
Subject: [-mm PATCH 0/9] Memory controller introduction (v6)
Posted by [Balbir Singh](#) on Fri, 17 Aug 2007 08:42:28 GMT
[View Forum Message](#) <> [Reply to Message](#)

Here's version 6 of the memory controller (against 2.6.23-rc2-mm2).

The tests that were run has been included in the `_Test Results` section_ below.

Changelog since version 5

1. Ported to 2.6.23-rc2-mm2
2. Added a `css_put()` in the case of race between allocation of `page_containers` (YAMAMOTO Takashi)

Changelog since version 4

1. Renamed `meta_page` to `page_container` (Nick Piggin)
2. Moved locking from page flags to last bit of the `page_container` pointer (Nick Piggin)
3. Fixed a rare race in `mem_container_isolate_pages` (YAMAMOTO Takashi)

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 memory /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/memory.limit`
example
`echo -n 204800 > /container/a/memory.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.

Dhaval ran several tests on v6 and gave his thumbs up to the controller (a hard to achieve goal :-)).

Run kernbench stress

Three simultaneously and with one inside a container of 800 MB.

Kernbench results running within the container of 800 MB

Thu Aug 16 22:34:59 IST 2007
2.6.23-rc2-mm2-mem-v6
Average Half load -j 4 Run (std deviation):
Elapsed Time 466.548 (47.6014)
User Time 876.598 (10.5273)
System Time 223.136 (1.29247)
Percent CPU 237.2 (23.2744)
Context Switches 146351 (6539.91)
Sleeps 174003 (5031.94)

Average Optimal load -j 32 Run (std deviation):
Elapsed Time 423.496 (60.625)
User Time 897.285 (23.0391)
System Time 228.836 (6.11205)
Percent CPU 257.1 (40.9022)
Context Switches 262134 (123397)
Sleeps 270815 (103597)

Kernbench results running within the default container

Thu Aug 16 22:34:33 IST 2007
2.6.23-rc2-mm2-mem-v6
Average Half load -j 4 Run (std deviation):
Elapsed Time 424.17 (3.45908)
User Time 841.992 (5.40178)
System Time 213.01 (0.706258)
Percent CPU 248.2 (0.83666)

Context Switches 134254 (9535.83)
Sleeps 167359 (6858.45)

Average Optimal load -j 32 Run (std deviation):
Elapsed Time 407.092 (108.932)
User Time 878.493 (38.6575)
System Time 222.155 (9.77127)
Percent CPU 278.4 (91.5826)
Context Switches 253836 (127708)
Sleeps 263760 (103468)

Kernbench results running within the default container

Thu Aug 16 22:34:52 IST 2007
2.6.23-rc2-mm2-mem-v6
Average Half load -j 4 Run (std deviation):
Elapsed Time 465.038 (48.5147)
User Time 874.742 (5.86563)
System Time 222.194 (0.561676)
Percent CPU 237.2 (22.5211)
Context Switches 144040 (7052.23)
Sleeps 172130 (5608.73)

Average Optimal load -j 32 Run (std deviation):
Elapsed Time 426.25 (62.34)
User Time 893.938 (20.6732)
System Time 227.717 (5.87502)
Percent CPU 255.6 (40.7163)
Context Switches 259560 (122953)
Sleeps 267402 (101801)

Ebizzy results

ebizzy on top of rc2-mm2 (without mem-controller) gave this

ebizzy 0.3, Copyright 2006 Intel Corporation
Written by Val Henson <val_henson@linux.intel.com>
always_mmap 0
never_mmap 0
chunks 10
prevent_coalescing_using_permissions 0
prevent_coalescing_using_holes 0
records 1048576
records_per_thread 65536
random_size 0

chunk_size 262144
threads 16
verbose 1
linear 0
page size 4096
Allocated memory
Wrote memory
Threads starting
Threads finished

real 5m54.362s
user 33m49.707s
sys 4m11.100s

ebizzy with 2500 pages, and oprofiled.

ebizzy 0.3, Copyright 2006 Intel Corporation
Written by Val Henson <val_henson@linux.intel.com>

always_mmap 0
never_mmap 0
chunks 10
prevent coalescing using permissions 0
prevent coalescing using holes 0
records 1048576
records per thread 65536
random_size 0
chunk_size 262144
threads 16
verbose 1
linear 0
page size 4096
Allocated memory
Wrote memory
Threads starting
Threads finished

real 6m15.561s
user 36m2.203s
sys 3m30.829s

Documentation

An article describing the design of the memory controller is available
at <http://lwn.net/Articles/243795/>

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 (v6)
Posted by [Balbir Singh](#) on Fri, 17 Aug 2007 08:42:38 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-rc2-mm2-balbir/include/linux/res_counter.h 2007-08-17 13:14:18.000000000
+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 opeate 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-rc2-mm2/init/Kconfig~res_counters_infra 2007-08-17 13:14:18.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/init/Kconfig 2007-08-17 13:14:18.000000000 +0530
@@ -337,6 +337,13 @@ 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-rc2-mm2/kernel/Makefile~res_counters_infra 2007-08-17 13:14:18.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/kernel/Makefile 2007-08-17 13:14:18.000000000 +0530
@@ -58,6 +58,7 @@ 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-rc2-mm2-balbir/kernel/res_counter.c 2007-08-17 13:14:18.000000000 +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 (v6)
 Posted by [Balbir Singh](#) on Fri, 17 Aug 2007 08:42:48 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-rc2-mm2/include/linux/container_subsys.h~mem-control-setup 2007-08-17
13:14:19.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/include/linux/container_subsys.h 2007-08-17
13:14:19.000000000 +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-rc2-mm2-balbir/include/linux/memcontrol.h 2007-08-17 13:14:19.000000000
+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-rc2-mm2/init/Kconfig~mem-control-setup 2007-08-17 13:14:19.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/init/Kconfig 2007-08-17 13:14:19.000000000 +0530
@@ -364,6 +364,13 @@ config SYSFS_DEPRECATED
    If you are using a distro that was released in 2006 or later,
```

it should be safe to say N here.

```
+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-rc2-mm2/mm/Makefile~mem-control-setup 2007-08-17 13:14:19.000000000
+0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/Makefile 2007-08-17 13:14:19.000000000 +0530
@@ -31,4 +31,5 @@ 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-rc2-mm2-balbir/mm/memcontrol.c 2007-08-17 13:14:19.000000000 +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 page_container page is associated with every page descriptor. The
+ * page_container helps us identify information about the container
+ */
+struct page_container {
+ 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 (v6)
Posted by [Balbir Singh](#) on Fri, 17 Aug 2007 08:42:59 GMT
[View Forum Message](#) <> [Reply to Message](#)

From: Pavel Emelianov <xemul@openvz.org>

Changelog for v5

1. Remove inclusion of memcontrol.h from mm_types.h

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 the page has a `page_container` associated with it.

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

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

```

include/linux/memcontrol.h | 36 ++++++
include/linux/mm_types.h | 6 ++++
include/linux/sched.h | 1
kernel/fork.c | 11 ++++++
mm/memcontrol.c | 57 ++++++-----
5 files changed, 104 insertions(+), 7 deletions(-)

```

```

diff -puN include/linux/memcontrol.h~mem-control-accounting-setup include/linux/memcontrol.h
--- linux-2.6.23-rc2-mm2/include/linux/memcontrol.h~mem-control-accounting-setup 2007-08-17
13:14:19.000000000 +0530

```

```

+++ linux-2.6.23-rc2-mm2-balbir/include/linux/memcontrol.h 2007-08-17 13:14:19.000000000
+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 +20,38 @@

```

```

#ifndef _LINUX_MEMCONTROL_H
#define _LINUX_MEMCONTROL_H

```

```

+struct mem_container;
+struct page_container;
+
+#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_page_container(struct page *page,
+    struct page_container *pc);
+extern struct page_container *page_get_page_container(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_page_container(struct page *page,

```

```

+ struct page_container *pc)
+{
+}
+
+static inline struct page_container *page_get_page_container(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-rc2-mm2/include/linux/mm_types.h~mem-control-accounting-setup 2007-08-17
13:14:19.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/include/linux/mm_types.h 2007-08-17 13:14:19.000000000
+0530
@@ -95,6 +95,9 @@ struct page {
    unsigned int gfp_mask;
    unsigned long trace[8];
#endif
+#ifdef CONFIG_CONTAINER_MEM_CONT
+ unsigned long page_container;
+#endif
};

/*
@@ -226,6 +229,9 @@ 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
};

#endif /* _LINUX_MM_TYPES_H */
diff -puN include/linux/sched.h~mem-control-accounting-setup include/linux/sched.h
--- linux-2.6.23-rc2-mm2/include/linux/sched.h~mem-control-accounting-setup 2007-08-17
13:14:19.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/include/linux/sched.h 2007-08-17 13:14:19.000000000 +0530
@@ -87,6 +87,7 @@ struct sched_param {

#include <asm/processor.h>

+struct mem_container;
+struct exec_domain;

```

```

struct futex_pi_state;
struct bio;
diff -puN kernel/fork.c~mem-control-accounting-setup kernel/fork.c
--- linux-2.6.23-rc2-mm2/kernel/fork.c~mem-control-accounting-setup 2007-08-17
13:14:19.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/kernel/fork.c 2007-08-17 13:14:19.000000000 +0530
@@ -50,6 +50,7 @@
#include <linux/taskstats_kern.h>
#include <linux/random.h>
#include <linux/tty.h>
+#include <linux/memcontrol.h>

#include <asm/pgtable.h>
#include <asm/pgalloc.h>
@@ -328,7 +329,7 @@ __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 +346,14 @@ static struct mm_struct * mm_init(struct
    mm->iocx_list = NULL;
    mm->free_area_cache = TASK_UNMAPPED_BASE;
    mm->cached_hole_size = ~0UL;
+ mm_init_container(mm, p);

    if (likely(!mm_alloc_pgdn(mm))) {
        mm->def_flags = 0;
        return mm;
    }
+
+ mm_free_container(mm);
    free_mm(mm);
    return NULL;
}
@@ -364,7 +368,7 @@ struct mm_struct * mm_alloc(void)
    mm = allocate_mm();
    if (mm) {
        memset(mm, 0, sizeof(*mm));
- mm = mm_init(mm);
+ mm = mm_init(mm, current);
    }
    return mm;
}
@@ -378,6 +382,7 @@ void fastcall __mmdrop(struct mm_struct

```

```

{
    BUG_ON(mm == &init_mm);
    mm_free_pgd(mm);
+ mm_free_container(mm);
    destroy_context(mm);
    free_mm(mm);
}
@@ -498,7 +503,7 @@ static struct mm_struct *dup_mm(struct t
    mm->token_priority = 0;
    mm->last_interval = 0;

- if (!mm_init(mm))
+ if (!mm_init(mm, tsk))
    goto fail_nomem;

    if (init_new_context(tsk, mm))
diff -puN mm/memcontrol.c~mem-control-accounting-setup mm/memcontrol.c
--- linux-2.6.23-rc2-mm2/mm/memcontrol.c~mem-control-accounting-setup 2007-08-17
13:14:19.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/memcontrol.c 2007-08-17 13:14:19.000000000 +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_page_container(struct page *page, struct page_container *pc)
+{
+ page->page_container = (unsigned long)pc;
+}
+
+struct page_container *page_get_page_container(struct page *page)
+{
+ return page->page_container;
+}
+
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 (v6)
Posted by [Balbir Singh](#) on Fri, 17 Aug 2007 08:43:09 GMT
[View Forum Message](#) <> [Reply to Message](#)

Changelog for v6
1. Do a css_put() in the case of a race in allocating page containers
(YAMAMOTO Takashi)

Changelog for v5

1. Rename meta_page to page_container
2. Remove PG_metapage and use the lower bit of the page_container pointer for locking

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 page_container is protected with the last bit of the page_container pointer, this makes handling of race conditions involving simultaneous mappings of a page easier. A reference count is kept in the page_container 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 page container. 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 +++++
mm/filemap.c               | 12 ++-
mm/memcontrol.c            | 166 ++++++++++++++++++++++++++++++++++++++
mm/memory.c               | 43 ++++++++
mm/migrate.c              | 6 +
mm/page_alloc.c           | 3
mm/rmap.c                 | 17 +++++
mm/swap_state.c           | 12 ++-
mm/swapfile.c             | 41 ++++++-----
9 files changed, 293 insertions(+), 27 deletions(-)
```

```
diff -puN include/linux/memcontrol.h~mem-control-accounting include/linux/memcontrol.h
--- linux-2.6.23-rc2-mm2/include/linux/memcontrol.h~mem-control-accounting 2007-08-17
```

```

13:14:19.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/include/linux/memcontrol.h 2007-08-17 13:14:19.000000000
+0530
@@ -30,6 +30,13 @@ extern void mm_free_container(struct mm_
extern void page_assign_page_container(struct page *page,
    struct page_container *pc);
extern struct page_container *page_get_page_container(struct page *page);
+extern int mem_container_charge(struct page *page, struct mm_struct *mm);
+extern void mem_container_uncharge(struct page_container *pc);
+
+static inline void mem_container_uncharge_page(struct page *page)
+{
+ mem_container_uncharge(page_get_page_container(page));
+}

#else /* CONFIG_CONTAINER_MEM_CONT */
static inline void mm_init_container(struct mm_struct *mm,
@@ -51,6 +58,19 @@ static inline struct page_container *pag
    return NULL;
}

+static inline int mem_container_charge(struct page *page, struct mm_struct *mm)
+{
+ return 0;
+}
+
+static inline void mem_container_uncharge(struct page_container *pc)
+{
+}
+
+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
diff -puN mm/filemap.c~mem-control-accounting mm/filemap.c
--- linux-2.6.23-rc2-mm2/mm/filemap.c~mem-control-accounting 2007-08-17 13:14:19.000000000
+0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/filemap.c 2007-08-17 13:14:19.000000000 +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->nrpages--;
@@ -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->nrpages++;
        __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-rc2-mm2/mm/memcontrol.c~mem-control-accounting 2007-08-17
13:14:19.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/memcontrol.c 2007-08-17 13:14:19.000000000 +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;
```

```
@@ -49,6 +54,14 @@ struct mem_container {
};
```

```
/*
```

```
+ * We use the lower bit of the page->page_container pointer as a bit spin
+ * lock. We need to ensure that page->page_container is atleast two
+ * byte aligned (based on comments from Nick Piggin)
+ */
#define PAGE_CONTAINER_LOCK_BIT 0x0
#define PAGE_CONTAINER_LOCK (1 << PAGE_CONTAINER_LOCK_BIT)
+
+/*
 * A page_container page is associated with every page descriptor. The
 * page_container helps us identify information about the container
 */
```

```
@@ -56,6 +69,8 @@ struct page_container {
    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 */
};
```

```
@@ -88,14 +103,157 @@ void mm_free_container(struct mm_struct
    css_put(&mm->mem_container->css);
}
```

```
+static inline int page_container_locked(struct page *page)
+{
+ return bit_spin_is_locked(PAGE_CONTAINER_LOCK_BIT,
+    &page->page_container);
+}
+
```

```

void page_assign_page_container(struct page *page, struct page_container *pc)
{
- page->page_container = (unsigned long)pc;
+ int locked;
+
+ /*
+  * While resetting the page_container we might not hold the
+  * page_container lock. free_hot_cold_page() is an example
+  * of such a scenario
+  */
+ if (pc)
+ VM_BUG_ON(!page_container_locked(page));
+ locked = (page->page_container & PAGE_CONTAINER_LOCK);
+ page->page_container = ((unsigned long)pc | locked);
}

struct page_container *page_get_page_container(struct page *page)
{
- return page->page_container;
+ return (struct page_container *)
+ (page->page_container & ~PAGE_CONTAINER_LOCK);
+}
+
+void __always_inline lock_page_container(struct page *page)
+{
+ bit_spin_lock(PAGE_CONTAINER_LOCK_BIT, &page->page_container);
+ VM_BUG_ON(!page_container_locked(page));
+}
+
+void __always_inline unlock_page_container(struct page *page)
+{
+ bit_spin_unlock(PAGE_CONTAINER_LOCK_BIT, &page->page_container);
+}
+
+/*
+ * 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 page_container *pc, *race_pc;
+
+ /*
+  * Should page_container'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 page_container associated
+ * with it
+ */
+ lock_page_container(page);
+ pc = page_get_page_container(page);
+ /*
+ * The page_container exists and the page has already been accounted
+ */
+ if (pc) {
+     atomic_inc(&pc->ref_cnt);
+     goto done;
+ }
+
+ unlock_page_container(page);
+
+ pc = kzalloc(sizeof(struct page_container), GFP_KERNEL);
+ if (pc == 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 page_container, we should free it on exceeding
+ * the container limit.
+ */
+ if (res_counter_charge(&mem->res, 1)) {
+     css_put(&mem->css);
+     goto free_pc;
+ }
+
+ lock_page_container(page);

```

```

+ /*
+  * Check if somebody else beat us to allocating the page_container
+  */
+ race_pc = page_get_page_container(page);
+ if (race_pc) {
+  kfree(pc);
+  pc = race_pc;
+  atomic_inc(&pc->ref_cnt);
+  res_counter_uncharge(&mem->res, 1);
+  css_put(&mem->css);
+  goto done;
+ }
+
+ atomic_set(&pc->ref_cnt, 1);
+ pc->mem_container = mem;
+ pc->page = page;
+ page_assign_page_container(page, pc);
+
+done:
+ unlock_page_container(page);
+ return 0;
+free_pc:
+ kfree(pc);
+ return -ENOMEM;
+err:
+ unlock_page_container(page);
+ return -ENOMEM;
+}
+
+/*
+ * Uncharging is always a welcome operation, we never complain, simply
+ * uncharge.
+ */
+void mem_container_uncharge(struct page_container *pc)
+{
+ struct mem_container *mem;
+ struct page *page;
+
+ if (!pc)
+  return;
+
+ if (atomic_dec_and_test(&pc->ref_cnt)) {
+  page = pc->page;
+  lock_page_container(page);
+  mem = pc->mem_container;
+  css_put(&mem->css);
+  page_assign_page_container(page, NULL);
+  unlock_page_container(page);

```

```
+ res_counter_uncharge(&mem->res, 1);
+ kfree(pc);
+ }
}
```

```
static ssize_t mem_container_read(struct container *cont, struct cftype *cft,
@@ -150,6 +308,8 @@ 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-rc2-mm2/mm/memory.c~mem-control-accounting 2007-08-17
13:14:19.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/memory.c 2007-08-17 13:14:19.000000000 +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>
@@ -1136,14 +1137,18 @@ 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;
```

```

@@ -1155,8 +1160,11 @@ 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;
}
@@ -1629,6 +1637,9 @@ gotten:
    goto oom;
    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
     */
@@ -1660,7 +1671,9 @@ 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)
@@ -1681,6 +1694,8 @@ unlock:
    put_page(dirty_page);
}
return ret;
+oom_free_new:
+ __free_page(new_page);
oom:
    if (old_page)
        page_cache_release(old_page);
@@ -2085,6 +2100,11 @@ 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);

@@ -2121,8 +2141,10 @@ 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;
}

@@ -2133,6 +2155,7 @@ unlock:
out:
return ret;
out_nomap:
+ mem_container_uncharge_page(page);
pte_unmap_unlock(page_table, ptl);
unlock_page(page);
page_cache_release(page);
@@ -2161,6 +2184,9 @@ 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);

@@ -2178,8 +2204,11 @@ 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;
}

```

```

@@ -2290,6 +2319,11 @@ 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);

/*
@@ -2325,6 +2359,7 @@ static int __do_fault(struct mm_struct *
/* no need to invalidate: a not-present page won't be cached */
update_mmu_cache(vma, address, 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-rc2-mm2/mm/migrate.c~mem-control-accounting 2007-08-17 13:14:19.000000000
+0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/migrate.c 2007-08-17 13:14:19.000000000 +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 = *pte;
diff -puN mm/page_alloc.c~mem-control-accounting mm/page_alloc.c
--- linux-2.6.23-rc2-mm2/mm/page_alloc.c~mem-control-accounting 2007-08-17
13:14:19.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/page_alloc.c 2007-08-17 13:14:19.000000000 +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>
@@ -1018,6 +1019,7 @@ static void fastcall free_hot_cold_page(

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

@@ -2637,6 +2639,7 @@ void __meminit memmap_init_zone(unsigned
    set_page_links(page, zone, nid, pfn);
    init_page_count(page);
    reset_page_mapcount(page);
+ page_assign_page_container(page, NULL);
    SetPageReserved(page);

/*
diff -puN mm/rmap.c~mem-control-accounting mm/rmap.c
--- linux-2.6.23-rc2-mm2/mm/rmap.c~mem-control-accounting 2007-08-17 13:14:19.000000000
+0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/rmap.c 2007-08-17 13:14:19.000000000 +0530
@@ -48,6 +48,7 @@
#include <linux/rcupdate.h>
#include <linux/module.h>
#include <linux/kallsyms.h>
+#include <linux/memcontrol.h>

#include <asm/tlbflush.h>

@@ -550,8 +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);

```

```

+ }
+ }

/*
@@ -582,6 +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
@@ -642,6 +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-rc2-mm2/mm/swapfile.c~mem-control-accounting 2007-08-17
13:14:19.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/swapfile.c 2007-08-17 13:14:19.000000000 +0530
@@ -27,6 +27,7 @@
#include <linux/mutex.h>
#include <linux/capability.h>
#include <linux/syscalls.h>
+#include <linux/memcontrol.h>

#include <asm/pgtable.h>
#include <asm/tlbflush.h>
@@ -506,9 +507,12 @@ 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 +524,7 @@ 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 +534,7 @@ static int unuse_pte_range(struct vm_are
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 +543,12 @@ static int unuse_pte_range(struct vm_are
* Test inline before going to call unuse_pte.
*/
if (unlikely(pte_same(*pte, swp_pte))) {
- unuse_pte(vma, pte++, addr, entry, page);
- 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 +557,16 @@ 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 +577,16 @@ 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 +596,7 @@ 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 +614,9 @@ 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 +625,7 @@ 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 +638,11 @@ 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-rc2-mm2/mm/swap_state.c~mem-control-accounting 2007-08-17
13:14:19.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/swap_state.c 2007-08-17 13:14:19.000000000 +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>

@@ -80,6 +81,11 @@ 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;
}

@@ -132,6 +141,7 @@ 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 (v6)
 Posted by [Balbir Singh](#) on Fri, 17 Aug 2007 08:43:20 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-rc2-mm2/mm/memcontrol.c~mem-control-task-migration 2007-08-17
13:14:19.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/memcontrol.c 2007-08-17 13:14:19.000000000 +0530
@@ -326,11 +326,46 @@ 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 (v6)

Posted by [Balbir Singh](#) on Fri, 17 Aug 2007 08:43:30 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 page_container 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 | 12 +++
include/linux/res_counter.h | 23 +++++++
include/linux/swap.h       | 3
mm/memcontrol.c            | 135 ++++++++++++++++++++++++++++++++++++++
mm/swap.c                  | 2
mm/vmscan.c                | 126 +++++++++++++++++++++++++++++++++-----
```

6 files changed, 275 insertions(+), 26 deletions(-)

```
diff -puN include/linux/memcontrol.h~mem-control-lru-and-reclaim include/linux/memcontrol.h
--- linux-2.6.23-rc2-mm2/include/linux/memcontrol.h~mem-control-lru-and-reclaim 2007-08-17
13:14:20.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/include/linux/memcontrol.h 2007-08-17 13:14:20.000000000
+0530
@@ -32,6 +32,13 @@ extern void page_assign_page_container(s
extern struct page_container *page_get_page_container(struct page *page);
extern int mem_container_charge(struct page *page, struct mm_struct *mm);
extern void mem_container_uncharge(struct page_container *pc);
+extern void mem_container_move_lists(struct page_container *pc, 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)
{
@@ -71,6 +78,11 @@ static inline void mem_container_uncharg
{
}

+static inline void mem_container_move_lists(struct page_container *pc,
+ 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-rc2-mm2/include/linux/res_counter.h~mem-control-lru-and-reclaim 2007-08-17
13:14:20.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/include/linux/res_counter.h 2007-08-17 13:14:20.000000000
+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-rc2-mm2/include/linux/swap.h~mem-control-lru-and-reclaim 2007-08-17
13:14:20.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/include/linux/swap.h 2007-08-17 13:14:20.000000000 +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-rc2-mm2/mm/memcontrol.c~mem-control-lru-and-reclaim 2007-08-17
13:14:20.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/memcontrol.c 2007-08-17 13:14:20.000000000 +0530
@@ -24,8 +24,12 @@
#include <linux/page-flags.h>
#include <linux/bit_spinlock.h>
#include <linux/rcupdate.h>
+#include <linux/swap.h>

```

```

#include <linux/spinlock.h>
#include <linux/fs.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 +55,10 @@ 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;
};

/*
@@ -141,6 +149,94 @@ void __always_inline unlock_page_contain
bit_spin_unlock(PAGE_CONTAINER_LOCK_BIT, &page->page_container);
}

+void __mem_container_move_lists(struct page_container *pc, bool active)
+{
+ if (active)
+ list_move(&pc->lru, &pc->mem_container->active_list);
+ else
+ list_move(&pc->lru, &pc->mem_container->inactive_list);
+}
+
+/*
+ * This routine assumes that the appropriate zone's lru lock is already held
+ */
+void mem_container_move_lists(struct page_container *pc, bool active)
+{
+ struct mem_container *mem;
+ if (!pc)
+ return;
+
+ mem = pc->mem_container;
+
+ spin_lock(&mem->lru_lock);
+ __mem_container_move_lists(pc, active);
+ spin_unlock(&mem->lru_lock);
+}
+
+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(pc_list);
+ struct list_head *src;
+ struct page_container *pc;
+
+ if (active)
+ src = &mem_cont->active_list;
+ else
+ src = &mem_cont->inactive_list;
+
+ spin_lock(&mem_cont->lru_lock);
+ for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
+ pc = list_entry(src->prev, struct page_container, lru);
+ page = pc->page;
+ VM_BUG_ON(!pc);
+
+ if (PageActive(page) && !active) {
+ __mem_container_move_lists(pc, true);
+ scan--;
+ continue;
+ }
+ if (!PageActive(page) && active) {
+ __mem_container_move_lists(pc, false);
+ scan--;
+ continue;
+ }
+
+ /*
+  * Reclaim, per zone
+  * TODO: make the active/inactive lists per zone
+  */
+ if (page_zone(page) != z)
+ continue;
+
+ /*
+  * Check if the meta page went away from under us
+  */
+ if (!list_empty(&pc->lru))
+ list_move(&pc->lru, &pc_list);
+ else

```

```

+ continue;
+
+ if (__isolate_lru_page(page, mode) == 0) {
+ list_move(&page->lru, dst);
+ nr_taken++;
+ }
+ }
+
+ list_splice(&pc_list, src);
+ spin_unlock(&mem_cont->lru_lock);
+
+ *scanned = scan;
+ return nr_taken;
+}
+
+/*
+ * Charge the memory controller for page usage.
+ * Return
@@ -151,6 +247,8 @@ int mem_container_charge(struct page *pa
{
    struct mem_container *mem;
    struct page_container *pc, *race_pc;
+ unsigned long flags;
+ unsigned long nr_retries = MEM_CONTAINER_RECLAIM_RETRIES;

    /*
     * Should page_container's go to their own slab?
@@ -197,7 +295,32 @@ int mem_container_charge(struct page *pa
    * If we created the page_container, 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_pc;
+ }
@@ -221,6 +344,10 @@ int mem_container_charge(struct page *pa
+     pc->page = page;
+     page_assign_page_container(page, pc);

+ spin_lock_irqsave(&mem->lru_lock, flags);
+ list_add(&pc->lru, &mem->active_list);
+ spin_unlock_irqrestore(&mem->lru_lock, flags);
+
done:
+     unlock_page_container(page);
+     return 0;
@@ -240,6 +367,7 @@ void mem_container_uncharge(struct page_
+ {
+     struct mem_container *mem;
+     struct page *page;
+ unsigned long flags;

+     if (!pc)
+         return;
@@ -252,6 +380,10 @@ void mem_container_uncharge(struct page_
+     page_assign_page_container(page, NULL);
+     unlock_page_container(page);
+     res_counter_uncharge(&mem->res, 1);
+
+
+ spin_lock_irqsave(&mem->lru_lock, flags);
+ list_del_init(&pc->lru);
+ spin_unlock_irqrestore(&mem->lru_lock, flags);
+     kfree(pc);
+ }
+ }
@@ -310,6 +442,7 @@ 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-rc2-mm2/mm/swap.c~mem-control-lru-and-reclaim 2007-08-17
13:14:20.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/swap.c 2007-08-17 13:14:20.000000000 +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_page_container(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-rc2-mm2/mm/vmscan.c~mem-control-lru-and-reclaim 2007-08-17
13:14:20.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/vmscan.c 2007-08-17 13:14:20.000000000 +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))
@@ -634,7 +644,7 @@ 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;

@@ -768,6 +778,21 @@ 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.
@@ -809,11 +834,11 @@ 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);
    __count_vm_events(PGDEACTIVATE, nr_active);

```

@@ -1026,8 +1051,9 @@ 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);
```

@@ -1062,6 +1088,7 @@ force_reclaim_mapped:

```
ClearPageActive(page);

list_move(&page->lru, &zone->inactive_list);
+ mem_container_move_lists(page_get_page_container(page), false);
pgmoved++;
if (!pagevec_add(&pvec, page)) {
__mod_zone_page_state(zone, NR_INACTIVE, pgmoved);
```

@@ -1090,6 +1117,7 @@ force_reclaim_mapped:

```
SetPageLRU(page);
VM_BUG_ON(!PageActive(page));
list_move(&page->lru, &zone->active_list);
+ mem_container_move_lists(page_get_page_container(page), true);
pgmoved++;
if (!pagevec_add(&pvec, page)) {
__mod_zone_page_state(zone, NR_ACTIVE, pgmoved);
```

@@ -1221,7 +1249,8 @@ 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;
```

@@ -1230,14 +1259,6 @@ 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_writpage = !laptop_mode,
-   .swap_cluster_max = SWAP_CLUSTER_MAX,
-   .may_swap = 1,
```

```

- .swappiness = vm_swappiness,
- .order = order,
- };

delay_swap_prefetch();
count_vm_event(ALLOCSTALL);
@@ -1253,17 +1274,22 @@ 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;
}
@@ -1275,18 +1301,18 @@ 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;
}

/* Take a nap, wait for some writeback to complete */

```

```

- if (sc.nr_scanned && priority < DEF_PRIORITY - 2)
+ if (sc->nr_scanned && priority < DEF_PRIORITY - 2)
    congestion_wait(WRITE, HZ/10);
}
/* top priority shrink_caches still had more to do? don't OOM, then */
- if (!sc.all_unreclaimable)
+ if (!sc->all_unreclaimable && sc->mem_container == NULL)
    ret = 1;
out:
/*
@@ -1309,6 +1335,54 @@ out:
    return ret;
}

+unsigned long try_to_free_pages(struct zone **zones, int order, gfp_t gfp_mask)
+{
+ struct scan_control sc = {
+  .gfp_mask = gfp_mask,
+  .may_writepage = !laptop_mode,
+  .swap_cluster_max = SWAP_CLUSTER_MAX,
+  .may_swap = 1,
+  .swappiness = vm_swappiness,
+  .order = order,
+  .mem_container = NULL,
+  .isolate_pages = isolate_pages_global,
+ };
+
+ return do_try_to_free_pages(zones, gfp_mask, &sc);
+}
+
+#ifdef CONFIG_CONTAINER_MEM_CONT
+
+#ifdef CONFIG_HIGHMEM
+#define ZONE_USERPAGES ZONE_HIGHMEM
+#else
+#define ZONE_USERPAGES ZONE_NORMAL
+#endif
+
+unsigned long try_to_free_mem_container_pages(struct mem_container *mem_cont)
+{
+ struct scan_control sc = {
+  .gfp_mask = GFP_KERNEL,
+  .may_writepage = !laptop_mode,
+  .may_swap = 1,
+  .swap_cluster_max = SWAP_CLUSTER_MAX,
+  .swappiness = vm_swappiness,
+  .order = 1,
+  .mem_container = mem_cont,

```

```

+ .isolate_pages = mem_container_isolate_pages,
+ };
+ int node;
+ struct zone **zones;
+
+ for_each_online_node(node) {
+   zones = NODE_DATA(node)->node_zonelists[ZONE_USERPAGES].zones;
+   if (do_try_to_free_pages(zones, sc.gfp_mask, &sc))
+     return 1;
+ }
+ return 0;
+}
+
+/*
+ * For kswapd, balance_pgdat() will work across all this node's zones until
+ * they are all at pages_high.
+@@ -1344,6 +1418,8 @@ static unsigned long balance_pgdat(pg_da
+   .swap_cluster_max = SWAP_CLUSTER_MAX,
+   .swappiness = vm_swappiness,
+   .order = order,
+ .mem_container = NULL,
+ .isolate_pages = isolate_pages_global,
+ };
+
+ * temp_priority is used to remember the scanning priority at which

```

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 7/9] Memory controller OOM handling (v6)
 Posted by [Balbir Singh](#) on Fri, 17 Aug 2007 08:43:41 GMT
[View Forum Message](#) <> [Reply to Message](#)

From: Pavel Emelianov <xemul@openvz.org>

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-rc2-mm2/include/linux/memcontrol.h~mem-control-out-of-memory 2007-08-17
13:14:20.000000000 +0530
```

```
+++ linux-2.6.23-rc2-mm2-balbir/include/linux/memcontrol.h 2007-08-17 13:14:20.000000000
+0530
```

```
@@ -39,6 +39,7 @@ 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-rc2-mm2/mm/memcontrol.c~mem-control-out-of-memory 2007-08-17
13:14:20.000000000 +0530
```

```
+++ linux-2.6.23-rc2-mm2-balbir/mm/memcontrol.c 2007-08-17 13:14:20.000000000 +0530
```

```
@@ -322,6 +322,7 @@ int mem_container_charge(struct page *pa
}
```

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

```
diff -puN mm/oom_kill.c~mem-control-out-of-memory mm/oom_kill.c
```

```
--- linux-2.6.23-rc2-mm2/mm/oom_kill.c~mem-control-out-of-memory 2007-08-17
13:14:20.000000000 +0530
```

```
+++ linux-2.6.23-rc2-mm2-balbir/mm/oom_kill.c 2007-08-17 13:14:20.000000000 +0530
```

```
@@ -25,6 +25,7 @@
```

```
#include <linux/cpuset.h>
```

```
#include <linux/module.h>
```

```
#include <linux/notifier.h>
```

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

```

int sysctl_panic_on_oom;
/* #define DEBUG */
@@ -48,7 +49,8 @@ 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;
@@ -61,6 +63,13 @@ 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.
 */
@@ -198,7 +207,8 @@ 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;
@@ -252,7 +262,7 @@ 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;
@@ -364,6 +374,30 @@ 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)
@@ -436,7 +470,7 @@ 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

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

Subject: [-mm PATCH 8/9] Memory controller add switch to control what type of pages to limit (v6)

Posted by [Balbir Singh](#) on Fri, 17 Aug 2007 08:43:51 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 3 > 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, 103 insertions(+), 2 deletions(-)
```

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

```
linux-2.6.23-rc2-mm2/include/linux/memcontrol.h~mem-control-choose-rss-vs-rss-and-pagecache
2007-08-17 13:14:20.000000000 +0530
```

```
+++ linux-2.6.23-rc2-mm2-balbir/include/linux/memcontrol.h 2007-08-17 13:14:20.000000000
+0530
```

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

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

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

```
+
```

```
struct mem_container;
struct page_container;
```

```
@@ -40,6 +42,7 @@ 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)
{
@@ -84,6 +87,12 @@ 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-rc2-mm2/mm/filemap.c~mem-control-choose-rss-vs-rss-and-pagecache 2007-08-17
13:14:20.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/filemap.c 2007-08-17 13:14:20.000000000 +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-rc2-mm2/mm/memcontrol.c~mem-control-choose-rss-vs-rss-and-pagecache 2007-08-
17 13:14:20.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/memcontrol.c 2007-08-17 13:14:20.000000000 +0530
@@ -28,6 +28,8 @@
#include <linux/spinlock.h>
#include <linux/fs.h>

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

@@ -59,6 +61,7 @@ struct mem_container {
* spin_lock to protect the per container LRU
*/

```

```

    spinlock_t lru_lock;
+ unsigned long control_type; /* control RSS or RSS+Pagecache */
};

/*
@@ -81,6 +84,15 @@ struct page_container {
    /* 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)
@@ -361,6 +373,22 @@ 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.
+ */
@@ -370,6 +398,10 @@ void mem_container_uncharge(struct page_
    struct page *page;
    unsigned long flags;

```

```

+ /*
+  * This can handle cases when a page is not charged at all and we
+  * are switching between handling the control_type.
+  */
+ if (!pc)
+   return;

@@ -405,6 +437,60 @@ 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",
@@ -422,6 +508,11 @@ 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;
@@ -444,6 +535,7 @@ 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-rc2-mm2/mm/swap_state.c~mem-control-choose-rss-vs-rss-and-pagecache 2007-08-17 13:14:20.000000000 +0530

+++ linux-2.6.23-rc2-mm2-balbir/mm/swap_state.c 2007-08-17 13:14:20.000000000 +0530

```

@@ -82,7 +82,7 @@ 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 (v6)

Posted by [Balbir Singh](#) on Fri, 17 Aug 2007 08:44:01 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-rc2-mm2/include/linux/rmap.h~mem-control-per-container-page-referenced 2007-08-17 13:14:20.000000000 +0530

+++ linux-2.6.23-rc2-mm2-balbir/include/linux/rmap.h 2007-08-17 13:14:20.000000000 +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-rc2-mm2/mm/rmap.c~mem-control-per-container-page-referenced 2007-08-17
13:14:20.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/rmap.c 2007-08-17 13:14:20.000000000 +0530
@@ -299,7 +299,8 @@ 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;
@@ -312,6 +313,13 @@ 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;
@@ -332,7 +340,8 @@ 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;
@@ -365,6 +374,13 @@ 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++;
@@ -387,7 +403,8 @@ 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;

@@ -399,14 +416,15 @@ 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-rc2-mm2/mm/vmscan.c~mem-control-per-container-page-referenced 2007-08-17
13:14:20.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/vmscan.c 2007-08-17 13:14:20.000000000 +0530
@@ -511,7 +511,7 @@ static unsigned long shrink_page_list(st
    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))
@@ -1065,7 +1065,7 @@ 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-rc2-mm2/include/linux/memcontrol.h~mem-control-per-container-page-referenced 20
07-08-17 13:14:20.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/include/linux/memcontrol.h 2007-08-17 13:14:20.000000000
+0530
@@ -43,6 +43,7 @@ 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)
{
@@ -93,6 +94,11 @@ 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-rc2-mm2/mm/memcontrol.c~mem-control-per-container-page-referenced 2007-08-17
13:14:20.000000000 +0530
+++ linux-2.6.23-rc2-mm2-balbir/mm/memcontrol.c 2007-08-17 13:14:20.000000000 +0530
@@ -109,6 +109,11 @@ 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 0/9] Memory controller introduction (v6)
Posted by [Dhaval Giani](#) on Fri, 17 Aug 2007 15:49:51 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Fri, Aug 17, 2007 at 02:12:28PM +0530, Balbir Singh wrote:
Hi Andrew,

> 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.
>
> Dhaval ran several tests on v6 and gave his thumbs up to the controller
> (a hard to achieve goal :-)).
>

I've been running v6 on x86 and there are a few things I've tried. Most of those results Balbir has already posted.

> Run kernbench stress
> -----
> Three simultaneously and with one inside a container of 800 MB.
>

The idea here was to create pressure inside the container while having global pressure.

>
> Kernbench results running within the container of 800 MB
> -----
>
> Thu Aug 16 22:34:59 IST 2007
> 2.6.23-rc2-mm2-mem-v6
> Average Half load -j 4 Run (std deviation):
> Elapsed Time 466.548 (47.6014)
> User Time 876.598 (10.5273)
> System Time 223.136 (1.29247)
> Percent CPU 237.2 (23.2744)
> Context Switches 146351 (6539.91)
> Sleeps 174003 (5031.94)
>
> Average Optimal load -j 32 Run (std deviation):
> Elapsed Time 423.496 (60.625)
> User Time 897.285 (23.0391)
> System Time 228.836 (6.11205)
> Percent CPU 257.1 (40.9022)
> Context Switches 262134 (123397)
> Sleeps 270815 (103597)
>
>
> Kernbench results running within the default container
> -----
>
> Thu Aug 16 22:34:33 IST 2007
> 2.6.23-rc2-mm2-mem-v6
> Average Half load -j 4 Run (std deviation):

```

> Elapsed Time 424.17 (3.45908)
> User Time 841.992 (5.40178)
> System Time 213.01 (0.706258)
> Percent CPU 248.2 (0.83666)
> Context Switches 134254 (9535.83)
> Sleeps 167359 (6858.45)
>
> Average Optimal load -j 32 Run (std deviation):
> Elapsed Time 407.092 (108.932)
> User Time 878.493 (38.6575)
> System Time 222.155 (9.77127)
> Percent CPU 278.4 (91.5826)
> Context Switches 253836 (127708)
> Sleeps 263760 (103468)
>
>
> Kernbench results running within the default container
> -----
>
> Thu Aug 16 22:34:52 IST 2007
> 2.6.23-rc2-mm2-mem-v6
> Average Half load -j 4 Run (std deviation):
> Elapsed Time 465.038 (48.5147)
> User Time 874.742 (5.86563)
> System Time 222.194 (0.561676)
> Percent CPU 237.2 (22.5211)
> Context Switches 144040 (7052.23)
> Sleeps 172130 (5608.73)
>
> Average Optimal load -j 32 Run (std deviation):
> Elapsed Time 426.25 (62.34)
> User Time 893.938 (20.6732)
> System Time 227.717 (5.87502)
> Percent CPU 255.6 (40.7163)
> Context Switches 259560 (122953)
> Sleeps 267402 (101801)
>

```

I've been running v6 for sometime and it has been stable (still not been able to break/crash it, whereas v4 was broken within 1 hour :-))

Andrew, I feel these patches whould be included in mm so that it gets wider testing.

--
 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 1/9] Memory controller resource counters (v6)

Posted by [Alexey Dobriyan](#) on Mon, 20 Aug 2007 08:20:54 GMT

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

On Fri, Aug 17, 2007 at 02:12:38PM +0530, Balbir Singh wrote:

```
> --- /dev/null
> +++ linux-2.6.23-rc2-mm2-balbir/kernel/res_counter.c
> +void res_counter_init(struct res_counter *counter)
> +{
> + spin_lock_init(&counter->lock);
> + counter->limit = (unsigned long)LONG_MAX;
```

why cast?

```
> +int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
> +{
> + if (counter->usage > (counter->limit - val)) {
```

() aren't needed.

```
> + 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;
```

explicit if and WARN_ON(1) is clearer. I should send a patch banning such type of usage soon.

```
> +
> + 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);
> +}
```

```
> +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);
```

please, switch to fixed buffer, allocating memory depending on size told by userspace will be later. Ditto for other proc writing functions.

```
> + 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;
> +}
```

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

Subject: Re: [-mm PATCH 1/9] Memory controller resource counters (v6)
Posted by [Balbir Singh](#) on Mon, 20 Aug 2007 09:01:51 GMT
[View Forum Message](#) <> [Reply to Message](#)

Alexey Dobriyan wrote:

```
> On Fri, Aug 17, 2007 at 02:12:38PM +0530, Balbir Singh wrote:
>> --- /dev/null
>> +++ linux-2.6.23-rc2-mm2-balbir/kernel/res_counter.c
>> +void res_counter_init(struct res_counter *counter)
>> +{
>> + spin_lock_init(&counter->lock);
>> + counter->limit = (unsigned long)LONG_MAX;
>
> why cast?
>
```

These patches come from Pavel. They add to readability since limit is unsigned long.

```
>> +int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
>> +{
>> + if (counter->usage > (counter->limit - val)) {
>
> () aren't needed.
>
```

it makes the code more readable

```
>> + if (WARN_ON(counter->usage < val))
>> + val = counter->usage;
>
> explicit if and WARN_ON(1) is clearer. I should send a patch banning such
> type of usage soon.
>
```

We had a WARN_ON(1) before, but we changed it in v2 or v3 based on review comments from Dave. I think WARN_ON(cond) is more readable than WARN_ON(1) for the same reason as BUG_ON(cond) vs BUG_ON(1)

```
>> + buf = kmalloc(nbytes + 1, GFP_KERNEL);  
>  
> please, switch to fixed buffer, allocating memory depending on size  
> told by userspace will be later. Ditto for other proc writing  
> functions.  
>
```

I agree with you in part, but the size of user input is not fixed.
Setting a fixed limit seems artificial, I'll see how this can be improved.

Thanks for the detailed review comments,

--

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 1/9] Memory controller resource counters (v6)
Posted by [Paul Menage](#) on Tue, 11 Sep 2007 00:42:20 GMT
[View Forum Message](#) <> [Reply to Message](#)

Hi Balbir/Pavel,

As I mentioned to you directly at the kernel summit, I think it might be cleaner to integrate resource counters more closely with control groups. So rather than controllers such as the memory controller having to create their own boilerplate cf_type structures and read/write functions, it should be possible to just call a function something like

```
control_group_add_rescounter(struct cgroup *cg, struct cgroup_subsys *ss,  
                           struct res_counter *res,  
                           const char *name)
```

and have it handle all the userspace API. This would simplify the task of keeping a consistent userspace API between different controllers using the resource counter abstraction.

Paul

On 8/17/07, Balbir Singh <balbir@linux.vnet.ibm.com> wrote:

```
> 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-rc2-mm2-balbir/include/linux/res_counter.h    2007-08-17 13:14:18.000000000
> +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-rc2-mm2/init/Kconfig~res_counters_infra      2007-08-17 13:14:18.000000000
+0530
> +++ linux-2.6.23-rc2-mm2-balbir/init/Kconfig  2007-08-17 13:14:18.000000000 +0530
> @@ -337,6 +337,13 @@ 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-rc2-mm2/kernel/Makefile~res_counters_infra 2007-08-17 13:14:18.000000000
+0530
> +++ linux-2.6.23-rc2-mm2-balbir/kernel/Makefile 2007-08-17 13:14:18.000000000 +0530
> @@ -58,6 +58,7 @@ 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-rc2-mm2-balbir/kernel/res_counter.c 2007-08-17 13:14:18.000000000 +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
>
> --
> To unsubscribe, send a message with 'unsubscribe linux-mm' in
> the body to majordomo@kvack.org. For more info on Linux MM,
> see: <http://www.linux-mm.org/> .
> Don't email: <[a href=mailto:"dont@kvack.org"> email@kvack.org](mailto:dont@kvack.org)>
>

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

Subject: Re: [-mm PATCH 1/9] Memory controller resource counters (v6)
Posted by [Balbir Singh](#) on Wed, 12 Sep 2007 10:05:14 GMT
[View Forum Message](#) <> [Reply to Message](#)

Paul Menage wrote:

> Hi Balbir/Pavel,
>
> As I mentioned to you directly at the kernel summit, I think it might
> be cleaner to integrate resource counters more closely with control
> groups. So rather than controllers such as the memory controller
> having to create their own boilerplate cf_type structures and
> read/write functions, it should be possible to just call a function
> something like
>
> control_group_add_rescounter(struct cgroup *cg, struct cgroup_subsys *ss,
> struct res_counter *res,
> const char *name)
>
> and have it handle all the userspace API. This would simplify the task
> of keeping a consistent userspace API between different controllers
> using the resource counter abstraction.
>
> Paul
>

Yes, I remember discussing it with you. I would expect res_counters
definition to be dynamic (to be able to add the guarantee, soft limit,
etc) for expansion in the future. In the future, I would also like
to do hierarchical resource groups, the hierarchy would represent
the current filesystem hierarchy.

--

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>
