
Subject: [-mm PATCH 0/9] Memory controller introduction (v4)

Posted by [Balbir Singh](#) on Fri, 27 Jul 2007 20:09:37 GMT

[View Forum Message](#) <> [Reply to Message](#)

Here's version 4 of the memory controller.

Changelog since version 3

1. Ported to v11 of the containers patchset (2.6.23-rc1-mm1). Paul Menage helped immensely with a detailed review of v3
2. Reclaim is retried to allow reclaim of pages coming in as a result of mapped pages reclaim (swap cache growing as a result of RSS reclaim)
3. `page_referenced()` is now container aware. During container reclaim, references from other containers do not prevent a page from being reclaimed from a non-referencing container
4. Fixed a possible race condition spotted by YAMAMOTO Takashi

Changelog since version 2

1. Improved error handling in `mm/memory.c` (spotted by YAMAMOTO Takashi)
2. Test results included
3. `try_to_free_mem_container_pages()` bug fix (`sc->may_writepage` is now set to `!laptop_mode`)

Changelog since version 1

1. Fixed some compile time errors (in `mm/migrate.c` from Vaidyanathan S)
2. Fixed a panic seen when `LIST_DEBUG` is enabled
3. Added a mechanism to control whether we track page cache or both page cache and mapped pages (as requested by Pavel)
4. Dave Hansen provided detail review comments on the code.

This patchset implements another version of the memory controller. These patches have been through a big churn, the first set of patches were posted last year and earlier this year at <http://lkml.org/lkml/2007/2/19/10>

This patchset draws from the patches listed above and from some of the contents of the patches posted by Vaidyanathan for page cache control. <http://lkml.org/lkml/2007/6/20/92>

At OLS, the resource management BOF, it was discussed that we need to manage RSS and unmapped page cache together. This patchset is a step towards that

TODO's

1. Add memory controller water mark support. Reclaim on high water mark
2. Add support for shrinking on limit change

3. Add per zone per container LRU lists (this is being actively worked on by Pavel Emelianov)
4. Figure out a better CLUI for the controller
5. Add better statistics
6. Explore using read_unit64() as recommended by Paul Menage
(NOTE: read_ulong() would also be nice to have)

In case you have been using/testing the RSS controller, you'll find that this controller works slower than the RSS controller. The reason being that both swap cache and page cache is accounted for, so pages do go out to swap upon reclaim (they cannot live in the swap cache).

Any test output, feedback, comments, suggestions are welcome! I am committed to fixing any bugs and improving the performance of the memory controller. Do not hesitate to send any fixes, request for fixes that is required.

Using the patches

1. Enable Memory controller configuration
2. Compile and boot the new kernel
3. mount -t container container -o mem_container /container
will mount the memory controller to the /container mount point
4. mkdir /container/a
5. echo \$\$ > /container/a/tasks (add tasks to the new container)
6. echo -n <num_pages> > /container/a/mem_limit
example
echo -n 204800 > /container/a/mem_limit, sets the limit to 800 MB on a system with 4K page size
7. run tasks, see the memory controller work
8. Report results, provide feedback
9. Develop/use new patches and go to step 1

Test Results

Results for version 3 of the patch were posted at
<http://lwn.net/Articles/242554/>

The code was also tested on a power box with regular machine usage scenarios, the config disabled and with a stress suite that touched all the memory in the system and was limited in a container.

series

```
res_counters_infra.patch
mem-control-setup.patch
mem-control-accounting-setup.patch
mem-control-accounting.patch
mem-control-task-migration.patch
```

mem-control-lru-and-reclaim.patch
mem-control-out-of-memory.patch
mem-control-choose-rss-vs-rss-and-pagecache.patch
mem-control-per-container-page-referenced.patch

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

Subject: [-mm PATCH 1/9] Memory controller resource counters (v4)
Posted by [Balbir Singh](#) on Fri, 27 Jul 2007 20:09:47 GMT
[View Forum Message](#) <> [Reply to Message](#)

From: Pavel Emelianov <xemul@openvz.org>

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>

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

```
include/linux/res_counter.h | 102 ++++++
init/Kconfig                |  7 ++
kernel/Makefile             |  1
kernel/res_counter.c        | 120 ++++++
4 files changed, 230 insertions(+)
```

```
diff -puN /dev/null include/linux/res_counter.h
--- /dev/null 2007-06-01 20:42:04.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/res_counter.h 2007-07-28 01:12:48.000000000
+0530
@@ -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 unsuccessful 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
+ *
+ * @counter: the counter in question
+ * @member: the field to work with (see RES_xxx below)
+ * @buf: the buffer to operate on,...
+ * @nbytes: its size...
+ * @pos: and the offset.
+ */
+
+ ssize_t res_counter_read(struct res_counter *counter, int member,

```

```

+ const char __user *buf, size_t nbytes, loff_t *pos);
+ssize_t res_counter_write(struct res_counter *counter, 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 *counter);
+
+/*
+ * charge - try to consume more resource.
+ *
+ * @counter: 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 counter->usage will exceed the
+ * counter->limit _locked call expects the counter->lock to be taken
+ */
+
+int res_counter_charge_locked(struct res_counter *counter, unsigned long val);
+int res_counter_charge(struct res_counter *counter, unsigned long val);
+
+/*
+ * uncharge - tell that some portion of the resource is released
+ *
+ * @counter: 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 counter->lock to be taken
+ */
+
+void res_counter_uncharge_locked(struct res_counter *counter, unsigned long val);
+void res_counter_uncharge(struct res_counter *counter, unsigned long val);
+
+#endif

```

```
diff -puN init/Kconfig~res_counters_infra init/Kconfig
--- linux-2.6.23-rc1-mm1/init/Kconfig~res_counters_infra 2007-07-28 01:12:48.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/init/Kconfig 2007-07-28 01:12:48.000000000 +0530
@@ -321,6 +321,13 @@ config CPUSETS
```

Say N if unsure.

```
+config RESOURCE_COUNTERS
+ bool "Resource counters"
+ help
+   This option enables controller independent resource accounting
+   infrastructure that works with containers
+ depends on CONTAINERS
+
+ config SYSFS_DEPRECATED
+   bool "Create deprecated sysfs files"
+   default y
diff -puN kernel/Makefile~res_counters_infra kernel/Makefile
--- linux-2.6.23-rc1-mm1/kernel/Makefile~res_counters_infra 2007-07-28 01:12:48.000000000
+0530
+++ linux-2.6.23-rc1-mm1-balbir/kernel/Makefile 2007-07-28 01:12:48.000000000 +0530
@@ -57,6 +57,7 @@ obj-$(CONFIG_RELAY) += relay.o
obj-$(CONFIG_SYSCTL) += utsname_sysctl.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 -puN /dev/null kernel/res_counter.c
--- /dev/null 2007-06-01 20:42:04.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/kernel/res_counter.c 2007-07-28 01:12:48.000000000 +0530
@@ -0,0 +1,120 @@
+/*
+ * 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 *counter)
+{
+ spin_lock_init(&counter->lock);
+ counter->limit = (unsigned long)LONG_MAX;
+}
+
+int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
+{
+ if (counter->usage > (counter->limit - val)) {
+ counter->failcnt++;
+ return -ENOMEM;
+ }
+
+ counter->usage += val;
+ return 0;
+}
+
+int res_counter_charge(struct res_counter *counter, unsigned long val)
+{
+ int ret;
+ unsigned long flags;
+
+ spin_lock_irqsave(&counter->lock, flags);
+ ret = res_counter_charge_locked(counter, val);
+ spin_unlock_irqrestore(&counter->lock, flags);
+ return ret;
+}
+
+void res_counter_uncharge_locked(struct res_counter *counter, unsigned long val)
+{
+ if (WARN_ON(counter->usage < val))
+ val = counter->usage;
+
+ counter->usage -= val;
+}
+
+void res_counter_uncharge(struct res_counter *counter, unsigned long val)
+{
+ unsigned long flags;
+
+ spin_lock_irqsave(&counter->lock, flags);
+ res_counter_uncharge_locked(counter, val);
+ spin_unlock_irqrestore(&counter->lock, flags);
+}
+
+static inline unsigned long *res_counter_member(struct res_counter *counter,
+ int member)

```

```

+{
+ switch (member) {
+ case RES_USAGE:
+ return &counter->usage;
+ case RES_LIMIT:
+ return &counter->limit;
+ case RES_FAILCNT:
+ return &counter->failcnt;
+ };
+
+ BUG();
+ return NULL;
+}
+
+ssize_t res_counter_read(struct res_counter *counter, 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(counter, 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 *counter, 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(counter, member);
+ *val = tmp;
+ ret = nbytes;
+out_free:
+ kfree(buf);
+out:
+ return ret;
+}

```

—

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

Subject: [-mm PATCH 2/9] Memory controller containers setup (v4)
Posted by [Balbir Singh](#) on Fri, 27 Jul 2007 20:09:58 GMT
[View Forum Message](#) <> [Reply to Message](#)

Changelong

1. use depends instead of select in init/Kconfig
2. Port to v11
3. Clean up the usage of names (container files) for v11

Setup the memory container and add basic hooks and controls to integrate and work with the container.

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

```

include/linux/container_subsys.h | 6 +
include/linux/memcontrol.h      | 21 ++++++
init/Kconfig                    | 7 ++
mm/Makefile                     | 1
mm/memcontrol.c                 | 127 ++++++
5 files changed, 162 insertions(+)

```

```

diff -puN include/linux/container_subsys.h~mem-control-setup include/linux/container_subsys.h
--- linux-2.6.23-rc1-mm1/include/linux/container_subsys.h~mem-control-setup 2007-07-28
01:12:49.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/container_subsys.h 2007-07-28

```

```

01:12:49.000000000 +0530
@@ -30,3 +30,9 @@ SUBSYS(ns)
#endif

/* */
+
+#ifdef CONFIG_CONTAINER_MEM_CONT
+SUBSYS(mem_container)
+#endif
+
+/* */
diff -puN /dev/null include/linux/memcontrol.h
--- /dev/null 2007-06-01 20:42:04.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/memcontrol.h 2007-07-28 01:12:49.000000000
+0530
@@ -0,0 +1,21 @@
+/* memcontrol.h - Memory Controller
+ *
+ * Copyright IBM Corporation, 2007
+ * Author Balbir Singh <balbir@linux.vnet.ibm.com>
+ *
+ * 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.
+ */
+
+#ifndef _LINUX_MEMCONTROL_H
+#define _LINUX_MEMCONTROL_H
+
+#endif /* _LINUX_MEMCONTROL_H */
+
diff -puN init/Kconfig~mem-control-setup init/Kconfig
--- linux-2.6.23-rc1-mm1/init/Kconfig~mem-control-setup 2007-07-28 01:12:49.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/init/Kconfig 2007-07-28 01:12:49.000000000 +0530
@@ -357,6 +357,13 @@ config CONTAINER_NS
    for instance virtual servers and checkpoint/restart
    jobs.

+config CONTAINER_MEM_CONT
+ bool "Memory controller for containers"
+ depends on CONTAINERS && RESOURCE_COUNTERS
+ help

```

```

+ Provides a memory controller that manages both page cache and
+ RSS memory.
+
config PROC_PID_CPUSET
    bool "Include legacy /proc/<pid>/cpuset file"
    depends on CPUSETS
diff -puN mm/Makefile~mem-control-setup mm/Makefile
--- linux-2.6.23-rc1-mm1/mm/Makefile~mem-control-setup 2007-07-28 01:12:49.000000000
+0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/Makefile 2007-07-28 01:12:49.000000000 +0530
@@ -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_CONTAINER_MEM_CONT) += memcontrol.o

diff -puN /dev/null mm/memcontrol.c
--- /dev/null 2007-06-01 20:42:04.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/memcontrol.c 2007-07-28 01:12:49.000000000 +0530
@@ -0,0 +1,127 @@
+/* memcontrol.c - Memory Controller
+ *
+ * Copyright IBM Corporation, 2007
+ * Author Balbir Singh <balbir@linux.vnet.ibm.com>
+ *
+ * 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.
+ */
+
+#include <linux/res_counter.h>
+#include <linux/memcontrol.h>
+#include <linux/container.h>
+
+struct container_subsys mem_container_subsys;
+
+/*
+ * The memory controller data structure. The memory controller controls both
+ * page cache and RSS per container. We would eventually like to provide
+ * statistics based on the statistics developed by Rik Van Riel for clock-pro,
+ * to help the administrator determine what knobs to tune.
+ */

```

```

+ * TODO: Add a water mark for the memory controller. Reclaim will begin when
+ * we hit the water mark.
+ */
+struct mem_container {
+ struct container_subsys_state css;
+ /*
+  * the counter to account for memory usage
+  */
+ struct res_counter res;
+};
+
+/*
+ * A meta page is associated with every page descriptor. The meta page
+ * helps us identify information about the container
+ */
+struct meta_page {
+ struct list_head lru; /* per container LRU list */
+ struct page *page;
+ struct mem_container *mem_container;
+};
+
+
+static inline
+struct mem_container *mem_container_from_cont(struct container *cont)
+{
+ return container_of(container_subsys_state(cont,
+ mem_container_subsys_id), struct mem_container,
+ css);
+}
+
+static ssize_t mem_container_read(struct container *cont, struct cftype *cft,
+ struct file *file, char __user *userbuf, size_t nbytes,
+ loff_t *ppos)
+{
+ return res_counter_read(&mem_container_from_cont(cont)->res,
+ cft->private, userbuf, nbytes, ppos);
+}
+
+static ssize_t mem_container_write(struct container *cont, struct cftype *cft,
+ struct file *file, const char __user *userbuf,
+ size_t nbytes, loff_t *ppos)
+{
+ return res_counter_write(&mem_container_from_cont(cont)->res,
+ cft->private, userbuf, nbytes, ppos);
+}
+
+static struct cftype mem_container_files[] = {
+ {

```

```

+ .name = "usage",
+ .private = RES_USAGE,
+ .read = mem_container_read,
+ },
+ {
+ .name = "limit",
+ .private = RES_LIMIT,
+ .write = mem_container_write,
+ .read = mem_container_read,
+ },
+ {
+ .name = "failcnt",
+ .private = RES_FAILCNT,
+ .read = mem_container_read,
+ },
+ };
+
+static struct container_subsys_state *
+mem_container_create(struct container_subsys *ss, struct container *cont)
+{
+ struct mem_container *mem;
+
+ mem = kzalloc(sizeof(struct mem_container), GFP_KERNEL);
+ if (!mem)
+ return -ENOMEM;
+
+ res_counter_init(&mem->res);
+ return &mem->css;
+}
+
+static void mem_container_destroy(struct container_subsys *ss,
+ struct container *cont)
+{
+ kfree(mem_container_from_cont(cont));
+}
+
+static int mem_container_populate(struct container_subsys *ss,
+ struct container *cont)
+{
+ return container_add_files(cont, ss, mem_container_files,
+ ARRAY_SIZE(mem_container_files));
+}
+
+struct container_subsys mem_container_subsys = {
+ .name = "memory",
+ .subsys_id = mem_container_subsys_id,
+ .create = mem_container_create,
+ .destroy = mem_container_destroy,

```

```
+ .populate = mem_container_populate,  
+ .early_init = 0,  
+};
```

—

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

Subject: [-mm PATCH 3/9] Memory controller accounting setup (v4)
Posted by [Balbir Singh](#) on Fri, 27 Jul 2007 20:10:08 GMT
[View Forum Message](#) <> [Reply to Message](#)

From: Pavel Emelianov <xemul@openvz.org>
Changelog

As per Paul's review comments

1. Drop `css_get()` for the root memory container
2. Use `mem_container_from_task()` as an optimization instead of using `mem_container_from_cont()` along with `task_container`.

Basic setup routines, the `mm_struct` has a pointer to the container that it belongs to and the page has a `meta_page` associated with it.

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

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

```
include/linux/memcontrol.h | 35 ++++++  
include/linux/mm_types.h   |  4 +++  
include/linux/sched.h      |  4 +++  
kernel/fork.c              | 10 +++++-  
mm/memcontrol.c            | 57 ++++++-----  
5 files changed, 103 insertions(+), 7 deletions(-)
```

```
diff -puN include/linux/memcontrol.h~mem-control-accounting-setup include/linux/memcontrol.h  
--- linux-2.6.23-rc1-mm1/include/linux/memcontrol.h~mem-control-accounting-setup 2007-07-28  
01:12:49.000000000 +0530  
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/memcontrol.h 2007-07-28 01:12:49.000000000
```

```

+0530
@@ -3,6 +3,9 @@
 * Copyright IBM Corporation, 2007
 * Author Balbir Singh <balbir@linux.vnet.ibm.com>
 *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
 * 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
@@ -17,5 +20,37 @@
#ifndef _LINUX_MEMCONTROL_H
#define _LINUX_MEMCONTROL_H

+struct mem_container;
+struct meta_page;
+
+#ifdef CONFIG_CONTAINER_MEM_CONT
+
+extern void mm_init_container(struct mm_struct *mm, struct task_struct *p);
+extern void mm_free_container(struct mm_struct *mm);
+extern void page_assign_meta_page(struct page *page, struct meta_page *mp);
+extern struct meta_page *page_get_meta_page(struct page *page);
+
+#else /* CONFIG_CONTAINER_MEM_CONT */
+static inline void mm_init_container(struct mm_struct *mm,
+    struct task_struct *p)
+{
+}
+
+static inline void mm_free_container(struct mm_struct *mm)
+{
+}
+
+static inline void page_assign_meta_page(struct page *page,
+    struct meta_page *mp)
+{
+}
+
+static inline struct meta_page *page_get_meta_page(struct page *page)
+{
+    return NULL;
+}
+
+#endif /* CONFIG_CONTAINER_MEM_CONT */
+
#endif /* _LINUX_MEMCONTROL_H */

```

```

diff -puN include/linux/mm_types.h~mem-control-accounting-setup include/linux/mm_types.h
--- linux-2.6.23-rc1-mm1/include/linux/mm_types.h~mem-control-accounting-setup 2007-07-28
01:12:49.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/mm_types.h 2007-07-28 01:12:49.000000000
+0530
@@ -5,6 +5,7 @@
#include <linux/threads.h>
#include <linux/list.h>
#include <linux/spinlock.h>
+#include <linux/memcontrol.h>

struct address_space;

@@ -83,6 +84,9 @@ struct page {
    unsigned int gfp_mask;
    unsigned long trace[8];
#endif
+#ifdef CONFIG_CONTAINER_MEM_CONT
+ struct meta_page *meta_page;
+#endif
};

#endif /* _LINUX_MM_TYPES_H */
diff -puN include/linux/sched.h~mem-control-accounting-setup include/linux/sched.h
--- linux-2.6.23-rc1-mm1/include/linux/sched.h~mem-control-accounting-setup 2007-07-28
01:12:49.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/sched.h 2007-07-28 01:12:49.000000000 +0530
@@ -89,6 +89,7 @@ struct sched_param {

#include <asm/processor.h>

+struct mem_container;
struct exec_domain;
struct futex_pi_state;
struct bio;
@@ -433,6 +434,9 @@ struct mm_struct {
    /* aio bits */
    rwlock_t ioctx_list_lock;
    struct kiocx *ioctx_list;
+#ifdef CONFIG_CONTAINER_MEM_CONT
+ struct mem_container *mem_container;
+#endif
};

struct sighand_struct {
diff -puN kernel/fork.c~mem-control-accounting-setup kernel/fork.c
--- linux-2.6.23-rc1-mm1/kernel/fork.c~mem-control-accounting-setup 2007-07-28

```

01:12:49.000000000 +0530

+++ linux-2.6.23-rc1-mm1-balbir/kernel/fork.c 2007-07-28 01:12:49.000000000 +0530

@@ -328,7 +328,7 @@ __cacheline_aligned_in_smp DEFINE_SPINLO

#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 *p)

```
{
    atomic_set(&mm->mm_users, 1);
    atomic_set(&mm->mm_count, 1);
```

@@ -345,11 +345,14 @@ 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, p);

```
if (likely(!mm_alloc_pgd(mm))) {
    mm->def_flags = 0;
    return mm;
}
```

+

+ mm_free_container(mm);

```
free_mm(mm);
return NULL;
```

}

@@ -364,7 +367,7 @@ 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;
```

}

@@ -378,6 +381,7 @@ 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);
}
```

@@ -498,7 +502,7 @@ 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))
diff -puN mm/memcontrol.c~mem-control-accounting-setup mm/memcontrol.c
--- linux-2.6.23-rc1-mm1/mm/memcontrol.c~mem-control-accounting-setup 2007-07-28
01:12:49.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/memcontrol.c 2007-07-28 01:12:49.000000000 +0530
@@ -3,6 +3,9 @@
 * Copyright IBM Corporation, 2007
 * Author Balbir Singh <balbir@linux.vnet.ibm.com>
 *
+ * Copyright 2007 OpenVZ SWsoft Inc
+ * Author: Pavel Emelianov <xemul@openvz.org>
+ *
 * 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
@@ -17,6 +20,7 @@
#include <linux/res_counter.h>
#include <linux/memcontrol.h>
#include <linux/container.h>
+#include <linux/mm.h>

struct container_subsys mem_container_subsys;

@@ -35,6 +39,13 @@ struct mem_container {
 * the counter to account for memory usage
 */
    struct res_counter res;
+ /*
+  * Per container active and inactive list, similar to the
+  * per zone LRU lists.
+  * TODO: Consider making these lists per zone
+  */
+ struct list_head active_list;
+ struct list_head inactive_list;
};

/*
@@ -56,6 +67,37 @@ struct mem_container *mem_container_from
    css);
}

+static inline
+struct mem_container *mem_container_from_task(struct task_struct *p)
+{
+ return container_of(task_subsys_state(p, mem_container_subsys_id),

```

```

+ struct mem_container, css);
+}
+
+void mm_init_container(struct mm_struct *mm, struct task_struct *p)
+{
+ struct mem_container *mem;
+
+ mem = mem_container_from_task(p);
+ css_get(&mem->css);
+ mm->mem_container = mem;
+}
+
+void mm_free_container(struct mm_struct *mm)
+{
+ css_put(&mm->mem_container->css);
+}
+
+void page_assign_meta_page(struct page *page, struct meta_page *mp)
+{
+ page->meta_page = mp;
+}
+
+struct meta_page *page_get_meta_page(struct page *page)
+{
+ return page->meta_page;
+}
+
+static ssize_t mem_container_read(struct container *cont, struct cftype *cft,
+ struct file *file, char __user *userbuf, size_t nbytes,
+ loff_t *ppos)
@@ -91,14 +133,21 @@ static struct cftype mem_container_files
+},
+};

+static struct mem_container init_mem_container;
+
+static struct container_subsys_state *
+mem_container_create(struct container_subsys *ss, struct container *cont)
+{
+ struct mem_container *mem;

- mem = kzalloc(sizeof(struct mem_container), GFP_KERNEL);
- if (!mem)
- return -ENOMEM;
+ if (unlikely((cont->parent) == NULL)) {
+ mem = &init_mem_container;
+ init_mm.mem_container = mem;
+ } else

```

```

+ mem = kzalloc(sizeof(struct mem_container), GFP_KERNEL);
+
+ if (mem == NULL)
+ return NULL;

    res_counter_init(&mem->res);
    return &mem->css;
@@ -123,5 +172,5 @@ struct container_subsys mem_container_su
    .create = mem_container_create,
    .destroy = mem_container_destroy,
    .populate = mem_container_populate,
- .early_init = 0,
+ .early_init = 1,
};

```

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

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

Subject: [-mm PATCH 4/9] Memory controller memory accounting (v4)
 Posted by [Balbir Singh](#) on Fri, 27 Jul 2007 20:10:18 GMT
[View Forum Message](#) <> [Reply to Message](#)

Changelog for v3

1. Fix a probable leak with meta_page's (pointed out by Paul Menage)
2. Introduce a wrapper around mem_container_uncharge for uncharging pages
 mem_container_uncharge_page()

Changelog

1. Improved error handling, uncharge on errors and check to see if we are
 leaking pages (review by YAMAMOTO Takashi)

Add the accounting hooks. The accounting is carried out for RSS and Page
 Cache (unmapped) pages. There is now a common limit and accounting for both.
 The RSS accounting is accounted at page_add_*_rmap() and page_remove_rmap()
 time. Page cache is accounted at add_to_page_cache(),
 __delete_from_page_cache(). Swap cache is also accounted for.

Each page's meta_page is protected with a bit in page flags, this makes handling of race conditions involving simultaneous mappings of a page easier. A reference count is kept in the meta_page to deal with cases where a page might be unmapped from the RSS of all tasks, but still lives in the page cache.

Credits go to Vaidyanathan Srinivasan for helping with reference counting work of the meta page. Almost all of the page cache accounting code has help from Vaidyanathan Srinivasan.

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

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

```
include/linux/memcontrol.h | 20 ++++++
include/linux/page-flags.h | 3
mm/filemap.c               | 12 +++
mm/memcontrol.c            | 139 ++++++
mm/memory.c               | 44 ++++++
mm/migrate.c              | 6 +
mm/page_alloc.c           | 3
mm/rmap.c                 | 16 ++++
mm/swap_state.c           | 12 +++
mm/swapfile.c             | 40 ++++++
10 files changed, 270 insertions(+), 25 deletions(-)
```

```
diff -puN include/linux/memcontrol.h~mem-control-accounting include/linux/memcontrol.h
--- linux-2.6.23-rc1-mm1/include/linux/memcontrol.h~mem-control-accounting 2007-07-28
01:12:49.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/memcontrol.h 2007-07-28 01:12:49.000000000
+0530
@@ -29,6 +29,13 @@ extern void mm_init_container(struct mm_
extern void mm_free_container(struct mm_struct *mm);
extern void page_assign_meta_page(struct page *page, struct meta_page *mp);
extern struct meta_page *page_get_meta_page(struct page *page);
+extern int mem_container_charge(struct page *page, struct mm_struct *mm);
+extern void mem_container_uncharge(struct meta_page *mp);
+
+static inline void mem_container_uncharge_page(struct page *page)
+{
+ mem_container_uncharge(page_get_meta_page(page));
+}

#else /* CONFIG_CONTAINER_MEM_CONT */
static inline void mm_init_container(struct mm_struct *mm,
@@ -50,6 +57,19 @@ static inline struct meta_page *page_get
return NULL;
}
```

```

+static inline int mem_container_charge(struct page *page, struct mm_struct *mm)
+{
+ return 0;
+}
+
+static inline void mem_container_uncharge(struct meta_page *mp)
+{
+}
+
+static inline void mem_container_uncharge_page(struct page *page)
+{
+}
+
#endif /* CONFIG_CONTAINER_MEM_CONT */

#endif /* _LINUX_MEMCONTROL_H */
diff -puN include/linux/page-flags.h~mem-control-accounting include/linux/page-flags.h
--- linux-2.6.23-rc1-mm1/include/linux/page-flags.h~mem-control-accounting 2007-07-28
01:12:49.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/page-flags.h 2007-07-28 01:12:49.000000000
+0530
@@ -98,6 +98,9 @@
#define PG_checked PG_owner_priv_1 /* Used by some filesystems */
#define PG_pinned PG_owner_priv_1 /* Xen pinned pagetable */

+#define PG_metapage 21 /* Used for checking if a meta_page */
+ /* is associated with a page */
+
+#if (BITS_PER_LONG > 32)
/*
 * 64-bit-only flags build down from bit 31
diff -puN mm/filemap.c~mem-control-accounting mm/filemap.c
--- linux-2.6.23-rc1-mm1/mm/filemap.c~mem-control-accounting 2007-07-28 01:12:49.000000000
+0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/filemap.c 2007-07-28 01:12:49.000000000 +0530
@@ -31,6 +31,7 @@
#include <linux/syscalls.h>
#include <linux/cpuset.h>
#include <linux/hardirq.h> /* for BUG_ON(!in_atomic()) only */
+#include <linux/memcontrol.h>
#include "internal.h"

/*
@@ -116,6 +117,7 @@ void __remove_from_page_cache(struct pag
{
struct address_space *mapping = page->mapping;

```

```

+ mem_container_uncharge_page(page);
+ radix_tree_delete(&mapping->page_tree, page->index);
+ page->mapping = NULL;
+ mapping->numpages--;
@@ -442,6 +444,11 @@ int add_to_page_cache(struct page *page,
+ int error = radix_tree_preload(gfp_mask & ~__GFP_HIGHMEM);

+ if (error == 0) {
+
+ error = mem_container_charge(page, current->mm);
+ if (error)
+ goto out;
+
+ write_lock_irq(&mapping->tree_lock);
+ error = radix_tree_insert(&mapping->page_tree, offset, page);
+ if (!error) {
@@ -451,10 +458,13 @@ int add_to_page_cache(struct page *page,
+ page->index = offset;
+ mapping->numpages++;
+ __inc_zone_page_state(page, NR_FILE_PAGES);
- }
+ } else
+ mem_container_uncharge_page(page);
+
+ write_unlock_irq(&mapping->tree_lock);
+ radix_tree_preload_end();
+ }
+out:
+ return error;
+ }
EXPORT_SYMBOL(add_to_page_cache);
diff -puN mm/memcontrol.c~mem-control-accounting mm/memcontrol.c
--- linux-2.6.23-rc1-mm1/mm/memcontrol.c~mem-control-accounting 2007-07-28
01:12:49.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/memcontrol.c 2007-07-28 01:12:49.000000000 +0530
@@ -21,6 +21,9 @@
#include <linux/memcontrol.h>
#include <linux/container.h>
#include <linux/mm.h>
+#include <linux/page-flags.h>
+#include <linux/bit_spinlock.h>
+#include <linux/rcupdate.h>

struct container_subsys mem_container_subsys;

@@ -31,7 +34,9 @@ struct container_subsys mem_container_su
* to help the administrator determine what knobs to tune.
*

```

```

* TODO: Add a water mark for the memory controller. Reclaim will begin when
- * we hit the water mark.
+ * we hit the water mark. May be even add a low water mark, such that
+ * no reclaim occurs from a container at it's low water mark, this is
+ * a feature that will be implemented much later in the future.
*/

```

```

struct mem_container {
    struct container_subsys_state css;
@@ -56,6 +61,8 @@ struct meta_page {
    struct list_head lru; /* per container LRU list */
    struct page *page;
    struct mem_container *mem_container;
+ atomic_t ref_cnt; /* Helpful when pages move b/w */
+ /* mapped and cached states */
};

```

```

@@ -98,6 +105,134 @@ struct meta_page *page_get_meta_page(str
    return page->meta_page;
}

```

```

+void __always_inline lock_meta_page(struct page *page)
+{
+ bit_spin_lock(PG_metapage, &page->flags);
+ VM_BUG_ON(!test_bit(PG_metapage, &page->flags));
+}
+
+void __always_inline unlock_meta_page(struct page *page)
+{
+ bit_spin_unlock(PG_metapage, &page->flags);
+}
+
+/*
+ * Charge the memory controller for page usage.
+ * Return
+ * 0 if the charge was successful
+ * < 0 if the container is over its limit
+ */
+int mem_container_charge(struct page *page, struct mm_struct *mm)
+{
+ struct mem_container *mem;
+ struct meta_page *mp, *race_mp;
+
+ /*
+ * Should meta_page's go to their own slab?
+ * One could optimize the performance of the charging routine
+ * by saving a bit in the page_flags and using it as a lock
+ * to see if the container page already has a meta_page associated
+ */
+}

```

```

+ * with it
+ */
+ lock_meta_page(page);
+ mp = page_get_meta_page(page);
+ /*
+ * The meta_page exists and the page has already been accounted
+ */
+ if (mp) {
+ atomic_inc(&mp->ref_cnt);
+ goto done;
+ }
+
+ unlock_meta_page(page);
+
+ mp = kzalloc(sizeof(struct meta_page), GFP_KERNEL);
+ if (mp == NULL)
+ goto err;
+
+ rcu_read_lock();
+ /*
+ * We always charge the container the mm_struct belongs to
+ * the mm_struct's mem_container changes on task migration if the
+ * thread group leader migrates. It's possible that mm is not
+ * set, if so charge the init_mm (happens for pagecache usage).
+ */
+ if (!mm)
+ mm = &init_mm;
+
+ mem = rcu_dereference(mm->mem_container);
+ /*
+ * For every charge from the container, increment reference
+ * count
+ */
+ css_get(&mem->css);
+ rcu_read_unlock();
+
+ /*
+ * If we created the meta_page, we should free it on exceeding
+ * the container limit.
+ */
+ if (res_counter_charge(&mem->res, 1)) {
+ css_put(&mem->css);
+ goto free_mp;
+ }
+
+ lock_meta_page(page);
+ /*
+ * Check if somebody else beat us to allocating the meta_page

```

```

+ */
+ race_mp = page_get_meta_page(page);
+ if (race_mp) {
+   kfree(mp);
+   mp = race_mp;
+   atomic_inc(&mp->ref_cnt);
+   res_counter_uncharge(&mem->res, 1);
+   goto done;
+ }
+
+ atomic_set(&mp->ref_cnt, 1);
+ mp->mem_container = mem;
+ mp->page = page;
+ page_assign_meta_page(page, mp);
+
+done:
+ unlock_meta_page(page);
+ return 0;
+free_mp:
+ kfree(mp);
+ return -ENOMEM;
+err:
+ unlock_meta_page(page);
+ return -ENOMEM;
+}
+
+/*
+ * Uncharging is always a welcome operation, we never complain, simply
+ * uncharge.
+ */
+void mem_container_uncharge(struct meta_page *mp)
+{
+ struct mem_container *mem;
+ struct page *page;
+
+ /*
+  * This can happen for PAGE_ZERO
+  */
+ if (!mp)
+   return;
+
+ if (atomic_dec_and_test(&mp->ref_cnt)) {
+   page = mp->page;
+   lock_meta_page(page);
+   mem = mp->mem_container;
+   css_put(&mem->css);
+   page_assign_meta_page(page, NULL);
+   unlock_meta_page(page);

```

```

+ res_counter_uncharge(&mem->res, 1);
+ kfree(mp);
+ }
+}
+
static ssize_t mem_container_read(struct container *cont, struct cftype *cft,
    struct file *file, char __user *userbuf, size_t nbytes,
    loff_t *ppos)
@@ -150,6 +285,8 @@ mem_container_create(struct container_su
    return NULL;

    res_counter_init(&mem->res);
+ INIT_LIST_HEAD(&mem->active_list);
+ INIT_LIST_HEAD(&mem->inactive_list);
    return &mem->css;
}

diff -puN mm/memory.c~mem-control-accounting mm/memory.c
--- linux-2.6.23-rc1-mm1/mm/memory.c~mem-control-accounting 2007-07-28
01:12:49.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/memory.c 2007-07-28 01:12:49.000000000 +0530
@@ -50,6 +50,7 @@
#include <linux/delayacct.h>
#include <linux/init.h>
#include <linux/writeback.h>
+#include <linux/memcontrol.h>

#include <asm/pgalloc.h>
#include <asm/uaccess.h>
@@ -1225,14 +1226,18 @@ static int insert_page(struct mm_struct
    pte_t *pte;
    spinlock_t *ptl;

+ retval = mem_container_charge(page, mm);
+ if (retval)
+ goto out;
+
    retval = -EINVAL;
    if (PageAnon(page))
- goto out;
+ goto out_uncharge;
    retval = -ENOMEM;
    flush_dcache_page(page);
    pte = get_locked_pte(mm, addr, &ptl);
    if (!pte)
- goto out;
+ goto out_uncharge;
    retval = -EBUSY;

```

```

    if (!pte_none(*pte))
        goto out_unlock;
@@ -1244,8 +1249,11 @@ static int insert_page(struct mm_struct
    set_pte_at(mm, addr, pte, mk_pte(page, prot));

    retval = 0;
+ return retval;
    out_unlock:
        pte_unmap_unlock(pte, ptl);
+out_uncharge:
+ mem_container_uncharge_page(page);
    out:
        return retval;
}
@@ -1725,6 +1733,9 @@ gotten:
    cow_user_page(new_page, old_page, address, vma);
}

+ if (mem_container_charge(new_page, mm))
+ goto oom_free_new;
+
    /*
     * Re-check the pte - we dropped the lock
     */
@@ -1757,7 +1768,9 @@ gotten:
    /* Free the old page.. */
    new_page = old_page;
    ret |= VM_FAULT_WRITE;
- }
+ } else
+ mem_container_uncharge_page(new_page);
+
    if (new_page)
        page_cache_release(new_page);
    if (old_page)
@@ -1778,6 +1791,8 @@ unlock:
    put_page(dirty_page);
}
    return ret;
+oom_free_new:
+ __free_page(new_page);
    oom:
    if (old_page)
        page_cache_release(old_page);
@@ -2182,6 +2197,11 @@ static int do_swap_page(struct mm_struct
}

    delayacct_clear_flag(DELAYACCT_PF_SWAPIN);

```

```

+ if (mem_container_charge(page, mm)) {
+ ret = VM_FAULT_OOM;
+ goto out;
+ }
+
+ mark_page_accessed(page);
+ lock_page(page);

@@ -2218,8 +2238,10 @@ static int do_swap_page(struct mm_struct
+ if (write_access) {
+ /* XXX: We could OR the do_wp_page code with this one? */
+ if (do_wp_page(mm, vma, address,
- page_table, pmd, ptl, pte) & VM_FAULT_OOM)
+ page_table, pmd, ptl, pte) & VM_FAULT_OOM) {
+ mem_container_uncharge_page(page);
+ ret = VM_FAULT_OOM;
+ }
+ goto out;
+ }

@@ -2230,6 +2252,7 @@ unlock:
out:
return ret;
out_nomap:
+ mem_container_uncharge_page(page);
+ pte_unmap_unlock(page_table, ptl);
+ unlock_page(page);
+ page_cache_release(page);
@@ -2250,6 +2273,7 @@ static int do_anonymous_page(struct mm_s
+ pte_t entry;

+ if (write_access) {
+
+ /* Allocate our own private page. */
+ pte_unmap(page_table);

@@ -2259,6 +2283,9 @@ static int do_anonymous_page(struct mm_s
+ if (!page)
+ goto oom;

+ if (mem_container_charge(page, mm))
+ goto oom_free_page;
+
+ entry = mk_pte(page, vma->vm_page_prot);
+ entry = maybe_mkdirty(entry, vma);

@@ -2291,8 +2318,11 @@ unlock:
+ pte_unmap_unlock(page_table, ptl);

```

```

    return 0;
release:
+ mem_container_uncharge_page(page);
  page_cache_release(page);
  goto unlock;
+oom_free_page:
+ __free_page(page);
oom:
  return VM_FAULT_OOM;
}
@@ -2403,6 +2433,11 @@ static int __do_fault(struct mm_struct *

}

+ if (mem_container_charge(page, mm)) {
+   ret = VM_FAULT_OOM;
+   goto out;
+ }
+
  page_table = pte_offset_map_lock(mm, pmd, address, &ptl);

  /*
@@ -2439,6 +2474,7 @@ static int __do_fault(struct mm_struct *
    update_mmu_cache(vma, address, entry);
    lazy_mmu_prot_update(entry);
  } else {
+ mem_container_uncharge_page(page);
    if (anon)
      page_cache_release(page);
    else
diff -puN mm/migrate.c~mem-control-accounting mm/migrate.c
--- linux-2.6.23-rc1-mm1/mm/migrate.c~mem-control-accounting 2007-07-28 01:12:49.000000000
+0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/migrate.c 2007-07-28 01:12:49.000000000 +0530
@@ -28,6 +28,7 @@
#include <linux/mempolicy.h>
#include <linux/vmalloc.h>
#include <linux/security.h>
+#include <linux/memcontrol.h>

#include "internal.h"

@@ -156,6 +157,11 @@ static void remove_migration_pte(struct
  return;
}

+ if (mem_container_charge(new, mm)) {
+   pte_unmap(pte);

```

```

+ return;
+ }
+
    ptl = pte_lockptr(mm, pmd);
    spin_lock(ptl);
    pte = *ptep;
diff -puN mm/page_alloc.c~mem-control-accounting mm/page_alloc.c
--- linux-2.6.23-rc1-mm1/mm/page_alloc.c~mem-control-accounting 2007-07-28
01:12:49.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/page_alloc.c 2007-07-28 01:12:49.000000000 +0530
@@ -42,6 +42,7 @@
#include <linux/backing-dev.h>
#include <linux/fault-inject.h>
#include <linux/page-isolation.h>
+#include <linux/memcontrol.h>

#include <asm/tlbflush.h>
#include <asm/div64.h>
@@ -1017,6 +1018,7 @@ static void fastcall free_hot_cold_page(

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

@@ -2580,6 +2582,7 @@ void __meminit memmap_init_zone(unsigned
    set_page_links(page, zone, nid, pfn);
    init_page_count(page);
    reset_page_mapcount(page);
+ page_assign_meta_page(page, NULL);
    SetPageReserved(page);

/*
diff -puN mm/rmap.c~mem-control-accounting mm/rmap.c
--- linux-2.6.23-rc1-mm1/mm/rmap.c~mem-control-accounting 2007-07-28 01:12:49.000000000
+0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/rmap.c 2007-07-28 01:12:49.000000000 +0530
@@ -551,8 +551,14 @@ void page_add_anon_rmap(struct page *pag
    VM_BUG_ON(address < vma->vm_start || address >= vma->vm_end);
    if (atomic_inc_and_test(&page->_mapcount))
        __page_set_anon_rmap(page, vma, address);
- else
+ else {
    __page_check_anon_rmap(page, vma, address);
+ }
+ /*
+  * We unconditionally charged during prepare, we uncharge here
+  * This takes care of balancing the reference counts

```

```

+ */
+ mem_container_uncharge_page(page);
+ }
+ }

/*
@@ -583,6 +589,12 @@ void page_add_file_rmap(struct page *pag
{
    if (atomic_inc_and_test(&page->_mapcount))
        __inc_zone_page_state(page, NR_FILE_MAPPED);
+ else
+ /*
+  * We unconditionally charged during prepare, we uncharge here
+  * This takes care of balancing the reference counts
+  */
+ mem_container_uncharge_page(page);
+ }

#ifdef CONFIG_DEBUG_VM
@@ -643,6 +655,8 @@ void page_remove_rmap(struct page *page,
    page_clear_dirty(page);
    set_page_dirty(page);
}
+ mem_container_uncharge_page(page);
+
    __dec_zone_page_state(page,
        PageAnon(page) ? NR_ANON_PAGES : NR_FILE_MAPPED);
}
diff -puN mm/swapfile.c~mem-control-accounting mm/swapfile.c
--- linux-2.6.23-rc1-mm1/mm/swapfile.c~mem-control-accounting 2007-07-28
01:12:49.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/swapfile.c 2007-07-28 01:12:49.000000000 +0530
@@ -506,9 +506,12 @@ unsigned int count_swap_pages(int type,
    * just let do_wp_page work it out if a write is requested later - to
    * force COW, vm_page_prot omits write permission from any private vma.
    */
-static void unuse_pte(struct vm_area_struct *vma, pte_t *pte,
+static int unuse_pte(struct vm_area_struct *vma, pte_t *pte,
    unsigned long addr, swp_entry_t entry, struct page *page)
{
+ if (mem_container_charge(page, vma->vm_mm))
+ return -ENOMEM;
+
    inc_mm_counter(vma->vm_mm, anon_rss);
    get_page(page);
    set_pte_at(vma->vm_mm, addr, pte,
@@ -520,6 +523,7 @@ static void unuse_pte(struct vm_area_str
    * immediately swapped out again after swapon.

```

```

    */
    activate_page(page);
+ return 1;
}

static int unuse_pte_range(struct vm_area_struct *vma, pmd_t *pmd,
@@ -529,7 +533,7 @@ static int unuse_pte_range(struct vm_area
    pte_t swp_pte = swp_entry_to_pte(entry);
    pte_t *pte;
    spinlock_t *ptl;
- int found = 0;
+ int ret = 0;

    pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
    do {
@@ -538,13 +542,12 @@ static int unuse_pte_range(struct vm_area
    * Test inline before going to call unuse_pte.
    */
    if (unlikely(pte_same(*pte, swp_pte))) {
- unuse_pte(vma, pte++, addr, entry, page);
- found = 1;
+ ret = unuse_pte(vma, pte++, addr, entry, page);
    break;
    }
    } while (pte++, addr += PAGE_SIZE, addr != end);
    pte_unmap_unlock(pte - 1, ptl);
- return found;
+ return ret;
}

static inline int unuse_pmd_range(struct vm_area_struct *vma, pud_t *pud,
@@ -553,14 +556,16 @@ static inline int unuse_pmd_range(struct
{
    pmd_t *pmd;
    unsigned long next;
+ int ret;

    pmd = pmd_offset(pud, addr);
    do {
        next = pmd_addr_end(addr, end);
        if (pmd_none_or_clear_bad(pmd))
            continue;
- if (unuse_pte_range(vma, pmd, addr, next, entry, page))
- return 1;
+ ret = unuse_pte_range(vma, pmd, addr, next, entry, page);
+ if (ret)
+ return ret;
    } while (pmd++, addr = next, addr != end);

```

```

    return 0;
}
@@ -571,14 +576,16 @@ static inline int unuse_pud_range(struct
{
    pud_t *pud;
    unsigned long next;
+ int ret;

    pud = pud_offset(pgd, addr);
    do {
        next = pud_addr_end(addr, end);
        if (pud_none_or_clear_bad(pud))
            continue;
- if (unuse_pmd_range(vma, pud, addr, next, entry, page))
- return 1;
+ ret = unuse_pmd_range(vma, pud, addr, next, entry, page);
+ if (ret)
+ return ret;
    } while (pud++, addr = next, addr != end);
    return 0;
}
@@ -588,6 +595,7 @@ static int unuse_vma(struct vm_area_stru
{
    pgd_t *pgd;
    unsigned long addr, end, next;
+ int ret;

    if (page->mapping) {
        addr = page_address_in_vma(page, vma);
@@ -605,8 +613,9 @@ static int unuse_vma(struct vm_area_stru
    next = pgd_addr_end(addr, end);
    if (pgd_none_or_clear_bad(pgd))
        continue;
- if (unuse_pud_range(vma, pgd, addr, next, entry, page))
- return 1;
+ ret = unuse_pud_range(vma, pgd, addr, next, entry, page);
+ if (ret)
+ return ret;
    } while (pgd++, addr = next, addr != end);
    return 0;
}
@@ -615,6 +624,7 @@ static int unuse_mm(struct mm_struct *mm
    swp_entry_t entry, struct page *page)
{
    struct vm_area_struct *vma;
+ int ret = 0;

    if (!down_read_trylock(&mm->mmap_sem)) {

```

```

/*
@@ -627,15 +637,11 @@ static int unuse_mm(struct mm_struct *mm
    lock_page(page);
}
for (vma = mm->mmap; vma; vma = vma->vm_next) {
- if (vma->anon_vma && unuse_vma(vma, entry, page))
+ if (vma->anon_vma && (ret = unuse_vma(vma, entry, page)))
    break;
}
up_read(&mm->mmap_sem);
- /*
- * Currently unuse_mm cannot fail, but leave error handling
- * at call sites for now, since we change it from time to time.
- */
- return 0;
+ return ret;
}

/*
diff -puN mm/swap_state.c~mem-control-accounting mm/swap_state.c
--- linux-2.6.23-rc1-mm1/mm/swap_state.c~mem-control-accounting 2007-07-28
01:12:49.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/swap_state.c 2007-07-28 01:12:49.000000000 +0530
@@ -17,6 +17,7 @@
#include <linux/backing-dev.h>
#include <linux/pagevec.h>
#include <linux/migrate.h>
+#include <linux/memcontrol.h>

#include <asm/pgtable.h>

@@ -79,6 +80,11 @@ static int __add_to_swap_cache(struct pa
    BUG_ON(PagePrivate(page));
    error = radix_tree_preload(gfp_mask);
    if (!error) {
+
+ error = mem_container_charge(page, current->mm);
+ if (error)
+ goto out;
+
    write_lock_irq(&swapper_space.tree_lock);
    error = radix_tree_insert(&swapper_space.page_tree,
        entry.val, page);
@@ -89,10 +95,13 @@ static int __add_to_swap_cache(struct pa
    set_page_private(page, entry.val);
    total_swapcache_pages++;
    __inc_zone_page_state(page, NR_FILE_PAGES);
- }

```

```

+ } else
+ mem_container_uncharge_page(page);
+
+ write_unlock_irq(&swapper_space.tree_lock);
+ radix_tree_preload_end();
+ }
+out:
+ return error;
+ }

@@ -129,6 +138,7 @@ void __delete_from_swap_cache(struct pag
BUG_ON(PageWriteback(page));
BUG_ON(PagePrivate(page));

+ mem_container_uncharge_page(page);
+ radix_tree_delete(&swapper_space.page_tree, page_private(page));
+ set_page_private(page, 0);
+ ClearPageSwapCache(page);

```

Warm Regards,
 Balbir Singh
 Linux Technology Center
 IBM, ISTL

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

Subject: [-mm PATCH 5/9] Memory controller task migration (v4)
 Posted by [Balbir Singh](#) on Fri, 27 Jul 2007 20:10:32 GMT
[View Forum Message](#) <> [Reply to Message](#)

Allow tasks to migrate from one container to the other. We migrate
 mm_struct's mem_container only when the thread group id migrates.

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

mm/memcontrol.c | 35 ++++++
 1 file changed, 35 insertions(+)

diff -puN mm/memcontrol.c~mem-control-task-migration mm/memcontrol.c
 --- linux-2.6.23-rc1-mm1/mm/memcontrol.c~mem-control-task-migration 2007-07-28
 01:12:50.000000000 +0530

```

+++ linux-2.6.23-rc1-mm1-balbir/mm/memcontrol.c 2007-07-28 01:12:50.000000000 +0530
@@ -303,11 +303,46 @@ static int mem_container_populate(struct
    ARRAY_SIZE(mem_container_files));
}

+static void mem_container_move_task(struct container_subsys *ss,
+ struct container *cont,
+ struct container *old_cont,
+ struct task_struct *p)
+{
+ struct mm_struct *mm;
+ struct mem_container *mem, *old_mem;
+
+ mm = get_task_mm(p);
+ if (mm == NULL)
+ return;
+
+ mem = mem_container_from_cont(cont);
+ old_mem = mem_container_from_cont(old_cont);
+
+ if (mem == old_mem)
+ goto out;
+
+ /*
+ * Only thread group leaders are allowed to migrate, the mm_struct is
+ * in effect owned by the leader
+ */
+ if (p->tgid != p->pid)
+ goto out;
+
+ css_get(&mem->css);
+ rcu_assign_pointer(mm->mem_container, mem);
+ css_put(&old_mem->css);
+
+out:
+ mmput(mm);
+ return;
+}
+
+struct container_subsys mem_container_subsys = {
+ .name = "memory",
+ .subsys_id = mem_container_subsys_id,
+ .create = mem_container_create,
+ .destroy = mem_container_destroy,
+ .populate = mem_container_populate,
+ .attach = mem_container_move_task,
+ .early_init = 1,
+};

```

—

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

Subject: [-mm PATCH 6/9] Memory controller add per container LRU and reclaim (v4)

Posted by [Balbir Singh](#) on Fri, 27 Jul 2007 20:10:41 GMT

[View Forum Message](#) <> [Reply to Message](#)

Changelog since v3

1. Added reclaim retry logic to avoid being OOM'ed due to pages from swap cache (coming in due to reclaim) don't overwhelm the container.

Changelog

1. Fix probable NULL pointer dereference based on review comments by YAMAMOTO Takashi

Add the meta_page to the per container LRU. The reclaim algorithm has been modified to make the isolate_lru_pages() as a pluggable component. The scan_control data structure now accepts the container on behalf of which reclaims are carried out. try_to_free_pages() has been extended to become container aware.

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

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

```
include/linux/memcontrol.h | 11 +++
include/linux/res_counter.h | 23 +++++++
include/linux/swap.h       | 3
mm/memcontrol.c            | 139 ++++++++++++++++++++++++++++++++++++++
mm/swap.c                  | 2
mm/vmscan.c                | 126 +++++++++++++++++++++++++++++++++-----
6 files changed, 278 insertions(+), 26 deletions(-)
```

```
diff -puN include/linux/memcontrol.h~mem-control-lru-and-reclaim include/linux/memcontrol.h
--- linux-2.6.23-rc1-mm1/include/linux/memcontrol.h~mem-control-lru-and-reclaim 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/memcontrol.h 2007-07-28 01:12:50.000000000
```

```

+0530
@@ -31,6 +31,13 @@ extern void page_assign_meta_page(struct
extern struct meta_page *page_get_meta_page(struct page *page);
extern int mem_container_charge(struct page *page, struct mm_struct *mm);
extern void mem_container_uncharge(struct meta_page *mp);
+extern void mem_container_move_lists(struct meta_page *mp, bool active);
+extern unsigned long mem_container_isolate_pages(unsigned long nr_to_scan,
+ struct list_head *dst,
+ unsigned long *scanned, int order,
+ int mode, struct zone *z,
+ struct mem_container *mem_cont,
+ int active);

static inline void mem_container_uncharge_page(struct page *page)
{
@@ -70,6 +77,10 @@ static inline void mem_container_uncharg
{
}

+static inline void mem_container_move_lists(struct meta_page *mp, bool active)
+{
+}
+
#endif /* CONFIG_CONTAINER_MEM_CONT */

#endif /* _LINUX_MEMCONTROL_H */
diff -puN include/linux/res_counter.h~mem-control-lru-and-reclaim include/linux/res_counter.h
--- linux-2.6.23-rc1-mm1/include/linux/res_counter.h~mem-control-lru-and-reclaim 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/res_counter.h 2007-07-28 01:12:50.000000000
+0530
@@ -99,4 +99,27 @@ int res_counter_charge(struct res_counte
void res_counter_uncharge_locked(struct res_counter *counter, unsigned long val);
void res_counter_uncharge(struct res_counter *counter, unsigned long val);

+static inline bool res_counter_limit_check_locked(struct res_counter *cnt)
+{
+ if (cnt->usage < cnt->limit)
+ return true;
+
+ return false;
+}
+
+/*
+ * Helper function to detect if the container is within it's limit or
+ * not. It's currently called from container_rss_prepare()
+ */
+static inline bool res_counter_check_under_limit(struct res_counter *cnt)

```

```

+{
+ bool ret;
+ unsigned long flags;
+
+ spin_lock_irqsave(&cnt->lock, flags);
+ ret = res_counter_limit_check_locked(cnt);
+ spin_unlock_irqrestore(&cnt->lock, flags);
+ return ret;
+}
+
#endif
diff -puN include/linux/swap.h~mem-control-lru-and-reclaim include/linux/swap.h
--- linux-2.6.23-rc1-mm1/include/linux/swap.h~mem-control-lru-and-reclaim 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/swap.h 2007-07-28 01:12:50.000000000 +0530
@@ -6,6 +6,7 @@
#include <linux/mmzone.h>
#include <linux/list.h>
#include <linux/sched.h>
+#include <linux/memcontrol.h>

#include <asm/atomic.h>
#include <asm/page.h>
@@ -191,6 +192,8 @@ 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);
+extern unsigned long try_to_free_mem_container_pages(struct mem_container *mem);
+extern 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 -puN mm/memcontrol.c~mem-control-lru-and-reclaim mm/memcontrol.c
--- linux-2.6.23-rc1-mm1/mm/memcontrol.c~mem-control-lru-and-reclaim 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/memcontrol.c 2007-07-28 01:12:50.000000000 +0530
@@ -24,8 +24,11 @@
#include <linux/page-flags.h>
#include <linux/bit_spinlock.h>
#include <linux/rcupdate.h>
+#include <linux/swap.h>
+#include <linux/spinlock.h>

struct container_subsys mem_container_subsys;
+static const int MEM_CONTAINER_RECLAIM_RETRIES = 5;

/*
 * The memory controller data structure. The memory controller controls both

```

```

@@ -51,6 +54,10 @@ struct mem_container {
    */
    struct list_head active_list;
    struct list_head inactive_list;
+ /*
+  * spin_lock to protect the per container LRU
+  */
+ spinlock_t lru_lock;
};

/*
@@ -116,6 +123,97 @@ void __always_inline unlock_meta_page(st
    bit_spin_unlock(PG_metapage, &page->flags);
}

+unsigned long mem_container_isolate_pages(unsigned long nr_to_scan,
+    struct list_head *dst,
+    unsigned long *scanned, int order,
+    int mode, struct zone *z,
+    struct mem_container *mem_cont,
+    int active)
+{
+    unsigned long nr_taken = 0;
+    struct page *page;
+    unsigned long scan;
+    LIST_HEAD(mp_list);
+    struct list_head *src;
+    struct meta_page *mp;
+
+    if (active)
+        src = &mem_cont->active_list;
+    else
+        src = &mem_cont->inactive_list;
+
+    for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
+        mp = list_entry(src->prev, struct meta_page, lru);
+        spin_lock(&mem_cont->lru_lock);
+        if (mp)
+            page = mp->page;
+        spin_unlock(&mem_cont->lru_lock);
+        if (!mp)
+            continue;
+
+        if (PageActive(page) && !active) {
+            mem_container_move_lists(mp, true);
+            scan--;
+            continue;
+        }
+    }

```

```

+ if (!PageActive(page) && active) {
+   mem_container_move_lists(mp, false);
+   scan--;
+   continue;
+ }
+
+ /*
+  * Reclaim, per zone
+  * TODO: make the active/inactive lists per zone
+  */
+ if (page_zone(page) != z)
+   continue;
+
+ spin_lock(&mem_cont->lru_lock);
+ /*
+  * Check if the meta page went away from under us
+  */
+ if (!list_empty(&mp->lru))
+   list_move(&mp->lru, &mp_list);
+ else {
+   spin_unlock(&mem_cont->lru_lock);
+   continue;
+ }
+ spin_unlock(&mem_cont->lru_lock);
+
+ if (__isolate_lru_page(page, mode) == 0) {
+   list_move(&page->lru, dst);
+   nr_taken++;
+ }
+ }
+
+ spin_lock(&mem_cont->lru_lock);
+ list_splice(&mp_list, src);
+ spin_unlock(&mem_cont->lru_lock);
+
+ *scanned = scan;
+ return nr_taken;
+}
+
+/*
+ * This routine assumes that the appropriate zone's lru lock is already held
+ */
+void mem_container_move_lists(struct meta_page *mp, bool active)
+{
+ struct mem_container *mem;
+ if (!mp)
+   return;
+
+

```

```

+ mem = mp->mem_container;
+
+ spin_lock(&mem->lru_lock);
+ if (active)
+ list_move(&mp->lru, &mp->mem_container->active_list);
+ else
+ list_move(&mp->lru, &mp->mem_container->inactive_list);
+ spin_unlock(&mem->lru_lock);
+}
+
/*
 * Charge the memory controller for page usage.
 * Return
@@ -126,6 +224,8 @@ int mem_container_charge(struct page *pa
{
    struct mem_container *mem;
    struct meta_page *mp, *race_mp;
+ unsigned long flags;
+ unsigned long nr_retries = MEM_CONTAINER_RECLAIM_RETRIES;

/*
 * Should meta_page's go to their own slab?
@@ -172,7 +272,32 @@ int mem_container_charge(struct page *pa
 * If we created the meta_page, we should free it on exceeding
 * the container limit.
 */
- if (res_counter_charge(&mem->res, 1)) {
+ while (res_counter_charge(&mem->res, 1)) {
+ if (try_to_free_mem_container_pages(mem))
+ continue;
+
+ /*
+  * try_to_free_mem_container_pages() might not give us a full
+  * picture of reclaim. Some pages are reclaimed and might be
+  * moved to swap cache or just unmapped from the container.
+  * Check the limit again to see if the reclaim reduced the
+  * current usage of the container before giving up
+  */
+ if (res_counter_check_under_limit(&mem->res))
+ continue;
+
+ /*
+  * Since we control both RSS and cache, we end up with a
+  * very interesting scenario where we end up reclaiming
+  * memory (essentially RSS), since the memory is pushed
+  * to swap cache, we eventually end up adding those
+  * pages back to our list. Hence we give ourselves a
+  * few chances before we fail
+  */

```

```

+ else if (nr_retries--) {
+   congestion_wait(WRITE, HZ/10);
+   continue;
+ }
+
+   css_put(&mem->css);
+   goto free_mp;
+ }
@@ -195,6 +320,10 @@ int mem_container_charge(struct page *pa
+   mp->page = page;
+   page_assign_meta_page(page, mp);

+ spin_lock_irqsave(&mem->lru_lock, flags);
+ list_add(&mp->lru, &mem->active_list);
+ spin_unlock_irqrestore(&mem->lru_lock, flags);
+
done:
+   unlock_meta_page(page);
+   return 0;
@@ -214,6 +343,7 @@ void mem_container_uncharge(struct meta_
+ {
+   struct mem_container *mem;
+   struct page *page;
+ unsigned long flags;

+   /*
+    * This can happen for PAGE_ZERO
@@ -223,12 +353,18 @@ void mem_container_uncharge(struct meta_

+   if (atomic_dec_and_test(&mp->ref_cnt)) {
+     page = mp->page;
+   }
+   lock_meta_page(page);
+   mem = mp->mem_container;
+   css_put(&mem->css);
+   page_assign_meta_page(page, NULL);
+   unlock_meta_page(page);
+
+   res_counter_uncharge(&mem->res, 1);
+
+   spin_lock_irqsave(&mem->lru_lock, flags);
+   list_del_init(&mp->lru);
+   spin_unlock_irqrestore(&mem->lru_lock, flags);
+   kfree(mp);
+ }
+ }
@@ -287,6 +423,7 @@ mem_container_create(struct container_su
+   res_counter_init(&mem->res);

```

```

INIT_LIST_HEAD(&mem->active_list);
INIT_LIST_HEAD(&mem->inactive_list);
+ spin_lock_init(&mem->lru_lock);
return &mem->css;
}

```

```

diff -puN mm/swap.c~mem-control-lru-and-reclaim mm/swap.c
--- linux-2.6.23-rc1-mm1/mm/swap.c~mem-control-lru-and-reclaim 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/swap.c 2007-07-28 01:12:50.000000000 +0530
@@ -29,6 +29,7 @@
#include <linux/percpu.h>
#include <linux/cpu.h>
#include <linux/notifier.h>
+#include <linux/memcontrol.h>

```

```

/* How many pages do we try to swap or page in/out together? */
int page_cluster;
@@ -146,6 +147,7 @@ void fastcall activate_page(struct page
SetPageActive(page);
add_page_to_active_list(zone, page);
__count_vm_event(PGACTIVATE);
+ mem_container_move_lists(page_get_meta_page(page), true);
}
spin_unlock_irq(&zone->lru_lock);
}

```

```

diff -puN mm/vmscan.c~mem-control-lru-and-reclaim mm/vmscan.c
--- linux-2.6.23-rc1-mm1/mm/vmscan.c~mem-control-lru-and-reclaim 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/vmscan.c 2007-07-28 01:12:50.000000000 +0530
@@ -39,6 +39,7 @@
#include <linux/delay.h>
#include <linux/kthread.h>
#include <linux/freezer.h>
+#include <linux/memcontrol.h>

```

```

#include <asm/tlbflush.h>
#include <asm/div64.h>
@@ -70,6 +71,15 @@ struct scan_control {
int all_unreclaimable;

int order;
+
+ /* Which container do we reclaim from */
+ struct mem_container *mem_container;
+
+ /* Pluggable isolate pages callback */
+ unsigned long (*isolate_pages)(unsigned long nr, struct list_head *dst,

```

```

+ unsigned long *scanned, int order, int mode,
+ struct zone *z, struct mem_container *mem_cont,
+ int active);
};

#define lru_to_page(_head) (list_entry((_head)->prev, struct page, lru))
@@ -604,7 +614,7 @@ 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;

@@ -738,6 +748,21 @@ static unsigned long isolate_lru_pages(u
    return nr_taken;
}

+static unsigned long isolate_pages_global(unsigned long nr,
+ struct list_head *dst,
+ unsigned long *scanned, int order,
+ int mode, struct zone *z,
+ struct mem_container *mem_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);
+}
+
+/*
+ * clear_active_flags() is a helper for shrink_active_list(), clearing
+ * any active bits from the pages in the list.
+@@ -779,11 +804,11 @@ static unsigned long shrink_inactive_lis
+ unsigned long nr_freed;
+ unsigned long nr_active;

- nr_taken = isolate_lru_pages(sc->swap_cluster_max,
- &z->inactive_list,
+ 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);
+ ISOLATE_BOTH : ISOLATE_INACTIVE,

```

```

+ zone, sc->mem_container, 0);
nr_active = clear_active_flags(&page_list);

__mod_zone_page_state(zone, NR_ACTIVE, -nr_active);
@@ -932,8 +957,9 @@ 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->mem_container, 1);
zone->pages_scanned += pgscanned;
__mod_zone_page_state(zone, NR_ACTIVE, -pgmoved);
spin_unlock_irq(&zone->lru_lock);
@@ -968,6 +994,7 @@ force_reclaim_mapped:
ClearPageActive(page);

list_move(&page->lru, &zone->inactive_list);
+ mem_container_move_lists(page_get_meta_page(page), false);
pgmoved++;
if (!pagevec_add(&pvec, page)) {
__mod_zone_page_state(zone, NR_INACTIVE, pgmoved);
@@ -996,6 +1023,7 @@ force_reclaim_mapped:
SetPageLRU(page);
VM_BUG_ON(!PageActive(page));
list_move(&page->lru, &zone->active_list);
+ mem_container_move_lists(page_get_meta_page(page), true);
pgmoved++;
if (!pagevec_add(&pvec, page)) {
__mod_zone_page_state(zone, NR_ACTIVE, pgmoved);
@@ -1127,7 +1155,8 @@ 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;
@@ -1136,14 +1165,6 @@ 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);
@@ -1159,17 +1180,22 @@ 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);
+ /*
+  * Don't shrink slabs when reclaiming memory from
+  * over limit containers
+  */
+ if (sc->mem_container == NULL)
+ shrink_slab(sc->nr_scanned, gfp_mask, lru_pages);
  if (reclaim_state) {
    nr_reclaimed += reclaim_state->reclaimed_slab;
    reclaim_state->reclaimed_slab = 0;
  }
- total_scanned += sc.nr_scanned;
- if (nr_reclaimed >= sc.swap_cluster_max) {
+ total_scanned += sc->nr_scanned;
+ if (nr_reclaimed >= sc->swap_cluster_max) {
  ret = 1;
  goto out;
}
@@ -1181,18 +1207,18 @@ unsigned long try_to_free_pages(struct z
  * that's undesirable in laptop mode, where we *want* lumpy
  * writeout. So in laptop mode, write out the whole world.
  */
- if (total_scanned > sc.swap_cluster_max +
-     sc.swap_cluster_max / 2) {
+ if (total_scanned > sc->swap_cluster_max +
+     sc->swap_cluster_max / 2) {
  wakeup_pdflush(laptop_mode ? 0 : total_scanned);
- sc.may_writepage = 1;
+ sc->may_writepage = 1;

```


Out of memory handling for containers over their limit. A task from the container over limit is chosen using the existing OOM logic and killed.

TODO:

1. As discussed in the OLS BOF session, consider implementing a user space policy for OOM handling.

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

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

```
include/linux/memcontrol.h | 1 +
mm/memcontrol.c            | 1 +
mm/oom_kill.c              | 42 ++++++-----
3 files changed, 40 insertions(+), 4 deletions(-)
```

```
diff -puN include/linux/memcontrol.h~mem-control-out-of-memory include/linux/memcontrol.h
--- linux-2.6.23-rc1-mm1/include/linux/memcontrol.h~mem-control-out-of-memory 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/memcontrol.h 2007-07-28 01:12:50.000000000
+0530
```

```
@@ -38,6 +38,7 @@ extern unsigned long mem_container_isola
```

```
    int mode, struct zone *z,
    struct mem_container *mem_cont,
    int active);
```

```
+extern void mem_container_out_of_memory(struct mem_container *mem);
```

```
static inline void mem_container_uncharge_page(struct page *page)
{
```

```
diff -puN mm/memcontrol.c~mem-control-out-of-memory mm/memcontrol.c
--- linux-2.6.23-rc1-mm1/mm/memcontrol.c~mem-control-out-of-memory 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/memcontrol.c 2007-07-28 01:12:50.000000000 +0530
```

```
@@ -299,6 +299,7 @@ int mem_container_charge(struct page *pa
}
```

```
    css_put(&mem->css);
+ mem_container_out_of_memory(mem);
    goto free_mp;
}
```

```
diff -puN mm/oom_kill.c~mem-control-out-of-memory mm/oom_kill.c
--- linux-2.6.23-rc1-mm1/mm/oom_kill.c~mem-control-out-of-memory 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/oom_kill.c 2007-07-28 01:12:50.000000000 +0530
@@ -24,6 +24,7 @@
#include <linux/cpuset.h>
```

```

#include <linux/module.h>
#include <linux/notifier.h>
#include <linux/memcontrol.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 mem_container *mem)
{
    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_CONTAINER_MEM_CONT
+ if (mem != NULL && mm->mem_container != mem) {
+ 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 mem_container *mem)
{
    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, mem);
    if (points > *ppoints || !chosen) {
        chosen = p;

```

```

    *ppoints = points;
@@ -370,6 +380,30 @@ static int oom_kill_process(struct task_
    return oom_kill_task(p);
}

#ifdef CONFIG_CONTAINER_MEM_CONT
+void mem_container_out_of_memory(struct mem_container *mem)
+{
+ unsigned long points = 0;
+ struct task_struct *p;
+
+ container_lock();
+ rcu_read_lock();
+retry:
+ p = select_bad_process(&points, mem);
+ if (PTR_ERR(p) == -1UL)
+ goto out;
+
+ if (!p)
+ p = current;
+
+ if (oom_kill_process(p, points, "Memory container out of memory"))
+ goto retry;
+out:
+ rcu_read_unlock();
+ container_unlock();
+}
+endif
+
static BLOCKING_NOTIFIER_HEAD(oom_notify_list);

int register_oom_notifier(struct notifier_block *nb)
@@ -442,7 +476,7 @@ 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;

```

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

Subject: [-mm PATCH 8/9] Memory controller add switch to control what type of pages to limit (v4)

Posted by [Balbir Singh](#) on Fri, 27 Jul 2007 20:11:03 GMT

[View Forum Message](#) <> [Reply to Message](#)

Choose if we want cached pages to be accounted or not. By default both are accounted for. A new set of tunables are added.

```
echo -n 1 > mem_control_type
```

switches the accounting to account for only mapped pages

```
echo -n 2 > mem_control_type
```

switches the behaviour back

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

```
include/linux/memcontrol.h | 9 ++++
mm/filemap.c               | 2
mm/memcontrol.c            | 92 ++++++++++++++++++++++++++++++++++++++
mm/swap_state.c           | 2
4 files changed, 102 insertions(+), 3 deletions(-)
```

```
diff -puN include/linux/memcontrol.h~mem-control-choose-rss-vs-rss-and-pagecache
include/linux/memcontrol.h
```

```
linux-2.6.23-rc1-mm1/include/linux/memcontrol.h~mem-control-choose-rss-vs-rss-and-pagecache
```

```
2007-07-28 01:12:50.000000000 +0530
```

```
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/memcontrol.h 2007-07-28 01:12:50.000000000
+0530
```

```
@@ -20,6 +20,8 @@
```

```
#ifndef _LINUX_MEMCONTROL_H
#define _LINUX_MEMCONTROL_H
```

```
+#include <linux/mm.h>
```

```
+
```

```
struct mem_container;
struct meta_page;
```

```
@@ -39,6 +41,7 @@ extern unsigned long mem_container_isola
```

```

    struct mem_container *mem_cont,
    int active);
extern void mem_container_out_of_memory(struct mem_container *mem);
+extern int mem_container_cache_charge(struct page *page, struct mm_struct *mm);

```

```

static inline void mem_container_uncharge_page(struct page *page)
{
@@ -82,6 +85,12 @@ static inline void mem_container_move_li
{
}

```

```

+static inline int mem_container_cache_charge(struct page *page,
+ struct mm_struct *mm)
+{
+ return 0;
+}
+
#endif /* CONFIG_CONTAINER_MEM_CONT */

```

```

#endif /* _LINUX_MEMCONTROL_H */

```

```

diff -puN mm/filemap.c~mem-control-choose-rss-vs-rss-and-pagecache mm/filemap.c

```

```

---
linux-2.6.23-rc1-mm1/mm/filemap.c~mem-control-choose-rss-vs-rss-and-pagecache 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/filemap.c 2007-07-28 01:12:50.000000000 +0530
@@ -445,7 +445,7 @@ int add_to_page_cache(struct page *page,

```

```

    if (error == 0) {

- error = mem_container_charge(page, current->mm);
+ error = mem_container_cache_charge(page, current->mm);
    if (error)
        goto out;

```

```

diff -puN mm/memcontrol.c~mem-control-choose-rss-vs-rss-and-pagecache mm/memcontrol.c

```

```

---
linux-2.6.23-rc1-mm1/mm/memcontrol.c~mem-control-choose-rss-vs-rss-and-pagecache 2007-07-
28 01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/memcontrol.c 2007-07-28 01:12:50.000000000 +0530
@@ -27,6 +27,8 @@
#include <linux/swap.h>
#include <linux/spinlock.h>

+#include <asm/uaccess.h>
+
struct container_subsys mem_container_subsys;
static const int MEM_CONTAINER_RECLAIM_RETRIES = 5;

```

```

@@ -58,6 +60,7 @@ struct mem_container {
    * spin_lock to protect the per container LRU
    */
    spinlock_t lru_lock;
+ unsigned long control_type; /* control RSS or RSS+Pagecache */
};

/*
@@ -72,6 +75,15 @@ struct meta_page {
    /* mapped and cached states */
};

+enum {
+ MEM_CONTAINER_TYPE_UNSPEC = 0,
+ MEM_CONTAINER_TYPE_MAPPED,
+ MEM_CONTAINER_TYPE_CACHED,
+ MEM_CONTAINER_TYPE_ALL,
+ MEM_CONTAINER_TYPE_MAX,
+} mem_control_type;
+
+static struct mem_container init_mem_container;

static inline
struct mem_container *mem_container_from_cont(struct container *cont)
@@ -337,6 +349,22 @@ err:
}

/*
+ * See if the cached pages should be charged at all?
+ */
+int mem_container_cache_charge(struct page *page, struct mm_struct *mm)
+{
+ struct mem_container *mem;
+ if (!mm)
+ mm = &init_mm;
+
+ mem = rcu_dereference(mm->mem_container);
+ if (mem->control_type == MEM_CONTAINER_TYPE_ALL)
+ return mem_container_charge(page, mm);
+ else
+ return 0;
+}
+
+/*
+ * Uncharging is always a welcome operation, we never complain, simply
+ * uncharge.
+ */
@@ -347,7 +375,9 @@ void mem_container_uncharge(struct meta_

```

```

unsigned long flags;

/*
- * This can happen for PAGE_ZERO
+ * This can happen for PAGE_ZERO. This can also handle cases when
+ * a page is not charged at all and we are switching between
+ * handling the control_type.
*/
if (!mp)
    return;
@@ -386,6 +416,60 @@ static ssize_t mem_container_write(struct
    cft->private, userbuf, nbytes, ppos);
}

+static ssize_t mem_control_type_write(struct container *cont,
+ struct cftype *cft, struct file *file,
+ const char __user *userbuf,
+ size_t nbytes, loff_t *pos)
+{
+ int ret;
+ char *buf, *end;
+ unsigned long tmp;
+ struct mem_container *mem;
+
+ mem = mem_container_from_cont(cont);
+ 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;
+
+ if (tmp <= MEM_CONTAINER_TYPE_UNSPEC || tmp >= MEM_CONTAINER_TYPE_MAX)
+ goto out_free;
+
+ mem->control_type = tmp;
+ ret = nbytes;
+out_free:
+ kfree(buf);
+out:

```

```

+ return ret;
+}
+
+static ssize_t mem_control_type_read(struct container *cont,
+ struct cftype *cft,
+ struct file *file, char __user *userbuf,
+ size_t nbytes, loff_t *ppos)
+{
+ unsigned long val;
+ char buf[64], *s;
+ struct mem_container *mem;
+
+ mem = mem_container_from_cont(cont);
+ s = buf;
+ val = mem->control_type;
+ s += sprintf(s, "%lu\n", val);
+ return simple_read_from_buffer((void __user *)userbuf, nbytes,
+ ppos, buf, s - buf);
+}
+
+static struct cftype mem_container_files[] = {
+ {
+ .name = "usage",
@@ -403,6 +487,11 @@ static struct cftype mem_container_files
+ .private = RES_FAILCNT,
+ .read = mem_container_read,
+ },
+ {
+ .name = "control_type",
+ .write = mem_control_type_write,
+ .read = mem_control_type_read,
+ },
+ };

static struct mem_container init_mem_container;
@@ -425,6 +514,7 @@ mem_container_create(struct container_su
INIT_LIST_HEAD(&mem->active_list);
INIT_LIST_HEAD(&mem->inactive_list);
spin_lock_init(&mem->lru_lock);
+ mem->control_type = MEM_CONTAINER_TYPE_ALL;
return &mem->css;
}

```

diff -puN mm/swap_state.c~mem-control-choose-rss-vs-rss-and-pagecache mm/swap_state.c

linux-2.6.23-rc1-mm1/mm/swap_state.c~mem-control-choose-rss-vs-rss-and-pagecache 2007-07-28 01:12:50.000000000 +0530

+++ linux-2.6.23-rc1-mm1-balbir/mm/swap_state.c 2007-07-28 01:12:50.000000000 +0530

```
@@ -81,7 +81,7 @@ static int __add_to_swap_cache(struct pa
    error = radix_tree_preload(gfp_mask);
    if (!error) {

-   error = mem_container_charge(page, current->mm);
+   error = mem_container_cache_charge(page, current->mm);
    if (error)
        goto out;
```

—

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

Subject: [-mm PATCH 9/9] Memory controller make page_referenced() container aware (v4)
Posted by [Balbir Singh](#) on Fri, 27 Jul 2007 20:11:19 GMT
[View Forum Message](#) <> [Reply to Message](#)

Make page_referenced() container aware. Without this patch, page_referenced() can cause a page to be skipped while reclaiming pages. This patch ensures that other containers do not hold pages in a particular container hostage. It is required to ensure that shared pages are freed from a container when they are not actively referenced from the container that brought them in

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

```
include/linux/memcontrol.h | 6 ++++++
include/linux/rmap.h       | 5 +++--
mm/memcontrol.c           | 5 ++++++
mm/rmap.c                 | 30 ++++++-----
mm/vmscan.c               | 4 +++--
5 files changed, 40 insertions(+), 10 deletions(-)
```

```
diff -puN include/linux/rmap.h~mem-control-per-container-page-referenced include/linux/rmap.h
---
linux-2.6.23-rc1-mm1/include/linux/rmap.h~mem-control-per-container-page-referenced 2007-07-
```

```

28 01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/rmap.h 2007-07-28 01:12:50.000000000 +0530
@@ -8,6 +8,7 @@
#include <linux/slab.h>
#include <linux/mm.h>
#include <linux/spinlock.h>
+#include <linux/memcontrol.h>

/*
 * The anon_vma heads a list of private "related" vmas, to scan if
@@ -86,7 +87,7 @@ static inline void page_dup_rmap(struct
/*
 * Called from mm/vmscan.c to handle paging out
 */
-int page_referenced(struct page *, int is_locked);
+int page_referenced(struct page *, int is_locked, struct mem_container *cnt);
int try_to_unmap(struct page *, int ignore_refs);

/*
@@ -114,7 +115,7 @@ int page_mkclean(struct page *);
#define anon_vma_prepare(vma) (0)
#define anon_vma_link(vma) do {} while (0)

-#define page_referenced(page,l) TestClearPageReferenced(page)
+#define page_referenced(page,l,cnt) TestClearPageReferenced(page)
#define try_to_unmap(page, refs) SWAP_FAIL

static inline int page_mkclean(struct page *page)
diff -puN mm/rmap.c~mem-control-per-container-page-referenced mm/rmap.c
--- linux-2.6.23-rc1-mm1/mm/rmap.c~mem-control-per-container-page-referenced 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/rmap.c 2007-07-28 01:12:50.000000000 +0530
@@ -298,7 +298,8 @@ out:
return referenced;
}

-static int page_referenced_anon(struct page *page)
+static int page_referenced_anon(struct page *page,
+ struct mem_container *mem_cont)
{
unsigned int mapcount;
struct anon_vma *anon_vma;
@@ -311,6 +312,13 @@ static int page_referenced_anon(struct p

mapcount = page_mapcount(page);
list_for_each_entry(vma, &anon_vma->head, anon_vma_node) {
+ /*
+ * If we are reclaiming on behalf of a container, skip

```

```

+ * counting on behalf of references from different
+ * containers
+ */
+ if (mem_cont && (mm_container(vma->vm_mm) != mem_cont))
+ continue;
+ referenced += page_referenced_one(page, vma, &mapcount);
+ if (!mapcount)
+ break;
@@ -331,7 +339,8 @@ static int page_referenced_anon(struct p
+
+ * This function is only called from page_referenced for object-based pages.
+ */
-static int page_referenced_file(struct page *page)
+static int page_referenced_file(struct page *page,
+ struct mem_container *mem_cont)
{
+ unsigned int mapcount;
+ struct address_space *mapping = page->mapping;
@@ -364,6 +373,13 @@ static int page_referenced_file(struct p
+ mapcount = page_mapcount(page);

+ vma_prio_tree_foreach(vma, &iter, &mapping->i_mmap, pgoff, pgoff) {
+ /*
+ * If we are reclaiming on behalf of a container, skip
+ * counting on behalf of references from different
+ * containers
+ */
+ if (mem_cont && (mm_container(vma->vm_mm) != mem_cont))
+ continue;
+ if ((vma->vm_flags & (VM_LOCKED|VM_MAYSHARE))
+ == (VM_LOCKED|VM_MAYSHARE)) {
+ referenced++;
@@ -386,7 +402,8 @@ static int page_referenced_file(struct p
+ * Quick test_and_clear_referenced for all mappings to a page,
+ * returns the number of ptes which referenced the page.
+ */
-int page_referenced(struct page *page, int is_locked)
+int page_referenced(struct page *page, int is_locked,
+ struct mem_container *mem_cont)
{
+ int referenced = 0;

@@ -398,14 +415,15 @@ int page_referenced(struct page *page, i

+ if (page_mapped(page) && page->mapping) {
+ if (PageAnon(page))
- referenced += page_referenced_anon(page);
+ referenced += page_referenced_anon(page, mem_cont);

```

```

    else if (is_locked)
-   referenced += page_referenced_file(page);
+   referenced += page_referenced_file(page, mem_cont);
    else if (TestSetPageLocked(page))
        referenced++;
    else {
        if (page->mapping)
-   referenced += page_referenced_file(page);
+   referenced +=
+   page_referenced_file(page, mem_cont);
        unlock_page(page);
    }
}
diff -puN mm/vmscan.c~mem-control-per-container-page-referenced mm/vmscan.c
--- linux-2.6.23-rc1-mm1/mm/vmscan.c~mem-control-per-container-page-referenced 2007-07-28
01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/mm/vmscan.c 2007-07-28 01:12:50.000000000 +0530
@@ -479,7 +479,7 @@ static unsigned long shrink_page_list(st
    if (PageWriteback(page))
        goto keep_locked;

-   referenced = page_referenced(page, 1);
+   referenced = page_referenced(page, 1, sc->mem_container);
    /* In active use or really unfreeable? Activate it. */
    if (sc->order <= PAGE_ALLOC_COSTLY_ORDER &&
        referenced && page_mapping_inuse(page))
@@ -971,7 +971,7 @@ force_reclaim_mapped:
    if (page_mapped(page)) {
        if (!reclaim_mapped ||
            (total_swap_pages == 0 && PageAnon(page)) ||
-        page_referenced(page, 0)) {
+        page_referenced(page, 0, sc->mem_container)) {
            list_add(&page->lru, &l_active);
            continue;
        }
diff -puN include/linux/memcontrol.h~mem-control-per-container-page-referenced
include/linux/memcontrol.h
---
linux-2.6.23-rc1-mm1/include/linux/memcontrol.h~mem-control-per-container-page-referenced 20
07-07-28 01:12:50.000000000 +0530
+++ linux-2.6.23-rc1-mm1-balbir/include/linux/memcontrol.h 2007-07-28 01:12:50.000000000
+0530
@@ -42,6 +42,7 @@ extern unsigned long mem_container_isola
    int active);
extern void mem_container_out_of_memory(struct mem_container *mem);
extern int mem_container_cache_charge(struct page *page, struct mm_struct *mm);
+extern struct mem_container *mm_container(struct mm_struct *mm);

```

```
static inline void mem_container_uncharge_page(struct page *page)
{
@@ -91,6 +92,11 @@ static inline int mem_container_cache_ch
    return 0;
}
```

```
+static inline struct mem_container *mm_container(struct mm_struct *mm)
+{
+ return NULL;
+}
+
#endif /* CONFIG_CONTAINER_MEM_CONT */
```

```
#endif /* _LINUX_MEMCONTROL_H */
```

```
diff -puN mm/memcontrol.c~mem-control-per-container-page-referenced mm/memcontrol.c
```

```
---
linux-2.6.23-rc1-mm1/mm/memcontrol.c~mem-control-per-container-page-referenced 2007-07-28
01:12:50.000000000 +0530
```

```
+++ linux-2.6.23-rc1-mm1-balbir/mm/memcontrol.c 2007-07-28 01:12:50.000000000 +0530
```

```
@@ -100,6 +100,11 @@ struct mem_container *mem_container_from
    struct mem_container, css);
}
```

```
+inline struct mem_container *mm_container(struct mm_struct *mm)
+{
+ return rcu_dereference(mm->mem_container);
+}
+
```

```
void mm_init_container(struct mm_struct *mm, struct task_struct *p)
{
    struct mem_container *mem;
```

—

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

Subject: Re: [-mm PATCH 6/9] Memory controller add per container LRU and reclaim (v4)
Posted by [Dhaval Giani](#) on Mon, 30 Jul 2007 13:37:58 GMT

Hi Balbir,

```
> diff -puN mm/memcontrol.c~mem-control-lru-and-reclaim mm/memcontrol.c
> --- linux-2.6.23-rc1-mm1/mm/memcontrol.c~mem-control-lru-and-reclaim 2007-07-28
01:12:50.000000000 +0530
> +++ linux-2.6.23-rc1-mm1-balbir/mm/memcontrol.c 2007-07-28 01:12:50.000000000 +0530

> /*
>  * The memory controller data structure. The memory controller controls both
> @@ -51,6 +54,10 @@ struct mem_container {
>  */
>  struct list_head active_list;
>  struct list_head inactive_list;
> + /*
> +  * spin_lock to protect the per container LRU
> +  */
> + spinlock_t lru_lock;
> };
```

The spinlock is not annotated by lockdep. The following patch should do it.

Signed-off-by: Dhaval Giani <dhaval@linux.vnet.ibm.com>

Signed-off-by: Gautham Shenoy R <ego@in.ibm.com>

Index: linux-2.6.23-rc1/mm/memcontrol.c

```
=====
--- linux-2.6.23-rc1.orig/mm/memcontrol.c 2007-07-30 17:27:24.000000000 +0530
+++ linux-2.6.23-rc1/mm/memcontrol.c 2007-07-30 18:43:40.000000000 +0530
@@ -501,6 +501,9 @@
```

```
static struct mem_container init_mem_container;

+/* lockdep should know about lru_lock */
+static struct lock_class_key lru_lock_key;
+
static struct container_subsys_state *
mem_container_create(struct container_subsys *ss, struct container *cont)
{
@@ -519,6 +522,7 @@
    INIT_LIST_HEAD(&mem->active_list);
    INIT_LIST_HEAD(&mem->inactive_list);
    spin_lock_init(&mem->lru_lock);
+ lockdep_set_class(&mem->lru_lock, &lru_lock_key);
    mem->control_type = MEM_CONTAINER_TYPE_ALL;
    return &mem->css;
```

```
}  
--  
regards,  
Dhaval
```

I would like to change the world but they don't give me the source code!

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

Subject: Re: [-mm PATCH 6/9] Memory controller add per container LRU and reclaim (v4)

Posted by [Peter Zijlstra](#) on Mon, 30 Jul 2007 13:59:20 GMT

[View Forum Message](#) <> [Reply to Message](#)

On Mon, 2007-07-30 at 19:07 +0530, Dhaval Giani wrote:

```
> Hi Balbir,  
>  
> > diff -puN mm/memcontrol.c~mem-control-lru-and-reclaim mm/memcontrol.c  
> > --- linux-2.6.23-rc1-mm1/mm/memcontrol.c~mem-control-lru-and-reclaim 2007-07-28  
> > 01:12:50.000000000 +0530  
> > +++ linux-2.6.23-rc1-mm1-balbir/mm/memcontrol.c 2007-07-28 01:12:50.000000000 +0530  
>  
> > /*  
> >  * The memory controller data structure. The memory controller controls both  
> > @@ -51,6 +54,10 @@ struct mem_container {  
> >  */  
> > struct list_head active_list;  
> > struct list_head inactive_list;  
> > + /*  
> > +  * spin_lock to protect the per container LRU  
> > +  */  
> > + spinlock_t lru_lock;  
> > };  
>  
> The spinlock is not annotated by lockdep. The following patch should do  
> it.
```

One does not need explicit lockdep annotations unless there is a non obvious use of the locks. A typical example of that would be the inode locks, that get placed differently in the various filesystem's locking hierarchy and might hence seem to generate contradictory locking rules - even though they are consistent within a particular filesystem.

So unless there are 2 or more distinct locking hierarchies this one lock partakes in, there is no need for this annotation.

Was this patch driven by a lockdep report?

```
> Signed-off-by: Dhaval Giani <dhaval@linux.vnet.ibm.com>
> Signed-off-by: Gautham Shenoy R <ego@in.ibm.com>
>
>
> Index: linux-2.6.23-rc1/mm/memcontrol.c
> =====
> --- linux-2.6.23-rc1.orig/mm/memcontrol.c 2007-07-30 17:27:24.000000000 +0530
> +++ linux-2.6.23-rc1/mm/memcontrol.c 2007-07-30 18:43:40.000000000 +0530
> @@ -501,6 +501,9 @@
>
> static struct mem_container init_mem_container;
>
> +/* lockdep should know about lru_lock */
> +static struct lock_class_key lru_lock_key;
> +
> static struct container_subsys_state *
> mem_container_create(struct container_subsys *ss, struct container *cont)
> {
> @@ -519,6 +522,7 @@
> INIT_LIST_HEAD(&mem->active_list);
> INIT_LIST_HEAD(&mem->inactive_list);
> spin_lock_init(&mem->lru_lock);
> + lockdep_set_class(&mem->lru_lock, &lru_lock_key);
> mem->control_type = MEM_CONTAINER_TYPE_ALL;
> return &mem->css;
> }
```

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

Subject: Re: [-mm PATCH 6/9] Memory controller add per container LRU and reclaim (v4)

Posted by [Gautham R Shenoy](#) on Mon, 30 Jul 2007 14:28:18 GMT

[View Forum Message](#) <> [Reply to Message](#)

On Mon, Jul 30, 2007 at 07:07:58PM +0530, Dhaval Giani wrote:

> Hi Balbir,

>

> > diff -puN mm/memcontrol.c~mem-control-lru-and-reclaim mm/memcontrol.c

> > --- linux-2.6.23-rc1-mm1/mm/memcontrol.c~mem-control-lru-and-reclaim 2007-07-28

01:12:50.000000000 +0530

> > +++ linux-2.6.23-rc1-mm1-balbir/mm/memcontrol.c 2007-07-28 01:12:50.000000000 +0530

```

>
>> /*
>>  * The memory controller data structure. The memory controller controls both
>> @@ -51,6 +54,10 @@ struct mem_container {
>>  */
>>  struct list_head active_list;
>>  struct list_head inactive_list;
>> + /*
>> +  * spin_lock to protect the per container LRU
>> +  */
>> + spinlock_t lru_lock;
>> };
>
> The spinlock is not annotated by lockdep. The following patch should do
> it.
>
> Signed-off-by: Dhaval Giani <dhaval@linux.vnet.ibm.com>
> Signed-off-by: Gautham Shenoy R <ego@in.ibm.com>
> ^^^^^^^^^^^^^^^^^^
> Gautham R Shenoy
>
>
> Index: linux-2.6.23-rc1/mm/memcontrol.c
> =====
> --- linux-2.6.23-rc1.orig/mm/memcontrol.c 2007-07-30 17:27:24.000000000 +0530
> +++ linux-2.6.23-rc1/mm/memcontrol.c 2007-07-30 18:43:40.000000000 +0530
> @@ -501,6 +501,9 @@
>
>  static struct mem_container init_mem_container;
>
>  +/* lockdep should know about lru_lock */
>  +static struct lock_class_key lru_lock_key;
>  +
>  static struct container_subsys_state *
>  mem_container_create(struct container_subsys *ss, struct container *cont)
>  {
>  @@ -519,6 +522,7 @@
>  INIT_LIST_HEAD(&mem->active_list);
>  INIT_LIST_HEAD(&mem->inactive_list);
>  spin_lock_init(&mem->lru_lock);
>  + lockdep_set_class(&mem->lru_lock, &lru_lock_key);
>  mem->control_type = MEM_CONTAINER_TYPE_ALL;
>  return &mem->css;
>  }
>  --
> regards,
> Dhaval
>

```

> I would like to change the world but they don't give me the source code!

--

Gautham R Shenoy
Linux Technology Center
IBM India.

"Freedom comes with a price tag of responsibility, which is still a bargain,
because Freedom is priceless!"

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

Subject: Re: [-mm PATCH 4/9] Memory controller memory accounting (v4)
Posted by [yamamoto](#) on Tue, 31 Jul 2007 03:38:32 GMT
[View Forum Message](#) <> [Reply to Message](#)

```
> + lock_meta_page(page);
> + /*
> +  * Check if somebody else beat us to allocating the meta_page
> +  */
> + race_mp = page_get_meta_page(page);
> + if (race_mp) {
> +     kfree(mp);
> +     mp = race_mp;
> +     atomic_inc(&mp->ref_cnt);
> +     res_counter_uncharge(&mem->res, 1);
> +     goto done;
> + }
```

i think you need css_put here.

YAMAMOTO Takashi

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

Subject: Re: [-mm PATCH 6/9] Memory controller add per container LRU and
reclaim (v4)
Posted by [yamamoto](#) on Tue, 31 Jul 2007 05:14:59 GMT
[View Forum Message](#) <> [Reply to Message](#)

```
> + unsigned long mem_container_isolate_pages(unsigned long nr_to_scan,
> +     struct list_head *dst,
```

```

> + unsigned long *scanned, int order,
> + int mode, struct zone *z,
> + struct mem_container *mem_cont,
> + int active)
> +{
> + unsigned long nr_taken = 0;
> + struct page *page;
> + unsigned long scan;
> + LIST_HEAD(mp_list);
> + struct list_head *src;
> + struct meta_page *mp;
> +
> + if (active)
> + src = &mem_cont->active_list;
> + else
> + src = &mem_cont->inactive_list;
> +
> + for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
> + mp = list_entry(src->prev, struct meta_page, lru);

```

what prevents another thread from freeing mp here?

```

> + spin_lock(&mem_cont->lru_lock);
> + if (mp)
> + page = mp->page;
> + spin_unlock(&mem_cont->lru_lock);
> + if (!mp)
> + continue;

```

YAMAMOTO Takashi

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

Subject: Re: [-mm PATCH 4/9] Memory controller memory accounting (v4)
Posted by [Vaidyanathan Srinivas](#) on Tue, 31 Jul 2007 12:44:26 GMT
[View Forum Message](#) <> [Reply to Message](#)

YAMAMOTO Takashi wrote:

```

>> + lock_meta_page(page);
>> + /*
>> + * Check if somebody else beat us to allocating the meta_page
>> + */
>> + race_mp = page_get_meta_page(page);
>> + if (race_mp) {
>> + kfree(mp);

```

```
>> + mp = race_mp;
>> + atomic_inc(&mp->ref_cnt);
>> + res_counter_uncharge(&mem->res, 1);
>> + goto done;
>> + }
>
> i think you need css_put here.
```

Thats correct. We do need css_put in this path.

Thanks,
Vaidy

```
> YAMAMOTO Takashi
> _____
> 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: [-mm PATCH 6/9] Memory controller add per container LRU and reclaim (v4)

Posted by [Vaidyanathan Srinivas](#) on Tue, 31 Jul 2007 12:55:40 GMT

[View Forum Message](#) <> [Reply to Message](#)

YAMAMOTO Takashi wrote:

```
>> +unsigned long mem_container_isolate_pages(unsigned long nr_to_scan,
>> +    struct list_head *dst,
>> +    unsigned long *scanned, int order,
>> +    int mode, struct zone *z,
>> +    struct mem_container *mem_cont,
>> +    int active)
>> +{
>> + unsigned long nr_taken = 0;
>> + struct page *page;
>> + unsigned long scan;
>> + LIST_HEAD(mp_list);
>> + struct list_head *src;
>> + struct meta_page *mp;
>> +
>> + if (active)
>> +   src = &mem_cont->active_list;
>> + else
```

```
>> + src = &mem_cont->inactive_list;
>> +
>> + for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
>> + mp = list_entry(src->prev, struct meta_page, lru);
>
> what prevents another thread from freeing mp here?
```

mem_cont->lru_lock protects the list and validity of mp. If we hold mem_cont->lru_lock for this entire loop, then we preserve the validity of mp. However that will be holding up container charge and uncharge.

This entire routine is called with zone->lru_lock held by the caller. So within a zone, this routine is serialized.

However page uncharge may race with isolate page. But will that lead to any corruption of the list? We may be holding the lock for too much time just to be on the safe side.

Please allow us some time to verify whether this is indeed inadequate locking that will lead to corruption of the list.

Thanks for pointing out this situation.
--Vaidy

```
>> + spin_lock(&mem_cont->lru_lock);
>> + if (mp)
>> + page = mp->page;
>> + spin_unlock(&mem_cont->lru_lock);
>> + if (!mp)
>> + continue;
>
> YAMAMOTO Takashi
> _____
> 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: [-mm PATCH 6/9] Memory controller add per container LRU and reclaim (v4)

Posted by [Vaidyanathan Srinivas](#) on Tue, 07 Aug 2007 18:30:11 GMT

[View Forum Message](#) <> [Reply to Message](#)

Vaidyanathan Srinivasan wrote:

```
>
> YAMAMOTO Takashi wrote:
>>> +unsigned long mem_container_isolate_pages(unsigned long nr_to_scan,
>>> + struct list_head *dst,
>>> + unsigned long *scanned, int order,
>>> + int mode, struct zone *z,
>>> + struct mem_container *mem_cont,
>>> + int active)
>>> +{
>>> + unsigned long nr_taken = 0;
>>> + struct page *page;
>>> + unsigned long scan;
>>> + LIST_HEAD(mp_list);
>>> + struct list_head *src;
>>> + struct meta_page *mp;
>>> +
>>> + if (active)
>>> + src = &mem_cont->active_list;
>>> + else
>>> + src = &mem_cont->inactive_list;
>>> +
>>> + for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
>>> + mp = list_entry(src->prev, struct meta_page, lru);
>> what prevents another thread from freeing mp here?
>
> mem_cont->lru_lock protects the list and validity of mp. If we hold
> mem_cont->lru_lock for this entire loop, then we preserve the validity
> of mp. However that will be holding up container charge and uncharge.
>
> This entire routine is called with zone->lru_lock held by the caller.
> So within a zone, this routine is serialized.
>
> However page uncharge may race with isolate page. But will that lead
> to any corruption of the list? We may be holding the lock for too
> much time just to be on the safe side.
>
> Please allow us some time to verify whether this is indeed inadequate
> locking that will lead to corruption of the list.
```

I did few runs and checked for ref_cnt on meta_page and there seems to be a race between isolate pages and uncharge. We will probably have to increase the ref_cnt on meta_page while we are isolating it. I am trying to see if we can solve the problem by manipulating the ref_cnt on the meta_page.

--Vaidy

> Thanks for pointing out this situation.
> --Vaidy
>
>>> + spin_lock(&mem_cont->lru_lock);
>>> + if (mp)
>>> + page = mp->page;
>>> + spin_unlock(&mem_cont->lru_lock);
>>> + if (!mp)
>>> + continue;
>> YAMAMOTO Takashi
>>

>> 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>
>

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

Subject: Re: [-mm PATCH 0/9] Memory controller introduction (v4)
Posted by [KAMEZAWA Hiroyuki](#) on Wed, 08 Aug 2007 03:51:39 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Sat, 28 Jul 2007 01:39:37 +0530
Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> At OLS, the resource management BOF, it was discussed that we need to manage
> RSS and unmapped page cache together. This patchset is a step towards that
>
Can I make a question ? Why limiting RSS instead of # of used pages per
container ? Maybe because of shared pages between container....
I'm sorry that I don't have much knowledge about history of this work.
If already discussed, the pointer to log is helpful.

Thanks,
-Kame

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

Subject: Re: [-mm PATCH 0/9] Memory controller introduction (v4)
Posted by [KAMEZAWA Hiroyuki](#) on Wed, 08 Aug 2007 07:51:31 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Wed, 8 Aug 2007 12:51:39 +0900
KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:

> On Sat, 28 Jul 2007 01:39:37 +0530
> Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> > At OLS, the resource management BOF, it was discussed that we need to manage
> > RSS and unmapped page cache together. This patchset is a step towards that
> >
> Can I make a question ? Why limiting RSS instead of # of used pages per
> container ? Maybe because of shared pages between container....
Sorry....Ignore above question.
I didn't understand what mem_container_charge() accounts and limits.
It controls # of meta_pages.

Sorry,
-Kame

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

Subject: Re: [-mm PATCH 0/9] Memory controller introduction (v4)
Posted by [Vaidyanathan Srinivas](#) on Fri, 10 Aug 2007 04:49:49 GMT
[View Forum Message](#) <> [Reply to Message](#)

KAMEZAWA Hiroyuki wrote:
> On Wed, 8 Aug 2007 12:51:39 +0900
> KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
>
>> On Sat, 28 Jul 2007 01:39:37 +0530
>> Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
>>> At OLS, the resource management BOF, it was discussed that we need to manage
>>> RSS and unmapped page cache together. This patchset is a step towards that
>>>
>> Can I make a question ? Why limiting RSS instead of # of used pages per
>> container ? Maybe because of shared pages between container....
> Sorry....Ignore above question.
> I didn't understand what mem_container_charge() accounts and limits.
> It controls # of meta_pages.

Hi Kame,

Actually the number of pages resident in memory brought in by a container is charged. However each such page will have a meta_page allocated to keep the extra data.

Yes, the accounting counts the number of meta_page which is same as the number of mapped and unmapped (pagecache) pages brought into the system memory by this container. Whether pagecache pages should be included or not is configurable per container through the 'type' file in containerfs.

--Vaidy

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

Subject: Re: [-mm PATCH 8/9] Memory controller add switch to control what type of pages to limit (v4)

Posted by [yamamoto](#) on Mon, 13 Aug 2007 00:33:48 GMT

[View Forum Message](#) <> [Reply to Message](#)

> Choose if we want cached pages to be accounted or not. By default both
> are accounted for. A new set of tunables are added.

>

> echo -n 1 > mem_control_type

>

> switches the accounting to account for only mapped pages

>

> echo -n 2 > mem_control_type

>

> switches the behaviour back

MEM_CONTAINER_TYPE_ALL is 3, not 2.

YAMAMOTO Takashi

```
> +enum {
> + MEM_CONTAINER_TYPE_UNSPEC = 0,
> + MEM_CONTAINER_TYPE_MAPPED,
> + MEM_CONTAINER_TYPE_CACHED,
> + MEM_CONTAINER_TYPE_ALL,
> + MEM_CONTAINER_TYPE_MAX,
> +} mem_control_type;
> +
> +static struct mem_container init_mem_container;
```

```
> + mem = rcu_dereference(mm->mem_container);
> + if (mem->control_type == MEM_CONTAINER_TYPE_ALL)
> + return mem_container_charge(page, mm);
> + else
> + return 0;
```

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>

Subject: Re: [-mm PATCH 8/9] Memory controller add switch to control what type of pages to limit (v4)

Posted by [Balbir Singh](#) on Mon, 13 Aug 2007 05:38:58 GMT

[View Forum Message](#) <> [Reply to Message](#)

YAMAMOTO Takashi wrote:

>> Choose if we want cached pages to be accounted or not. By default both
>> are accounted for. A new set of tunables are added.

>>

>> echo -n 1 > mem_control_type

>>

>> switches the accounting to account for only mapped pages

>>

>> echo -n 2 > mem_control_type

>>

>> switches the behaviour back

>

> MEM_CONTAINER_TYPE_ALL is 3, not 2.

>

Thanks, I'll fix the comment on the top.

> YAMAMOTO Takashi

>

>> +enum {

>> + MEM_CONTAINER_TYPE_UNSPEC = 0,

>> + MEM_CONTAINER_TYPE_MAPPED,

>> + MEM_CONTAINER_TYPE_CACHED,

>> + MEM_CONTAINER_TYPE_ALL,

>> + MEM_CONTAINER_TYPE_MAX,

>> +} mem_control_type;

>> +

>> +static struct mem_container init_mem_container;

>

>> + mem = rcu_dereference(mm->mem_container);

>> + if (mem->control_type == MEM_CONTAINER_TYPE_ALL)

>> + return mem_container_charge(page, mm);

```
>> + else
>> + return 0;
```

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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

Subject: Re: [-mm PATCH 8/9] Memory controller add switch to control what type of pages to limit (v4)

Posted by [yamamoto](#) on Mon, 13 Aug 2007 06:04:26 GMT

[View Forum Message](#) <> [Reply to Message](#)

> YAMAMOTO Takashi wrote:

> >> Choose if we want cached pages to be accounted or not. By default both
> >> are accounted for. A new set of tunables are added.

> >>

> >> echo -n 1 > mem_control_type

> >>

> >> switches the accounting to account for only mapped pages

> >>

> >> echo -n 2 > mem_control_type

> >>

> >> switches the behaviour back

> >

> > MEM_CONTAINER_TYPE_ALL is 3, not 2.

> >

>

> Thanks, I'll fix the comment on the top.

>

> > YAMAMOTO Takashi

> >

> >> +enum {

> >> + MEM_CONTAINER_TYPE_UNSPEC = 0,

> >> + MEM_CONTAINER_TYPE_MAPPED,

> >> + MEM_CONTAINER_TYPE_CACHED,

what's MEM_CONTAINER_TYPE_CACHED, btw?

it seems that nothing distinguishes it from MEM_CONTAINER_TYPE_MAPPED.

YAMAMOTO Takashi

Subject: Re: [-mm PATCH 8/9] Memory controller add switch to control what type of pages to limit (v4)

Posted by [Balbir Singh](#) on Mon, 13 Aug 2007 06:17:03 GMT

[View Forum Message](#) <> [Reply to Message](#)

YAMAMOTO Takashi wrote:

>> YAMAMOTO Takashi wrote:

>>>> Choose if we want cached pages to be accounted or not. By default both

>>>> are accounted for. A new set of tunables are added.

>>>>

>>>> echo -n 1 > mem_control_type

>>>>

>>>> switches the accounting to account for only mapped pages

>>>>

>>>> echo -n 2 > mem_control_type

>>>>

>>>> switches the behaviour back

>>> MEM_CONTAINER_TYPE_ALL is 3, not 2.

>>>

>> Thanks, I'll fix the comment on the top.

>>

>>> YAMAMOTO Takashi

>>>

>>>> +enum {

>>>> + MEM_CONTAINER_TYPE_UNSPEC = 0,

>>>> + MEM_CONTAINER_TYPE_MAPPED,

>>>> + MEM_CONTAINER_TYPE_CACHED,

>

> what's MEM_CONTAINER_TYPE_CACHED, btw?

> it seems that nothing distinguishes it from MEM_CONTAINER_TYPE_MAPPED.

>

The important types are currently MEM_CONTAINER_TYPE_MAPPED and MEM_CONTAINER_TYPE_ALL.

I added the types for future use (may be for finer accounting of types, later).

> YAMAMOTO Takashi

--

Warm Regards,
Balbir Singh

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

Subject: Re: [-mm PATCH 4/9] Memory controller memory accounting (v4)
Posted by [yamamoto](#) on Wed, 15 Aug 2007 08:44:53 GMT
[View Forum Message](#) <> [Reply to Message](#)

```
> YAMAMOTO Takashi wrote:
> >> + lock_meta_page(page);
> >> + /*
> >> + * Check if somebody else beat us to allocating the meta_page
> >> + */
> >> + race_mp = page_get_meta_page(page);
> >> + if (race_mp) {
> >> + kfree(mp);
> >> + mp = race_mp;
> >> + atomic_inc(&mp->ref_cnt);
> >> + res_counter_uncharge(&mem->res, 1);
> >> + goto done;
> >> + }
> >
> > i think you need css_put here.
>
> Thats correct. We do need css_put in this path.
>
> Thanks,
> Vaidy
```

v5 still seems to have the problem.

YAMAMOTO Takashi

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

Subject: Re: [-mm PATCH 4/9] Memory controller memory accounting (v4)
Posted by [Balbir Singh](#) on Wed, 15 Aug 2007 10:02:30 GMT
[View Forum Message](#) <> [Reply to Message](#)

YAMAMOTO Takashi wrote:

```
>> YAMAMOTO Takashi wrote:
>>>> + lock_meta_page(page);
>>>> + /*
>>>> +  * Check if somebody else beat us to allocating the meta_page
>>>> + */
>>>> + race_mp = page_get_meta_page(page);
>>>> + if (race_mp) {
>>>> +  kfree(mp);
>>>> +  mp = race_mp;
>>>> +  atomic_inc(&mp->ref_cnt);
>>>> +  res_counter_uncharge(&mem->res, 1);
>>>> +  goto done;
>>>> + }
>> i think you need css_put here.
>> Thats correct. We do need css_put in this path.
>>
>> Thanks,
>> Vaidy
>
> v5 still seems to have the problem.
>
> YAMAMOTO Takashi
>
```

Hi,

I've got the fix in v6 now, thanks for spotting it.

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

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