container *cont)
+{
+ kfree(container_pca(cont));
+}
+
```

```
+static unsigned int pagecache_get_usage(struct pagecache_acct *pca)
+{
+ unsigned int count, removed_count, pagecache_used_kB;
+
+ count = (unsigned int) atomic_read(&pca->count);
+ removed_count = (unsigned int) atomic_read(&pca->removed_count);
+ /* Take care of roll over in the counters */
+ if (count >= removed_count)
+  pagecache_used_kB = count - removed_count;
+ else
+  pagecache_used_kB = ~0UL - (removed_count - count) + 1;
+
+ /* Convert unit from pages into kB */
+ pagecache_used_kB = K(pagecache_used_kB);
+
+ return pagecache_used_kB;
+}
+
+static ssize_t pagecache_usage_read(struct container *cont,
+
+        struct cftype *cft,
+        struct file *file,
+        char __user *buf,
+        size_t nbytes, loff_t *ppos)
+{
+ struct pagecache_acct *pca = container_pca(cont);
+ char usagebuf[64];
+ char *s = usagebuf;
+
+ s += sprintf(s, "%u kB \n", pagecache_get_usage(pca));
+ return simple_read_from_buffer(buf, nbytes, ppos, usagebuf,
+      s - usagebuf);
+}
+
+static ssize_t pagecache_debug_read(struct container *cont,
+        struct cftype *cft,
+        struct file *file,
+        char __user *buf,
+        size_t nbytes, loff_t *ppos)
+{
+ struct pagecache_acct *pca = container_pca(cont);
+ char usagebuf[64];
+ char *s = usagebuf;
+
+ s += sprintf(s, "%u kB Cnt: %u, %u Failed: %u, %u Reclaim: %u\n",
+    pagecache_get_usage(pca),
+    (unsigned int) atomic_read(&pca->count),
+    (unsigned int) atomic_read(&pca->removed_count),
```

```
+    (unsigned int) atomic_read(&failed_count),
+    (unsigned int) atomic_read(&failed_removed_count),
+    (unsigned int) atomic_read(&reclaim_count));
+
+ return simple_read_from_buffer(buf, nbytes, ppos, usagebuf,
+      s - usagebuf);
+}
+
+static ssize_t pagecache_limit_read(struct container *cont,
+        struct cftype *cft,
+        struct file *file,
+        char __user *buf,
+        size_t nbytes, loff_t *ppos)
+{
+ struct pagecache_acct *pca = container_pca(cont);
+ char usagebuf[64];
+ char *s = usagebuf;
+
+ s += sprintf(s, "%u kB\n", pca->limit);
+
+ return simple_read_from_buffer(buf, nbytes, ppos, usagebuf,
+      s - usagebuf);
+}
+
+static ssize_t pagecache_limit_write(struct container *cont,
+        struct cftype *cft,
+        struct file *file,
+        const char __user *buf,
+        size_t nbytes, loff_t *ppos)
+{
+ struct pagecache_acct *pca = container_pca(cont);
+ char buffer[MAX_LIMIT_STRING];
+ unsigned int limit;
+ unsigned int nr_pages;
+
+ if( nbytes > (MAX_LIMIT_STRING-1) || nbytes < 1 )
+  return -EINVAL;
+
+ if (copy_from_user(buffer, buf, nbytes))
+  return -EFAULT;
+
+ buffer[nbytes] = '\0'; /* Null termination */
+ limit = simple_strtoul(buffer, NULL, 10);
+ /* Round it off to lower 4K page boundary */
+ limit &= ~0x3;
+
+ /* Set the value in pca struct (atomic store). No read-modify-update */
+ pca->limit = limit;
```

```
+
+ /* Check for overlimit and initiate reclaim if needed */
+ /* The limits have changed now */
+ if ((nr_pages = pagecache_acct_cont_overlimit(cont))) {
+  pagecache_acct_shrink_used(nr_pages);
+ }
+ return nbytes;
+}
+
+static struct cftype cft_usage = {
+ .name = "pagecache_usage",
+ .read = pagecache_usage_read,
+};
+
+static struct cftype cft_debug = {
+ .name = "pagecache_debug",
+ .read = pagecache_debug_read,
+};
+
+static struct cftype cft_limit = {
+ .name = "pagecache_limit",
+ .read = pagecache_limit_read,
+ .write = pagecache_limit_write,
+};
+
+static int pagecache_acct_populate(struct container_subsys *ss,
+     struct container *cont)
+{
+ int rc;
+ rc = container_add_file(cont, &cft_usage);
+ if (rc)
+  return rc;
+ rc = container_add_file(cont, &cft_debug);
+ if (rc)
+  /* Cleanup with container_remove_file()? */
+  return rc;
+ rc = container_add_file(cont, &cft_limit);
+ return rc;
+}
+
+static struct container *mapping_container(struct address_space *mapping)
+{
+ if ((unsigned long) mapping & PAGE_MAPPING_ANON)
+  mapping = NULL;
+ if(!mapping)
+  return NULL;
+ if (!mapping->container) {
+  printk( KERN_DEBUG "Null Container in mapping: %p\n", mapping);
```

```
+ }
+ return mapping->container;
+}
+
+static struct container *page_container(struct page *page)
+{
+ return mapping_container(page->mapping);
+}
+
+void pagecache_acct_init_page_ptr(struct page *page)
+{
+ struct address_space *mapping;
+ mapping = page->mapping;
+ if ((unsigned long) mapping & PAGE_MAPPING_ANON)
+  mapping = NULL;
+ BUG_ON(!mapping);
+ if (current) {
+  if(!mapping->container)
+   mapping->container = task_container(current, &pagecache_acct_subsys);
+ } else
+  mapping->container = NULL;
+}
+
+void pagecache_acct_charge(struct page *page)
+{
+ struct container *cont;
+ struct pagecache_acct *pca;
+ unsigned int nr_pages;
+
+ if (pagecache_acct_subsys.subsys_id < 0) return;
+ cont = page_container(page);
+ if (cont) {
+  pca = container_pca(cont);
+  BUG_ON(!pca);
+  atomic_inc(&pca->count);
+ } else {
+  /* page->container is null??? */
+  printk(KERN_WARNING "pca_charge:page_container null\n");
+  atomic_inc(&failed_count);
+ }
+ /* Check for overlimit and initiate reclaim if needed */
+ if ((nr_pages = pagecache_acct_page_overlimit(page))) {
+  pagecache_acct_shrink_used(nr_pages);
+ }
+}
+
+void pagecache_acct_uncharge(struct page * page)
+{
```

```
+ struct container *cont;
+ struct pagecache_acct *pca;
+
+ if (pagecache_acct_subsys.subsys_id < 0) return;
+ cont = page_container(page);
+ if (cont) {
+  pca = container_pca(cont);
+  BUG_ON(!pca);
+  atomic_inc(&pca->removed_count);
+ } else {
+  /* page->container is null??? */
+  printk(KERN_WARNING "pca_uncharge:page_container null\n");
+  atomic_inc(&failed_removed_count);
+ }
+}
+
+int pagecache_acct_page_overlimit(struct page *page)
+{
+ struct container *cont;
+
+ if (pagecache_acct_subsys.subsys_id < 0)
+  return 0;
+ cont = page_container(page);
+ if (!cont)
+  return 0;
+ return pagecache_acct_cont_overlimit(cont);
+}
+
+int pagecache_acct_mapping_overlimit(struct address_space *mapping)
+{
+ struct container *cont;
+
+ if (pagecache_acct_subsys.subsys_id < 0)
+  return 0;
+ cont = mapping_container(mapping);
+ if (!cont)
+  return 0;
+ return pagecache_acct_cont_overlimit(cont);
+}
+
+int pagecache_acct_cont_overlimit(struct container *cont)
+{
+ struct pagecache_acct *pca;
+ unsigned int used, limit;
+
+ if (pagecache_acct_subsys.subsys_id < 0)
+  return 0;
+
```

```
+ if (!cont)
+  return 0;
+ pca = container_pca(cont);
+ used = pagecache_get_usage(pca);
+ limit = pca->limit;
+ if( limit && (used > limit) )
+  return K_to_pages(used - limit);
+ else
+  return 0;
+}
+
+extern unsigned long shrink_all_pagecache_memory(unsigned long nr_pages);
+
+int pagecache_acct_shrink_used(unsigned long nr_pages)
+{
+ unsigned long ret = 0;
+ atomic_inc(&reclaim_count);
+ return 0;
+}
+
+static struct container_subsys pagecache_acct_subsys = {
+ .name = "pagecache_acct",
+ .create = pagecache_acct_create,
+ .destroy = pagecache_acct_destroy,
+ .populate = pagecache_acct_populate,
+ .subsys_id = -1,
+};
+
+int __init init_pagecache_acct(void)
+{
+ int id = container_register_subsys(&pagecache_acct_subsys);
+ int rc = 0; /* Default to success */
+ /* Flag failure */
+ if (id < 0)
+  rc = id;
+ return rc;
+}
+
+__initcall(init_pagecache_acct);
+

--
```