Subject: [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by dev on Tue, 05 Sep 2006 14:59:30 GMT
View Forum Message <> Reply to Message

Core Resource Beancounters (BC) + kernel/user memory control.

BC allows to account and control consumption
of kernel resources used by group of processes.

Draft UBC description on OpenVZ wiki can be found at
http://wiki.openvz.org/UBC_parameters

The full BC patch set allows to control:
- kernel memory. All the kernel objects allocatable
on user demand should be accounted and limited
for DoS protection.
E.g. page tables, task structs, vmas etc.

- virtual memory pages. BCs allow to
limit a container to some amount of memory and
introduces 2-level OOM killer taking into account
container's consumption.
pages shared between containers are correctly
charged as fractions (tunable).

- network buffers. These includes TCP/IP rcv/snd
buffers, dgram snd buffers, unix, netlinks and
other buffers.

- minor resources accounted/limited by number:
tasks, files, flocks, ptys, siginfo, pinned dcache
mem, sockets, iptentries (for containers with
virtualized networking)

As the first step we want to propose for discussion
the most complicated parts of resource management:
kernel memory and virtual memory.
The patch set to be sent provides core for BC and
management of kernel memory only. Virtual memory
management will be sent in a couple of days.

The patches in these series are:
diff-atomic-dec-and-lock-irqsave.patch
 introduce atomic_dec_and_lock_irqsave()

diff-bc-kconfig.patch:
 Adds kernel/bc/Kconfig file with UBC options and
 includes it into arch Kconfigs

diff-bc-core.patch:
 Contains core functionality and interfaces of BC:
 find/create beancounter, initialization,
 charge/uncharge of resource, core objects' declarations.

diff-bc-task.patch:
 Contains code responsible for setting BC on task,
 it's inheriting and setting host context in interrupts.

 Task contains three beancounters:
 1. exec_bc - current context. all resources are charged
             to this beancounter.
 2. fork_bc - beancounter which is inherited by
             task's children on fork

diff-bc-syscalls.patch:
 Patch adds system calls for BC management:
 1. sys_get_bcid   - get current BC id
 2. sys_set_bcid   - changes exec_ and fork_ BCs on current
 3. sys_set_bclimit - set limits for resources consumtions
 4. sys_get_bcstat - returns limits/usages/fails for BC

diff-bc-kmem-core.patch:
 Introduces BC_KMEMSIZE resource which accounts kernel
 objects allocated by task's request.

 Objects are accounted via struct page and slab objects.
 For the latter ones each slab contains a set of pointers
 corresponding object is charged to.

 Allocation charge rules:
 1. Pages - if allocation is performed with __GFP_BC flag - page
    is charged to current's exec_bc.
 2. Slabs - kmem_cache may be created with SLAB_BC flag - in this
    case each allocation is charged. Caches used by kmalloc are
    created with SLAB_BC | SLAB_BC_NOCHARGE flags. In this case
    only __GFP_BC allocations are charged.

diff-bc-kmem-charge.patch:
 Adds SLAB_BC and __GFP_BC flags in appropriate places
 to cause charging/limiting of specified resources.

diff-bc-vmlocked-core.patch:
Introduces new resource BC_LOCKEDPAGES for accounting
of mlock-ed user pages.

diff-bc-vmlocked-charge.patch:

Places calls to BC core over the kernel to charge locked memory.

diff-bc-privvm.patch:
This patch instroduces new resource - BC_PRIVVMPAGES.
Privvmpages acointing is described in details in
http://wiki.openvz.org/User_pages_accounting

diff-bc-vmrss-prep.patch:
This patch intruduces small preparations for vmrss accounting
to make reviewing simpler.

diff-bc-vmrss-core.patch:
This is the core of vmrss accounting.
Pages are accounted in fractions and it is described in details in
http://wiki.openvz.org/RSS_fractions_accounting

diff-bc-vmrss-charge.patch:
Calls to vmrss core code over the kernel to do accounting.

Summary of changes from v3 patch set:

* Added basic user pages accounting (lockedpages/privvmpages)
* spell in Kconfig
* Makefile reworked
* EXPORT_SYMBOL_GPL
* union w/o name in struct page
* bc_task_charge is void now
* adjust minheld/maxheld splitted

Summary of changes from v2 patch set:

* introduced atomic_dec_and_lock_irqsave()
* bc_adjust_held_minmax comment
* added __must_check for bc_*charge* funcs
* use hash_long() instead of own one
* bc/Kconfig is sourced from init/Kconfig now
* introduced bcid_t type with comment from Alan Cox
* check for barrier <= limit in sys_set_bclimit()
* removed (bc == NULL) checks
* replaced memcpy in beancounter_findcrate with assignment
* moved check 'if (mask & BC_ALLOC)' out of the lock
* removed unnecessary memset()

Summary of changes from v1 patch set:

* CONFIG_BEANCOUNTERS is 'n' by default
* fixed Kconfig includes in arches

* removed hierarchical beancounters to simplify first patchset
* removed unused 'private' pointer
* removed unused EXPORTS
* MAXVALUE redeclared as LONG_MAX
* beancounter_findcreate clarification
* renamed UBC -> BC, ub -> bc etc.
* moved BC inheritance into copy_process
* introduced reset_exec_bc() with proposed BUG_ON
* removed task_bc beancounter (not used yet, for numproc)
* fixed syscalls for sparc
* added sys_get_bcstat(): return info that was in /proc
* cond_syscall instead of #ifdefs

Many thanks to Oleg Nesterov, Alan Cox, Matt Helsley and others
for patch review and comments.

Patch set is applicable to 2.6.18-rc5-mm1

Thanks,
Kirill

Subject: [PATCH 1/13] BC: introduce atomic_dec_and_lock_irqsave()
Posted by dev on Tue, 05 Sep 2006 15:16:08 GMT
View Forum Message <> Reply to Message

Oleg Nesterov noticed to me that the construction like
(used in beancounter patches and free_uid()):

```
local_irq_save(flags);
if (atomic_dec_and_lock(&refcnt, &lock))
 ...
```

is not that good for preemtible kernels, since with preemption
spin_lock() can schedule() to reduce latency. However, it won't schedule
if interrupts are disabled.

So this patch introduces atomic_dec_and_lock_irqsave() as a logical
counterpart to atomic_dec_and_lock().

Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
Signed-Off-By: Kirill Korotaev <dev@sw.ru>

```
include/linux/spinlock.h |    6 ++++++
kernel/user.c            |    5 +----
lib/dec_and_lock.c       |   19 +++++++++++++++++++
```

```
 3 files changed, 26 insertions(+), 4 deletions(-)

--- ./include/linux/spinlock.h.dlirq 2006-08-28 10:17:35.000000000 +0400
+++ ./include/linux/spinlock.h 2006-08-28 11:22:37.000000000 +0400
@@ -266,6 +266,12 @@ extern int _atomic_dec_and_lock(atomic_t
 #define atomic_dec_and_lock(atomic, lock) \
   __cond_lock(lock, _atomic_dec_and_lock(atomic, lock))

+extern int _atomic_dec_and_lock_irqsave(atomic_t *atomic, spinlock_t *lock,
+  unsigned long *flagsp);
+#define atomic_dec_and_lock_irqsave(atomic, lock, flags) \
+  __cond_lock(lock, \
+   _atomic_dec_and_lock_irqsave(atomic, lock, &flags))
+
 /**
  * spin_can_lock - would spin_trylock() succeed?
  * @lock: the spinlock in question.
--- ./kernel/user.c.dlirq 2006-07-10 12:39:20.000000000 +0400
+++ ./kernel/user.c 2006-08-28 11:08:56.000000000 +0400
@@ -108,15 +108,12 @@ void free_uid(struct user_struct *up)
  if (!up)
   return;

- local_irq_save(flags);
- if (atomic_dec_and_lock(&up->__count, &uidhash_lock)) {
+ if (atomic_dec_and_lock_irqsave(&up->__count, &uidhash_lock, flags)) {
   uid_hash_remove(up);
   spin_unlock_irqrestore(&uidhash_lock, flags);
   key_put(up->uid_keyring);
   key_put(up->session_keyring);
   kmem_cache_free(uid_cachep, up);
- } else {
-  local_irq_restore(flags);
  }
 }

--- ./lib/dec_and_lock.c.dlirq 2006-04-21 11:59:36.000000000 +0400
+++ ./lib/dec_and_lock.c 2006-08-28 11:22:08.000000000 +0400
@@ -33,3 +33,22 @@ int _atomic_dec_and_lock(atomic_t *atomi
 }

 EXPORT_SYMBOL(_atomic_dec_and_lock);
+
+/*
+ * the same, but takes the lock with _irqsave
+ */
+int _atomic_dec_and_lock_irqsave(atomic_t *atomic, spinlock_t *lock,
+  unsigned long *flagsp)
```

```
+{
+#ifdef CONFIG_SMP
+ if (atomic_add_unless(atomic, -1, 1))
+  return 0;
+#endif
+ spin_lock_irqsave(lock, *flagsp);
+ if (atomic_dec_and_test(atomic))
+  return 1;
+ spin_unlock_irqrestore(lock, *flagsp);
+ return 0;
+}
+
+EXPORT_SYMBOL(_atomic_dec_and_lock_irqsave);
```

Subject: [PATCH 2/13] BC: kconfig
Posted by dev on Tue, 05 Sep 2006 15:16:34 GMT
View Forum Message <> Reply to Message

Add kernel/bc/Kconfig file with BC options and
include it into arch Kconfigs

Signed-off-by: Pavel Emelianov <xemul@sw.ru>
Signed-off-by: Kirill Korotaev <dev@sw.ru>

```
 init/Kconfig      |    2 ++
 kernel/bc/Kconfig |   25 +++++++++++++++++++++++++
 2 files changed, 27 insertions(+)

--- ./init/Kconfig.bckm 2006-07-10 12:39:10.000000000 +0400
+++ ./init/Kconfig 2006-07-28 14:10:41.000000000 +0400
@@ -222,6 +222,8 @@ source "crypto/Kconfig"

    Say N if unsure.

+source "kernel/bc/Kconfig"
+
 config SYSCTL
   bool

--- ./kernel/bc/Kconfig.bckconf 2006-09-05 12:21:09.000000000 +0400
+++ ./kernel/bc/Kconfig 2006-09-05 12:19:54.000000000 +0400
@@ -0,0 +1,25 @@
+#
+# Resource beancounters (BC)
+#
```

```
+# Copyright (C) 2006 OpenVZ. SWsoft Inc
+
+menu "User resources"
+
+config BEANCOUNTERS
+ bool "Enable resource accounting/control"
+ default n
+ help
+         When Y this option provides accounting and allows configuring
+         limits for user's consumption of exhaustible system resources.
+         The most important resource controlled by this patch is unswappable
+         memory (either mlock'ed or used by internal kernel structures and
+         buffers). The main goal of this patch is to protect processes
+         from running short of important resources because of accidental
+         misbehavior of processes or malicious activity aiming to ``kill''
+         the system. It's worth mentioning that resource limits configured
+         by setrlimit(2) do not give an acceptable level of protection
+         because they cover only a small fraction of resources and work on a
+         per-process basis.  Per-process accounting doesn't prevent malicious
+         users from spawning a lot of resource-consuming processes.
+
+endmenu
```

Subject: [PATCH 3/17] BC: beancounters core (API)

Posted by dev on Tue, 05 Sep 2006 15:17:35 GMT

View Forum Message <> Reply to Message

Core functionality and interfaces of BC:
find/create beancounter, initialization,
charge/uncharge of resource, core objects' declarations.

Basic structures:
 bc_resource_parm - resource description
 beancounter   - set of resources, id, lock

Signed-off-by: Pavel Emelianov <xemul@sw.ru>
Signed-off-by: Kirill Korotaev <dev@sw.ru>

```
 include/bc/beancounter.h |  155 ++++++++++++++++++++++++++++++++++++++++++++++
 include/linux/types.h    |   16 ++
 init/main.c              |    4
 kernel/Makefile          |    1
 kernel/bc/Makefile       |    7 +
 kernel/bc/beancounter.c  |  263 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 6 files changed, 446 insertions(+)
```

```
--- ./include/bc/beancounter.h.bccore 2006-09-05 12:06:35.000000000 +0400
+++ ./include/bc/beancounter.h 2006-09-05 12:15:57.000000000 +0400
@@ -0,0 +1,155 @@
+/*
+ *  include/bc/beancounter.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef _LINUX_BEANCOUNTER_H
+#define _LINUX_BEANCOUNTER_H
+
+/*
+ * Resource list.
+ */
+
+#define BC_RESOURCES 0
+
+struct bc_resource_parm {
+ unsigned long barrier; /* A barrier over which resource allocations
+     * are failed gracefully. e.g. if the amount
+     * of consumed memory is over the barrier
+     * further sbrk() or mmap() calls fail, the
+     * existing processes are not killed.
+     */
+ unsigned long limit; /* hard resource limit */
+ unsigned long held; /* consumed resources */
+ unsigned long maxheld; /* maximum amount of consumed resources */
+ unsigned long minheld; /* minumum amount of consumed resources */
+ unsigned long failcnt; /* count of failed charges */
+};
+
+/*
+ * Kernel internal part.
+ */
+
+#ifdef __KERNEL__
+
+#include <linux/spinlock.h>
+#include <linux/list.h>
+#include <asm/atomic.h>
+
+#define BC_MAXVALUE LONG_MAX
+
+/*
+ * Resource management structures
```

```
+ * Serialization issues:
+ *   beancounter list management is protected via bc_hash_lock
+ *   task pointers are set only for current task and only once
+ *   refcount is managed atomically
+ *   value and limit comparison and change are protected by per-bc spinlock
+ */
+
+struct beancounter {
+ atomic_t  bc_refcount;
+ spinlock_t  bc_lock;
+ bcid_t   bc_id;
+ struct hlist_node hash;
+
+ /* resources statistics and settings */
+ struct bc_resource_parm bc_parms[BC_RESOURCES];
+};
+
+enum bc_severity { BC_BARRIER, BC_LIMIT, BC_FORCE };
+
+/* Flags passed to beancounter_findcreate() */
+#define BC_LOOKUP 0x00
+#define BC_ALLOC 0x01 /* may allocate new one */
+#define BC_ALLOC_ATOMIC 0x02 /* when BC_ALLOC is set causes
+     * GFP_ATOMIC allocation
+     */
+
+#ifdef CONFIG_BEANCOUNTERS
+
+/*
+ * These functions tune minheld and maxheld values for a given
+ * resource when held value changes
+ */
+static inline void bc_adjust_maxheld(struct beancounter *bc, int resource)
+{
+ struct bc_resource_parm *parm;
+
+ parm = &bc->bc_parms[resource];
+ if (parm->maxheld < parm->held)
+  parm->maxheld = parm->held;
+}
+
+static inline void bc_adjust_minheld(struct beancounter *bc, int resource)
+{
+ struct bc_resource_parm *parm;
+
+ parm = &bc->bc_parms[resource];
+ if (parm->minheld > parm->held)
+  parm->minheld = parm->held;
```

```
+}
+
+int __must_check bc_charge_locked(struct beancounter *bc,
+  int res, unsigned long val, enum bc_severity strict);
+int __must_check bc_charge(struct beancounter *bc,
+  int res, unsigned long val, enum bc_severity strict);
+
+void bc_uncharge_locked(struct beancounter *bc, int res, unsigned long val);
+void bc_uncharge(struct beancounter *bc, int res, unsigned long val);
+
+struct beancounter *beancounter_findcreate(bcid_t id, int mask);
+
+static inline struct beancounter *get_beancounter(struct beancounter *bc)
+{
+ atomic_inc(&bc->bc_refcount);
+ return bc;
+}
+
+void put_beancounter(struct beancounter *bc);
+
+void bc_init_early(void);
+void bc_init_late(void);
+void bc_init_proc(void);
+
+extern struct beancounter init_bc;
+extern const char *bc_rnames[];
+
+#else /* CONFIG_BEANCOUNTERS */
+
+#define beancounter_findcreate(id, f)   (NULL)
+#define get_beancounter(bc)    (NULL)
+#define put_beancounter(bc)    do { } while (0)
+
+static inline __must_check int bc_charge_locked(struct beancounter *bc,
+  int res, unsigned long val, enum bc_severity strict)
+{
+ return 0;
+}
+
+static inline __must_check int bc_charge(struct beancounter *bc,
+  int res, unsigned long val, enum bc_severity strict)
+{
+ return 0;
+}
+
+static inline void bc_uncharge_locked(struct beancounter *bc, int res,
+  unsigned long val)
+{
```

```
+}
+
+static inline void bc_uncharge(struct beancounter *bc, int res,
+  unsigned long val)
+{
+}
+
+#define bc_init_early()     do { } while (0)
+#define bc_init_late()     do { } while (0)
+#define bc_init_proc()     do { } while (0)
+
+#endif /* CONFIG_BEANCOUNTERS */
+#endif /* __KERNEL__ */
+
+#endif /* _LINUX_BEANCOUNTER_H */
--- ./include/linux/types.h.bccore 2006-09-05 11:47:33.000000000 +0400
+++ ./include/linux/types.h 2006-09-05 12:06:35.000000000 +0400
@@ -40,6 +40,21 @@ typedef __kernel_gid32_t gid_t;
 typedef __kernel_uid16_t        uid16_t;
 typedef __kernel_gid16_t        gid16_t;

+/*
+ * Type of beancounter id (CONFIG_BEANCOUNTERS)
+ *
+ * The ancient Unix implementations of this kind of resource management and
+ * security are built around setluid() which sets a uid value that cannot
+ * be changed again and is normally used for security purposes. That
+ * happened to be a uid_t and in simple setups at login uid = luid = euid
+ * would be the norm.
+ *
+ * Thus the Linux one happens to be a uid_t. It could be something else but
+ * for the "container per user" model whatever a container is must be able
+ * to hold all possible uid_t values. Alan Cox.
+ */
+typedef uid_t    bcid_t;
+
 #ifdef CONFIG_UID16
 /* This is defined by include/asm-{arch}/posix_types.h */
 typedef __kernel_old_uid_t old_uid_t;
@@ -52,6 +67,7 @@ typedef __kernel_old_gid_t old_gid_t;
 #else
 typedef __kernel_uid_t  uid_t;
 typedef __kernel_gid_t  gid_t;
+typedef __kernel_uid_t  bcid_t;
 #endif /* __KERNEL__ */

 #if defined(__GNUC__) && !defined(__STRICT_ANSI__)
--- ./init/main.c.bccore 2006-09-05 11:47:33.000000000 +0400
```

```
+++ ./init/main.c 2006-09-05 12:06:35.000000000 +0400
@@ -50,6 +50,8 @@
 #include <linux/debug_locks.h>
 #include <linux/lockdep.h>

+#include <bc/beancounter.h>
+
 #include <asm/io.h>
 #include <asm/bugs.h>
 #include <asm/setup.h>
@@ -493,6 +495,7 @@ asmlinkage void __init start_kernel(void
  early_boot_irqs_off();
  early_init_irq_lock_class();

+ bc_init_early();
 /*
  * Interrupts are still disabled. Do necessary setups, then
  * enable them
@@ -585,6 +588,7 @@ asmlinkage void __init start_kernel(void
 #endif
  fork_init(num_physpages);
  proc_caches_init();
+ bc_init_late();
  buffer_init();
  unnamed_dev_init();
  key_init();
--- ./kernel/Makefile.bccore 2006-09-05 11:47:33.000000000 +0400
+++ ./kernel/Makefile 2006-09-05 12:09:53.000000000 +0400
@@ -12,6 +12,7 @@ obj-y     = sched.o fork.o exec_domain.o

 obj-$(CONFIG_STACKTRACE) += stacktrace.o
 obj-y += time/
+obj-$(CONFIG_BEANCOUNTERS) += bc/
 obj-$(CONFIG_DEBUG_MUTEXES) += mutex-debug.o
 obj-$(CONFIG_LOCKDEP) += lockdep.o
 ifeq ($(CONFIG_PROC_FS),y)
--- ./kernel/bc/Makefile.bccore 2006-09-05 12:06:35.000000000 +0400
+++ ./kernel/bc/Makefile 2006-09-05 12:10:05.000000000 +0400
@@ -0,0 +1,7 @@
+#
+# Beancounters (BC)
+#
+# Copyright (C) 2006 OpenVZ. SWsoft Inc
+#
+
+obj-y += beancounter.o
--- ./kernel/bc/beancounter.c.bccore 2006-09-05 12:06:35.000000000 +0400
+++ ./kernel/bc/beancounter.c 2006-09-05 12:16:50.000000000 +0400
```

```
@@ -0,0 +1,263 @@
+/*
+ *  kernel/bc/beancounter.c
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *  Original code by (C) 1998      Alan Cox
+ *                   1998-2000 Andrey Savochkin <saw@saw.sw.com.sg>
+ */
+
+#include <linux/slab.h>
+#include <linux/module.h>
+#include <linux/hash.h>
+
+#include <bc/beancounter.h>
+
+static kmem_cache_t *bc_cachep;
+static struct beancounter default_beancounter;
+
+static void init_beancounter_struct(struct beancounter *bc, bcid_t id);
+
+struct beancounter init_bc;
+
+const char *bc_rnames[] = {
+};
+
+#define BC_HASH_BITS  8
+#define BC_HASH_SIZE  (1 << BC_HASH_BITS)
+
+static struct hlist_head bc_hash[BC_HASH_SIZE];
+static spinlock_t bc_hash_lock;
+#define bc_hash_fn(bcid) (hash_long(bcid, BC_HASH_BITS))
+
+/*
+ * Per resource beancounting. Resources are tied to their bc id.
+ * The resource structure itself is tagged both to the process and
+ * the charging resources (a socket doesn't want to have to search for
+ * things at irq time for example). Reference counters keep things in
+ * hand.
+ *
+ * The case where a user creates resource, kills all his processes and
+ * then starts new ones is correctly handled this way. The refcounters
+ * will mean the old entry is still around with resource tied to it.
+ */
+
+struct beancounter *beancounter_findcreate(bcid_t id, int mask)
+{
+ struct beancounter *new_bc, *bc;
+ unsigned long flags;
```

```
+ struct hlist_head *slot;
+ struct hlist_node *pos;
+
+ slot = &bc_hash[bc_hash_fn(id)];
+ new_bc = NULL;
+
+retry:
+ spin_lock_irqsave(&bc_hash_lock, flags);
+ hlist_for_each_entry (bc, pos, slot, hash)
+  if (bc->bc_id == id)
+   break;
+
+ if (pos != NULL) {
+  get_beancounter(bc);
+  spin_unlock_irqrestore(&bc_hash_lock, flags);
+
+  if (new_bc != NULL)
+   kmem_cache_free(bc_cachep, new_bc);
+  return bc;
+ }
+
+ if (new_bc != NULL)
+  goto out_install;
+
+ spin_unlock_irqrestore(&bc_hash_lock, flags);
+
+ if (!(mask & BC_ALLOC))
+  goto out;
+
+ new_bc = kmem_cache_alloc(bc_cachep,
+   mask & BC_ALLOC_ATOMIC ? GFP_ATOMIC : GFP_KERNEL);
+ if (new_bc == NULL)
+  goto out;
+
+ *new_bc = default_beancounter;
+ init_beancounter_struct(new_bc, id);
+ goto retry;
+
+out_install:
+ hlist_add_head(&new_bc->hash, slot);
+ spin_unlock_irqrestore(&bc_hash_lock, flags);
+out:
+ return new_bc;
+}
+
+void put_beancounter(struct beancounter *bc)
+{
+ int i;
```

```
+ unsigned long flags;
+
+ if (!atomic_dec_and_lock_irqsave(&bc->bc_refcount,
+     &bc_hash_lock, flags))
+  return;
+
+ BUG_ON(bc == &init_bc);
+
+ for (i = 0; i < BC_RESOURCES; i++)
+  if (bc->bc_parms[i].held != 0)
+   printk("BC: %d has %lu of %s held on put", bc->bc_id,
+    bc->bc_parms[i].held, bc_rnames[i]);
+
+ hlist_del(&bc->hash);
+ spin_unlock_irqrestore(&bc_hash_lock, flags);
+
+ kmem_cache_free(bc_cachep, bc);
+}
+
+EXPORT_SYMBOL_GPL(put_beancounter);
+
+/*
+ * Generic resource charging stuff
+ */
+
+/* called with bc->bc_lock held and interrupts disabled */
+int bc_charge_locked(struct beancounter *bc, int resource, unsigned long val,
+  enum bc_severity strict)
+{
+ unsigned long new_held;
+
+ /*
+  * bc_value <= BC_MAXVALUE, value <= BC_MAXVALUE, and only one addition
+  * at the moment is possible so an overflow is impossible.
+  */
+ new_held = bc->bc_parms[resource].held + val;
+
+ switch (strict) {
+ case BC_BARRIER:
+  if (bc->bc_parms[resource].held >
+    bc->bc_parms[resource].barrier)
+   break;
+  /* fallthrough */
+ case BC_LIMIT:
+  if (bc->bc_parms[resource].held >
+    bc->bc_parms[resource].limit)
+   break;
+  /* fallthrough */
```

```
+ case BC_FORCE:
+  bc->bc_parms[resource].held = new_held;
+  bc_adjust_maxheld(bc, resource);
+  return 0;
+
+ default:
+  BUG();
+ }
+
+ bc->bc_parms[resource].failcnt++;
+ return -ENOMEM;
+}
+EXPORT_SYMBOL_GPL(bc_charge_locked);
+
+int bc_charge(struct beancounter *bc, int resource, unsigned long val,
+  enum bc_severity strict)
+{
+ int retval;
+ unsigned long flags;
+
+ BUG_ON(val > BC_MAXVALUE);
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ retval = bc_charge_locked(bc, resource, val, strict);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+ return retval;
+}
+EXPORT_SYMBOL_GPL(bc_charge);
+
+/* called with bc->bc_lock held and interrupts disabled */
+void bc_uncharge_locked(struct beancounter *bc, int resource, unsigned long val)
+{
+ if (unlikely(bc->bc_parms[resource].held < val)) {
+  printk("BC: overuncharging bc %d %s: val %lu, holds %lu\n",
+    bc->bc_id, bc_rnames[resource], val,
+    bc->bc_parms[resource].held);
+  val = bc->bc_parms[resource].held;
+ }
+
+ bc->bc_parms[resource].held -= val;
+ bc_adjust_minheld(bc, resource);
+}
+EXPORT_SYMBOL_GPL(bc_uncharge_locked);
+
+void bc_uncharge(struct beancounter *bc, int resource, unsigned long val)
+{
+ unsigned long flags;
+
```

```
+ BUG_ON(val > BC_MAXVALUE);
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ bc_uncharge_locked(bc, resource, val);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+}
+EXPORT_SYMBOL_GPL(bc_uncharge);
+
+/*
+ * Initialization
+ *
+ * struct beancounter contains
+ *  - limits and other configuration settings
+ *  - structural fields: lists, spinlocks and so on.
+ *
+ * Before these parts are initialized, the structure should be memset
+ * to 0 or copied from a known clean structure.  That takes care of a lot
+ * of fields not initialized explicitly.
+ */
+
+static void init_beancounter_struct(struct beancounter *bc, bcid_t id)
+{
+ atomic_set(&bc->bc_refcount, 1);
+ spin_lock_init(&bc->bc_lock);
+ bc->bc_id = id;
+}
+
+static void init_beancounter_nolimits(struct beancounter *bc)
+{
+ int k;
+
+ for (k = 0; k < BC_RESOURCES; k++) {
+  bc->bc_parms[k].limit = BC_MAXVALUE;
+  bc->bc_parms[k].barrier = BC_MAXVALUE;
+ }
+}
+
+static void init_beancounter_syslimits(struct beancounter *bc)
+{
+ int k;
+
+ for (k = 0; k < BC_RESOURCES; k++)
+  bc->bc_parms[k].barrier = bc->bc_parms[k].limit;
+}
+
+void __init bc_init_early(void)
+{
+ struct beancounter *bc;
```

```
+ struct hlist_head *slot;
+
+ bc = &init_bc;
+
+ init_beancounter_nolimits(bc);
+ init_beancounter_struct(bc, 0);
+
+ spin_lock_init(&bc_hash_lock);
+ slot = &bc_hash[bc_hash_fn(bc->bc_id)];
+ hlist_add_head(&bc->hash, slot);
+}
+
+void __init bc_init_late(void)
+{
+ struct beancounter *bc;
+
+ bc_cachep = kmem_cache_create("beancounters",
+   sizeof(struct beancounter), 0,
+   SLAB_HWCACHE_ALIGN | SLAB_PANIC, NULL, NULL);
+
+ bc = &default_beancounter;
+ init_beancounter_syslimits(bc);
+ init_beancounter_struct(bc, 0);
+}
```

Subject: [PATCH 4/13] BC: context inheriting and changing
Posted by dev on Tue, 05 Sep 2006 15:19:40 GMT
View Forum Message <> Reply to Message

Contains code responsible for setting BC on task,
it's inheriting and setting host context in interrupts.

Task references 2 beancounters:
1. exec_bc: current context. all resources are
        charged to this beancounter.
3. fork_bc: beancounter which is inherited by
        task's children on fork

Signed-off-by: Pavel Emelianov <xemul@sw.ru>
Signed-off-by: Kirill Korotaev <dev@sw.ru>

```
 include/bc/task.h       |   57 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 include/linux/sched.h   |    5 ++++
 kernel/bc/Makefile      |    1
 kernel/bc/beancounter.c |    3 ++
```

```
 kernel/bc/misc.c        |   31 ++++++++++++++++++++++++++++
 kernel/fork.c           |    5 ++++
 kernel/irq/handle.c     |    9 +++++++
 kernel/softirq.c        |    8 ++++++
 8 files changed, 119 insertions(+)

--- ./include/bc/task.h.bctask 2006-09-05 12:24:07.000000000 +0400
+++ ./include/bc/task.h 2006-09-05 12:38:53.000000000 +0400
@@ -0,0 +1,57 @@
+/*
+ *  include/bc/task.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef __BC_TASK_H_
+#define __BC_TASK_H_
+
+struct beancounter;
+
+struct task_beancounter {
+ struct beancounter *exec_bc;
+ struct beancounter *fork_bc;
+};
+
+#ifdef CONFIG_BEANCOUNTERS
+
+#define get_exec_bc() (current->task_bc.exec_bc)
+
+#define set_exec_bc(new) ({    \
+  struct task_beancounter *tbc;  \
+  struct beancounter *old;  \
+  tbc = &current->task_bc;  \
+  old = tbc->exec_bc;   \
+  tbc->exec_bc = new;   \
+  old;     \
+ })
+
+#define reset_exec_bc(old, expected) do {  \
+  struct task_beancounter *tbc;  \
+  tbc = &current->task_bc;  \
+  BUG_ON(tbc->exec_bc != expected); \
+  tbc->exec_bc = old;   \
+ } while (0)
+
+void bc_task_charge(struct task_struct *parent, struct task_struct *new);
+void bc_task_uncharge(struct task_struct *tsk);
```

```
+
+#else
+
+#define get_exec_bc()   (NULL)
+#define set_exec_bc(new)  (NULL)
+#define reset_exec_bc(new, expected) do { } while (0)
+
+static inline void bc_task_charge(struct task_struct *parent,
+  struct task_struct *new)
+{
+}
+
+static inline void bc_task_uncharge(struct task_struct *tsk)
+{
+}
+
+#endif
+#endif
--- ./include/linux/sched.h.bctask 2006-09-05 11:47:33.000000000 +0400
+++ ./include/linux/sched.h 2006-09-05 12:33:45.000000000 +0400
@@ -83,6 +83,8 @@ struct sched_param {
 #include <linux/timer.h>
 #include <linux/hrtimer.h>

+#include <bc/task.h>
+
 #include <asm/processor.h>

 struct exec_domain;
@@ -1041,6 +1043,9 @@ struct task_struct {
 #ifdef CONFIG_TASK_DELAY_ACCT
  struct task_delay_info *delays;
 #endif
+#ifdef CONFIG_BEANCOUNTERS
+ struct task_beancounter task_bc;
+#endif
 };

 static inline pid_t process_group(struct task_struct *tsk)
--- ./kernel/bc/Makefile.bctask 2006-09-05 12:10:05.000000000 +0400
+++ ./kernel/bc/Makefile 2006-09-05 12:24:39.000000000 +0400
@@ -5,3 +5,4 @@
 #

 obj-y += beancounter.o
+obj-y += misc.o
--- ./kernel/bc/beancounter.c.bctask 2006-09-05 12:16:50.000000000 +0400
+++ ./kernel/bc/beancounter.c 2006-09-05 12:24:07.000000000 +0400
```

```
@@ -247,6 +247,9 @@ void __init bc_init_early(void)
  spin_lock_init(&bc_hash_lock);
  slot = &bc_hash[bc_hash_fn(bc->bc_id)];
  hlist_add_head(&bc->hash, slot);
+
+ current->task_bc.exec_bc = get_beancounter(bc);
+ current->task_bc.fork_bc = get_beancounter(bc);
 }

 void __init bc_init_late(void)
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/misc.c 2006-09-05 12:30:57.000000000 +0400
@@ -0,0 +1,31 @@
+/*
+ * kernel/bc/misc.c
+ *
+ * Copyright (C) 2006 OpenVZ. SWsoft Inc.
+ *
+ */
+
+#include <linux/sched.h>
+
+#include <bc/beancounter.h>
+#include <bc/task.h>
+
+void bc_task_charge(struct task_struct *parent, struct task_struct *new)
+{
+ struct task_beancounter *old_bc;
+ struct task_beancounter *new_bc;
+ struct beancounter *bc;
+
+ old_bc = &parent->task_bc;
+ new_bc = &new->task_bc;
+
+ bc = old_bc->fork_bc;
+ new_bc->exec_bc = get_beancounter(bc);
+ new_bc->fork_bc = get_beancounter(bc);
+}
+
+void bc_task_uncharge(struct task_struct *tsk)
+{
+ put_beancounter(tsk->task_bc.exec_bc);
+ put_beancounter(tsk->task_bc.fork_bc);
+}
--- ./kernel/fork.c.bctask 2006-09-05 11:47:33.000000000 +0400
+++ ./kernel/fork.c 2006-09-05 12:30:38.000000000 +0400
@@ -48,6 +48,8 @@
 #include <linux/delayacct.h>
```

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

+#include <bc/task.h>
+
 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
 #include <asm/uaccess.h>
@@ -104,6 +106,7 @@ static kmem_cache_t *mm_cachep;

 void free_task(struct task_struct *tsk)
 {
+ bc_task_uncharge(tsk);
  free_thread_info(tsk->thread_info);
  rt_mutex_debug_task_free(tsk);
  free_task_struct(tsk);
@@ -979,6 +982,8 @@ static struct task_struct *copy_process(
  if (!p)
   goto fork_out;

+ bc_task_charge(current, p);
+
 #ifdef CONFIG_TRACE_IRQFLAGS
  DEBUG_LOCKS_WARN_ON(!p->hardirqs_enabled);
  DEBUG_LOCKS_WARN_ON(!p->softirqs_enabled);
--- ./kernel/irq/handle.c.bctask 2006-09-05 11:47:33.000000000 +0400
+++ ./kernel/irq/handle.c 2006-09-05 12:24:07.000000000 +0400
@@ -16,6 +16,9 @@
 #include <linux/interrupt.h>
 #include <linux/kernel_stat.h>

+#include <bc/beancounter.h>
+#include <bc/task.h>
+
 #include "internals.h"

 /**
@@ -171,6 +174,9 @@ fastcall unsigned int __do_IRQ(unsigned
  struct irq_desc *desc = irq_desc + irq;
  struct irqaction *action;
  unsigned int status;
+ struct beancounter *bc;
+
+ bc = set_exec_bc(&init_bc);

  kstat_this_cpu.irqs[irq]++;
  if (CHECK_IRQ_PER_CPU(desc->status)) {
@@ -183,6 +189,8 @@ fastcall unsigned int __do_IRQ(unsigned
    desc->chip->ack(irq);
```

```
   action_ret = handle_IRQ_event(irq, regs, desc->action);
   desc->chip->end(irq);
+
+  reset_exec_bc(bc, &init_bc);
   return 1;
  }

@@ -251,6 +259,7 @@ out:
  desc->chip->end(irq);
  spin_unlock(&desc->lock);

+ reset_exec_bc(bc, &init_bc);
  return 1;
 }

--- ./kernel/softirq.c.bctask 2006-09-05 11:47:33.000000000 +0400
+++ ./kernel/softirq.c 2006-09-05 12:38:42.000000000 +0400
@@ -18,6 +18,9 @@
 #include <linux/rcupdate.h>
 #include <linux/smp.h>

+#include <bc/beancounter.h>
+#include <bc/task.h>
+
 #include <asm/irq.h>
 /*
    - No shared variables, all the data are CPU local.
@@ -209,6 +212,9 @@ asmlinkage void __do_softirq(void)
  __u32 pending;
  int max_restart = MAX_SOFTIRQ_RESTART;
  int cpu;
+ struct beancounter *bc;
+
+ bc = set_exec_bc(&init_bc);

  pending = local_softirq_pending();
  account_system_vtime(current);
@@ -247,6 +253,8 @@ restart:

  account_system_vtime(current);
  _local_bh_enable();
+
+ reset_exec_bc(bc, &init_bc);
 }

 #ifndef __ARCH_HAS_DO_SOFTIRQ
```

Subject: [PATCH 5/13] BC: user interface (syscalls)
Posted by dev on Tue, 05 Sep 2006 15:21:07 GMT
View Forum Message <> Reply to Message

Add the following system calls for BC management:
 1. sys_get_bcid   - get current BC id
 2. sys_set_bcid   - change exec_ and fork_ BCs on current
 3. sys_set_bclimit - set limits for resources consumtions
 4. sys_get_bcstat   - return br_resource_parm on resource

Signed-off-by: Pavel Emelianov <xemul@sw.ru>
Signed-off-by: Kirill Korotaev <dev@sw.ru>

---

 arch/i386/kernel/syscall_table.S | 4 +
 arch/ia64/kernel/entry.S | 4 +
 arch/sparc/kernel/entry.S | 2
 arch/sparc/kernel/systbls.S | 6 +
 arch/sparc64/kernel/entry.S | 2
 arch/sparc64/kernel/systbls.S | 10 ++-
 include/asm-i386/unistd.h | 6 +
 include/asm-ia64/unistd.h | 6 +
 include/asm-powerpc/systbl.h | 4 +
 include/asm-powerpc/unistd.h | 6 +
 include/asm-sparc/unistd.h | 4 +
 include/asm-sparc64/unistd.h | 4 +
 include/asm-x86_64/unistd.h | 10 ++-
 kernel/bc/Makefile | 1
 kernel/bc/sys.c | 120 ++
 kernel/sys_ni.c | 6 +
 16 files changed, 186 insertions(+), 9 deletions(-)

--- ./arch/i386/kernel/syscall_table.S.bcsys 2006-09-05 11:47:31.000000000 +0400
+++ ./arch/i386/kernel/syscall_table.S 2006-09-05 12:47:21.000000000 +0400
@@ -318,3 +318,7 @@ ENTRY(sys_call_table)
  .long sys_vmsplice
  .long sys_move_pages
  .long sys_getcpu
+ .long sys_get_bcid
+ .long sys_set_bcid /* 320 */
+ .long sys_set_bclimit
+ .long sys_get_bcstat
--- ./arch/ia64/kernel/entry.S.bcsys 2006-09-05 11:47:31.000000000 +0400
+++ ./arch/ia64/kernel/entry.S 2006-09-05 12:47:21.000000000 +0400
@@ -1610,5 +1610,9 @@ sys_call_table:
  data8 sys_sync_file_range // 1300
  data8 sys_tee
  data8 sys_vmsplice

```
+ data8 sys_get_bcid
+ data8 sys_set_bcid
+ data8 sys_set_bclimit   // 1305
+ data8 sys_get_bcstat

  .org sys_call_table + 8*NR_syscalls // guard against failures to increase NR_syscalls
--- ./arch/sparc/kernel/entry.S.bcsys 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/sparc/kernel/entry.S 2006-09-05 12:47:21.000000000 +0400
@@ -37,7 +37,7 @@

 #define curptr      g6

-#define NR_SYSCALLS 300      /* Each OS is different... */
+#define NR_SYSCALLS 304      /* Each OS is different... */

 /* These are just handy. */
 #define _SV save %sp, -STACKFRAME_SZ, %sp
--- ./arch/sparc/kernel/systbls.S.bcsys 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/sparc/kernel/systbls.S 2006-09-05 12:47:21.000000000 +0400
@@ -78,7 +78,8 @@ sys_call_table:
 /*285*/ .long sys_mkdirat, sys_mknodat, sys_fchownat, sys_futimesat, sys_fstatat64
 /*290*/ .long sys_unlinkat, sys_renameat, sys_linkat, sys_symlinkat, sys_readlinkat
 /*295*/ .long sys_fchmodat, sys_faccessat, sys_pselect6, sys_ppoll, sys_unshare
-/*300*/ .long sys_set_robust_list, sys_get_robust_list
+/*300*/ .long sys_set_robust_list, sys_get_robust_list, sys_get_bcid, sys_set_bcid,
sys_set_bclimit
+/*305*/ .long sys_get_bcstat

 #ifdef CONFIG_SUNOS_EMUL
  /* Now the SunOS syscall table. */
@@ -192,4 +193,7 @@ sunos_sys_table:
  .long sunos_nosys, sunos_nosys, sunos_nosys
  .long sunos_nosys, sunos_nosys, sunos_nosys

+ .long sunos_nosys, sunos_nosys, sunos_nosys,
+ .long sunos_nosys
+
 #endif
--- ./arch/sparc64/kernel/entry.S.bcsys 2006-07-10 12:39:10.000000000 +0400
+++ ./arch/sparc64/kernel/entry.S 2006-09-05 12:47:21.000000000 +0400
@@ -25,7 +25,7 @@

 #define curptr      g6

-#define NR_SYSCALLS 300      /* Each OS is different... */
+#define NR_SYSCALLS 304      /* Each OS is different... */

  .text
```

```
  .align  32
--- ./arch/sparc64/kernel/systbls.S.bcsys 2006-07-10 12:39:11.000000000 +0400
+++ ./arch/sparc64/kernel/systbls.S 2006-09-05 12:47:21.000000000 +0400
@@ -79,7 +79,8 @@ sys_call_table32:
  .word sys_mkdirat, sys_mknodat, sys_fchownat, compat_sys_futimesat, compat_sys_fstatat64
 /*290*/ .word sys_unlinkat, sys_renameat, sys_linkat, sys_symlinkat, sys_readlinkat
  .word sys_fchmodat, sys_faccessat, compat_sys_pselect6, compat_sys_ppoll, sys_unshare
-/*300*/ .word compat_sys_set_robust_list, compat_sys_get_robust_list
+/*300*/ .word compat_sys_set_robust_list, compat_sys_get_robust_list, sys_nis_syscall,
sys_nis_syscall, sys_nis_syscall
+ .word sys_nis_syscall

 #endif /* CONFIG_COMPAT */

@@ -149,7 +150,9 @@ sys_call_table:
  .word sys_mkdirat, sys_mknodat, sys_fchownat, sys_futimesat, sys_fstatat64
 /*290*/ .word sys_unlinkat, sys_renameat, sys_linkat, sys_symlinkat, sys_readlinkat
  .word sys_fchmodat, sys_faccessat, sys_pselect6, sys_ppoll, sys_unshare
-/*300*/ .word sys_set_robust_list, sys_get_robust_list
+/*300*/ .word sys_set_robust_list, sys_get_robust_list, sys_get_bcid, sys_set_bcid,
sys_set_bclimit
+ .word sys_get_bcstat
+

 #if defined(CONFIG_SUNOS_EMUL) || defined(CONFIG_SOLARIS_EMUL) || \
     defined(CONFIG_SOLARIS_EMUL_MODULE)
@@ -263,4 +266,7 @@ sunos_sys_table:
  .word sunos_nosys, sunos_nosys, sunos_nosys
  .word sunos_nosys, sunos_nosys, sunos_nosys
+
+ .word sunos_nosys, sunos_nosys, sunos_nosys
+ .word sunos_nosys
 #endif
--- ./include/asm-i386/unistd.h.bcsys 2006-09-05 11:47:33.000000000 +0400
+++ ./include/asm-i386/unistd.h 2006-09-05 12:48:37.000000000 +0400
@@ -324,8 +324,12 @@
 #define __NR_vmsplice  316
 #define __NR_move_pages  317
 #define __NR_getcpu  318
+#define __NR_get_bcid  319
+#define __NR_set_bcid  320
+#define __NR_set_bclimit 321
+#define __NR_get_bcstat  322

-#define NR_syscalls 318
+#define NR_syscalls 323
 #include <linux/err.h>
```

```
 /*
--- ./include/asm-ia64/unistd.h.bcsys 2006-09-05 11:47:33.000000000 +0400
+++ ./include/asm-ia64/unistd.h 2006-09-05 12:47:21.000000000 +0400
@@ -291,11 +291,15 @@
 #define __NR_sync_file_range  1300
 #define __NR_tee   1301
 #define __NR_vmsplice   1302
+#define __NR_get_bcid   1303
+#define __NR_set_bcid   1304
+#define __NR_set_bclimit  1305
+#define __NR_get_bcstat   1306

 #ifdef __KERNEL__


-#define NR_syscalls   279 /* length of syscall table */
+#define NR_syscalls   283 /* length of syscall table */

 #define __ARCH_WANT_SYS_RT_SIGACTION

--- ./include/asm-powerpc/systbl.h.bcsys 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-powerpc/systbl.h 2006-09-05 12:47:21.000000000 +0400
@@ -304,3 +304,7 @@ SYSCALL_SPU(fchmodat)
 SYSCALL_SPU(faccessat)
 COMPAT_SYS_SPU(get_robust_list)
 COMPAT_SYS_SPU(set_robust_list)
+SYSCALL(sys_get_bcid)
+SYSCALL(sys_set_bcid)
+SYSCALL(sys_set_bclimit)
+SYSCALL(sys_get_bcstat)
--- ./include/asm-powerpc/unistd.h.bcsys 2006-09-05 11:47:33.000000000 +0400
+++ ./include/asm-powerpc/unistd.h 2006-09-05 12:47:21.000000000 +0400
@@ -323,10 +323,14 @@
 #define __NR_faccessat  298
 #define __NR_get_robust_list 299
 #define __NR_set_robust_list 300
+#define __NR_get_bcid  301
+#define __NR_set_bcid  302
+#define __NR_set_bclimit 303
+#define __NR_get_bcstat  304

 #ifdef __KERNEL__

-#define __NR_syscalls  301
+#define __NR_syscalls  305

 #define __NR__exit __NR_exit
```

```
 #define NR_syscalls __NR_syscalls
--- ./include/asm-sparc/unistd.h.bcsys 2006-09-05 11:47:33.000000000 +0400
+++ ./include/asm-sparc/unistd.h 2006-09-05 12:47:21.000000000 +0400
@@ -318,6 +318,10 @@
 #define __NR_unshare  299
 #define __NR_set_robust_list 300
 #define __NR_get_robust_list 301
+#define __NR_get_bcid  302
+#define __NR_set_bcid  303
+#define __NR_set_bclimit 304
+#define __NR_get_bcstat  305

 #ifdef __KERNEL__
 /* WARNING: You MAY NOT add syscall numbers larger than 301, since
--- ./include/asm-sparc64/unistd.h.bcsys 2006-09-05 11:47:33.000000000 +0400
+++ ./include/asm-sparc64/unistd.h 2006-09-05 12:47:21.000000000 +0400
@@ -320,6 +320,10 @@
 #define __NR_unshare  299
 #define __NR_set_robust_list 300
 #define __NR_get_robust_list 301
+#define __NR_get_bcid  302
+#define __NR_set_bcid  303
+#define __NR_set_bclimit 304
+#define __NR_get_bcstat  305

 #ifdef __KERNEL__
 /* WARNING: You MAY NOT add syscall numbers larger than 301, since
--- ./include/asm-x86_64/unistd.h.bcsys 2006-09-05 11:47:33.000000000 +0400
+++ ./include/asm-x86_64/unistd.h 2006-09-05 12:49:03.000000000 +0400
@@ -619,8 +619,16 @@ __SYSCALL(__NR_sync_file_range, sys_sync
 __SYSCALL(__NR_vmsplice, sys_vmsplice)
 #define __NR_move_pages  279
 __SYSCALL(__NR_move_pages, sys_move_pages)
+#define __NR_get_bcid  280
+__SYSCALL(__NR_get_bcid, sys_get_bcid)
+#define __NR_set_bcid  281
+__SYSCALL(__NR_set_bcid, sys_set_bcid)
+#define __NR_set_bclimit 282
+__SYSCALL(__NR_set_bclimit, sys_set_bclimit)
+#define __NR_get_bcstat  283
+__SYSCALL(__NR_get_bcstat, sys_get_bcstat)

-#define __NR_syscall_max __NR_move_pages
+#define __NR_syscall_max __NR_get_bcstat
 #include <linux/err.h>

 #ifndef __NO_STUBS
--- ./kernel/bc/Makefile.bcsys 2006-09-05 12:24:39.000000000 +0400
```

```
+++ ./kernel/bc/Makefile 2006-09-05 12:49:28.000000000 +0400
@@ -6,3 +6,4 @@

 obj-y += beancounter.o
 obj-y += misc.o
+obj-y += sys.o
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/sys.c 2006-09-05 12:47:21.000000000 +0400
@@ -0,0 +1,120 @@
+/*
+ *  kernel/bc/sys.c
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#include <linux/sched.h>
+#include <asm/uaccess.h>
+
+#include <bc/beancounter.h>
+#include <bc/task.h>
+
+asmlinkage long sys_get_bcid(void)
+{
+ struct beancounter *bc;
+
+ bc = get_exec_bc();
+ return bc->bc_id;
+}
+
+asmlinkage long sys_set_bcid(bcid_t id)
+{
+ int error;
+ struct beancounter *bc;
+ struct task_beancounter *task_bc;
+
+ task_bc = &current->task_bc;
+
+ /* You may only set an bc as root */
+ error = -EPERM;
+ if (!capable(CAP_SETUID))
+  goto out;
+
+ /* Ok - set up a beancounter entry for this user */
+ error = -ENOMEM;
+ bc = beancounter_findcreate(id, BC_ALLOC);
+ if (bc == NULL)
+  goto out;
```

```
+
+ /* install bc */
+ put_beancounter(task_bc->exec_bc);
+ task_bc->exec_bc = bc;
+ put_beancounter(task_bc->fork_bc);
+ task_bc->fork_bc = get_beancounter(bc);
+ error = 0;
+out:
+ return error;
+}
+
+asmlinkage long sys_set_bclimit(bcid_t id, unsigned long resource,
+  unsigned long __user *limits)
+{
+ int error;
+ unsigned long flags;
+ struct beancounter *bc;
+ unsigned long new_limits[2];
+
+ error = -EPERM;
+ if(!capable(CAP_SYS_RESOURCE))
+  goto out;
+
+ error = -EINVAL;
+ if (resource >= BC_RESOURCES)
+  goto out;
+
+ error = -EFAULT;
+ if (copy_from_user(&new_limits, limits, sizeof(new_limits)))
+  goto out;
+
+ error = -EINVAL;
+ if (new_limits[0] > BC_MAXVALUE || new_limits[1] > BC_MAXVALUE ||
+    new_limits[0] > new_limits[1])
+  goto out;
+
+ error = -ENOENT;
+ bc = beancounter_findcreate(id, BC_LOOKUP);
+ if (bc == NULL)
+  goto out;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ bc->bc_parms[resource].barrier = new_limits[0];
+ bc->bc_parms[resource].limit = new_limits[1];
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+
+ put_beancounter(bc);
+ error = 0;
```

```
+out:
+ return error;
+}
+
+int sys_get_bcstat(bcid_t id, unsigned long resource,
+  struct bc_resource_parm __user *uparm)
+{
+ int error;
+ unsigned long flags;
+ struct beancounter *bc;
+ struct bc_resource_parm parm;
+
+ error = -EINVAL;
+ if (resource >= BC_RESOURCES)
+  goto out;
+
+ error = -ENOENT;
+ bc = beancounter_findcreate(id, BC_LOOKUP);
+ if (bc == NULL)
+  goto out;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ parm = bc->bc_parms[resource];
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+ put_beancounter(bc);
+
+ error = 0;
+ if (copy_to_user(uparm, &parm, sizeof(parm)))
+  error = -EFAULT;
+
+out:
+ return error;
+}
--- ./kernel/sys_ni.c.bcsys 2006-09-05 11:47:33.000000000 +0400
+++ ./kernel/sys_ni.c 2006-09-05 12:49:16.000000000 +0400
@@ -139,3 +139,9 @@ cond_syscall(compat_sys_move_pages);
 cond_syscall(sys_bdflush);
 cond_syscall(sys_ioprio_set);
 cond_syscall(sys_ioprio_get);
+
+/* user resources syscalls */
+cond_syscall(sys_set_bcid);
+cond_syscall(sys_get_bcid);
+cond_syscall(sys_set_bclimit);
+cond_syscall(sys_get_bcstat);
```

Subject: [PATCH 6/13] BC: kernel memory (core)
Posted by dev on Tue, 05 Sep 2006 15:21:47 GMT
View Forum Message <> Reply to Message

Introduce BC_KMEMSIZE resource which accounts kernel
objects allocated by task's request.

Reference to BC is kept on struct page or slab object.
For slabs each struct slab contains a set of pointers
corresponding objects are charged to.

Allocation charge rules:
1. Pages - if allocation is performed with __GFP_BC flag - page
   is charged to current's exec_bc.
2. Slabs - kmem_cache may be created with SLAB_BC flag - in this
   case each allocation is charged. Caches used by kmalloc are
   created with SLAB_BC | SLAB_BC_NOCHARGE flags. In this case
   only __GFP_BC allocations are charged.

Signed-off-by: Pavel Emelianov <xemul@sw.ru>
Signed-off-by: Kirill Korotaev <dev@sw.ru>

---

 include/bc/beancounter.h | 4 +
 include/bc/kmem.h | 46 ++++++++++++++++++
 include/linux/gfp.h | 8 ++-
 include/linux/mm.h | 4 +
 include/linux/slab.h | 4 +
 include/linux/vmalloc.h | 1
 kernel/bc/Makefile | 1
 kernel/bc/beancounter.c | 3 +
 kernel/bc/kmem.c | 85 ++++++++++++++++++++++++++++++++++++
 mm/mempool.c | 2
 mm/page_alloc.c | 11 ++++
 mm/slab.c | 121 ++---------
 mm/vmalloc.c | 6 ++
 13 files changed, 271 insertions(+), 25 deletions(-)

--- ./include/bc/beancounter.h.bckmemcore 2006-09-05 12:54:17.000000000 +0400
+++ ./include/bc/beancounter.h 2006-09-05 12:54:40.000000000 +0400
@@ -12,7 +12,9 @@
  * Resource list.
  */

-#define BC_RESOURCES 0
+#define BC_KMEMSIZE 0
+
+#define BC_RESOURCES 1

```
 struct bc_resource_parm {
  unsigned long barrier; /* A barrier over which resource allocations
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/kmem.h 2006-09-05 12:54:40.000000000 +0400
@@ -0,0 +1,46 @@
+/*
+ *  include/bc/kmem.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef __BC_KMEM_H_
+#define __BC_KMEM_H_
+
+/*
+ * BC_KMEMSIZE accounting
+ */
+
+struct mm_struct;
+struct page;
+struct beancounter;
+
+#ifdef CONFIG_BEANCOUNTERS
+int __must_check bc_page_charge(struct page *page, int order, gfp_t flags);
+void bc_page_uncharge(struct page *page, int order);
+
+int __must_check bc_slab_charge(kmem_cache_t *cachep, void *obj, gfp_t flags);
+void bc_slab_uncharge(kmem_cache_t *cachep, void *obj);
+#else
+static inline int __must_check bc_page_charge(struct page *page,
+  int order, gfp_t flags)
+{
+ return 0;
+}
+
+static inline void bc_page_uncharge(struct page *page, int order)
+{
+}
+
+static inline int __must_check bc_slab_charge(kmem_cache_t *cachep,
+  void *obj, gfp_t flags)
+{
+ return 0;
+}
+
+static inline void bc_slab_uncharge(kmem_cache_t *cachep, void *obj)
```

```
+{
+}
+#endif
+#endif /* __BC_SLAB_H_ */
--- ./include/linux/gfp.h.bckmemcore 2006-09-05 12:53:55.000000000 +0400
+++ ./include/linux/gfp.h 2006-09-05 12:54:40.000000000 +0400
@@ -46,15 +46,18 @@ struct vm_area_struct;
 #define __GFP_NOMEMALLOC ((__force gfp_t)0x10000u) /* Don't use emergency reserves */
 #define __GFP_HARDWALL   ((__force gfp_t)0x20000u) /* Enforce hardwall cpuset memory
allocs */
 #define __GFP_THISNODE ((__force gfp_t)0x40000u)/* No fallback, no policies */
+#define __GFP_BC  ((__force gfp_t)0x80000u) /* Charge allocation with BC */
+#define __GFP_BC_LIMIT ((__force gfp_t)0x100000u) /* Charge against BC limit */

-#define __GFP_BITS_SHIFT 20 /* Room for 20 __GFP_FOO bits */
+#define __GFP_BITS_SHIFT 21 /* Room for 21 __GFP_FOO bits */
 #define __GFP_BITS_MASK ((__force gfp_t)((1 << __GFP_BITS_SHIFT) - 1))

 /* if you forget to add the bitmask here kernel will crash, period */
 #define GFP_LEVEL_MASK (__GFP_WAIT|__GFP_HIGH|__GFP_IO|__GFP_FS| \
    __GFP_COLD|__GFP_NOWARN|__GFP_REPEAT| \
    __GFP_NOFAIL|__GFP_NORETRY|__GFP_NO_GROW|__GFP_COMP| \
-   __GFP_NOMEMALLOC|__GFP_HARDWALL|__GFP_THISNODE)
+   __GFP_NOMEMALLOC|__GFP_HARDWALL|__GFP_THISNODE| \
+   __GFP_BC|__GFP_BC_LIMIT)

 /* This equals 0, but use constants in case they ever change */
 #define GFP_NOWAIT (GFP_ATOMIC & ~__GFP_HIGH)
@@ -63,6 +66,7 @@ struct vm_area_struct;
 #define GFP_NOIO (__GFP_WAIT)
 #define GFP_NOFS (__GFP_WAIT | __GFP_IO)
 #define GFP_KERNEL (__GFP_WAIT | __GFP_IO | __GFP_FS)
+#define GFP_KERNEL_BC (__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_BC)
 #define GFP_USER (__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_HARDWALL)
 #define GFP_HIGHUSER (__GFP_WAIT | __GFP_IO | __GFP_FS | __GFP_HARDWALL | \
     __GFP_HIGHMEM)
--- ./include/linux/mm.h.bckmemcore 2006-09-05 12:53:55.000000000 +0400
+++ ./include/linux/mm.h 2006-09-05 12:55:28.000000000 +0400
@@ -274,8 +274,12 @@ struct page {
  unsigned int gfp_mask;
  unsigned long trace[8];
 #endif
+#ifdef CONFIG_BEANCOUNTERS
+ struct beancounter *page_bc;
+#endif
 };

+#define page_bc(page)   ((page)->page_bc)
```

```
 #define page_private(page)  ((page)->private)
 #define set_page_private(page, v) ((page)->private = (v))

--- ./include/linux/slab.h.bckmemcore 2006-09-05 12:53:59.000000000 +0400
+++ ./include/linux/slab.h 2006-09-05 12:54:40.000000000 +0400
@@ -46,6 +46,8 @@ typedef struct kmem_cache kmem_cache_t;
 #define SLAB_PANIC  0x00040000UL /* panic if kmem_cache_create() fails */
 #define SLAB_DESTROY_BY_RCU 0x00080000UL /* defer freeing pages to RCU */
 #define SLAB_MEM_SPREAD  0x00100000UL /* Spread some memory over cpuset */
+#define SLAB_BC  0x00200000UL /* Account with BC */
+#define SLAB_BC_NOCHARGE 0x00400000UL /* Explicit accounting */

 /* flags passed to a constructor func */
 #define SLAB_CTOR_CONSTRUCTOR 0x001UL  /* if not set, then deconstructor */
@@ -291,6 +293,8 @@ extern kmem_cache_t *fs_cachep;
 extern kmem_cache_t *sighand_cachep;
 extern kmem_cache_t *bio_cachep;

+struct beancounter;
+struct beancounter **kmem_cache_bcp(kmem_cache_t *cachep, void *obj);
 #endif /* __KERNEL__ */

 #endif /* _LINUX_SLAB_H */
--- ./include/linux/vmalloc.h.bckmemcore 2006-09-05 12:53:59.000000000 +0400
+++ ./include/linux/vmalloc.h 2006-09-05 12:54:40.000000000 +0400
@@ -36,6 +36,7 @@ struct vm_struct {
  * Highlevel APIs for driver use
  */
 extern void *vmalloc(unsigned long size);
+extern void *vmalloc_bc(unsigned long size);
 extern void *vmalloc_user(unsigned long size);
 extern void *vmalloc_node(unsigned long size, int node);
 extern void *vmalloc_exec(unsigned long size);
--- ./kernel/bc/Makefile.bckmemcore 2006-09-05 12:54:24.000000000 +0400
+++ ./kernel/bc/Makefile 2006-09-05 12:54:50.000000000 +0400
@@ -7,3 +7,4 @@
 obj-y += beancounter.o
 obj-y += misc.o
 obj-y += sys.o
+obj-y += kmem.o
--- ./kernel/bc/beancounter.c.bckmemcore 2006-09-05 12:54:21.000000000 +0400
+++ ./kernel/bc/beancounter.c 2006-09-05 12:55:13.000000000 +0400
@@ -20,6 +20,7 @@ static void init_beancounter_struct(stru
 struct beancounter init_bc;

 const char *bc_rnames[] = {
+ "kmemsize", /* 0 */
 };
```

```
 #define BC_HASH_BITS  8
@@ -230,6 +231,8 @@ static void init_beancounter_syslimits(s
 {
  int k;

+ bc->bc_parms[BC_KMEMSIZE].limit = 32 * 1024 * 1024;
+
  for (k = 0; k < BC_RESOURCES; k++)
   bc->bc_parms[k].barrier = bc->bc_parms[k].limit;
 }
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/kmem.c 2006-09-05 12:54:40.000000000 +0400
@@ -0,0 +1,85 @@
+/*
+ *  kernel/bc/kmem.c
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/gfp.h>
+#include <linux/slab.h>
+#include <linux/mm.h>
+
+#include <bc/beancounter.h>
+#include <bc/kmem.h>
+#include <bc/task.h>
+
+/*
+ * Slab accounting
+ */
+
+int bc_slab_charge(kmem_cache_t *cachep, void *objp, gfp_t flags)
+{
+ unsigned int size;
+ struct beancounter *bc, **slab_bcp;
+
+ bc = get_exec_bc();
+
+ size = kmem_cache_size(cachep);
+ if (bc_charge(bc, BC_KMEMSIZE, size,
+   (flags & __GFP_BC_LIMIT ? BC_LIMIT : BC_BARRIER)))
+  return -ENOMEM;
+
+ slab_bcp = kmem_cache_bcp(cachep, objp);
+ *slab_bcp = get_beancounter(bc);
```

```
+ return 0;
+}
+
+void bc_slab_uncharge(kmem_cache_t *cachep, void *objp)
+{
+ unsigned int size;
+ struct beancounter *bc, **slab_bcp;
+
+ slab_bcp = kmem_cache_bcp(cachep, objp);
+ if (*slab_bcp == NULL)
+  return;
+
+ bc = *slab_bcp;
+ size = kmem_cache_size(cachep);
+ bc_uncharge(bc, BC_KMEMSIZE, size);
+ put_beancounter(bc);
+ *slab_bcp = NULL;
+}
+
+/*
+ * Pages accounting
+ */
+
+int bc_page_charge(struct page *page, int order, gfp_t flags)
+{
+ struct beancounter *bc;
+
+ BUG_ON(page_bc(page) != NULL);
+
+ bc = get_exec_bc();
+
+ if (bc_charge(bc, BC_KMEMSIZE, PAGE_SIZE << order,
+    (flags & __GFP_BC_LIMIT ? BC_LIMIT : BC_BARRIER)))
+  return -ENOMEM;
+
+ page_bc(page) = get_beancounter(bc);
+ return 0;
+}
+
+void bc_page_uncharge(struct page *page, int order)
+{
+ struct beancounter *bc;
+
+ bc = page_bc(page);
+ if (bc == NULL)
+  return;
+
+ bc_uncharge(bc, BC_KMEMSIZE, PAGE_SIZE << order);
```

```
+ put_beancounter(bc);
+ page_bc(page) = NULL;
+}
--- ./mm/mempool.c.bckmemcore 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/mempool.c 2006-09-05 12:54:40.000000000 +0400
@@ -119,6 +119,7 @@ int mempool_resize(mempool_t *pool, int
  unsigned long flags;

  BUG_ON(new_min_nr <= 0);
+ gfp_mask &= ~__GFP_BC;

  spin_lock_irqsave(&pool->lock, flags);
  if (new_min_nr <= pool->min_nr) {
@@ -212,6 +213,7 @@ void * mempool_alloc(mempool_t *pool, gf
  gfp_mask |= __GFP_NOMEMALLOC; /* don't allocate emergency reserves */
  gfp_mask |= __GFP_NORETRY; /* don't loop in __alloc_pages */
  gfp_mask |= __GFP_NOWARN; /* failures are OK */
+ gfp_mask &= ~__GFP_BC;  /* do not charge */

  gfp_temp = gfp_mask & ~(__GFP_WAIT|__GFP_IO);

--- ./mm/page_alloc.c.bckmemcore 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/page_alloc.c 2006-09-05 12:54:40.000000000 +0400
@@ -40,6 +40,8 @@
 #include <linux/sort.h>
 #include <linux/pfn.h>

+#include <bc/kmem.h>
+
 #include <asm/tlbflush.h>
 #include <asm/div64.h>
 #include "internal.h"
@@ -516,6 +518,8 @@ static void __free_pages_ok(struct page
  if (reserved)
   return;

+ bc_page_uncharge(page, order);
+
  kernel_map_pages(page, 1 << order, 0);
  local_irq_save(flags);
  __count_vm_events(PGFREE, 1 << order);
@@ -799,6 +803,8 @@ static void fastcall free_hot_cold_page(
  if (free_pages_check(page))
   return;

+ bc_page_uncharge(page, 0);
+
  kernel_map_pages(page, 1, 0);
```

```
  pcp = &zone_pcp(zone, get_cpu())->pcp[cold];
@@ -1188,6 +1194,11 @@ nopage:
   show_mem();
  }
 got_pg:
+ if ((gfp_mask & __GFP_BC) &&
+    bc_page_charge(page, order, gfp_mask)) {
+  __free_pages(page, order);
+  page = NULL;
+ }
 #ifdef CONFIG_PAGE_OWNER
  if (page)
   set_page_owner(page, order, gfp_mask);
--- ./mm/slab.c.bckmemcore 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/slab.c 2006-09-05 12:54:40.000000000 +0400
@@ -108,6 +108,8 @@
 #include <linux/mutex.h>
 #include <linux/rtmutex.h>

+#include <bc/kmem.h>
+
 #include <asm/uaccess.h>
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>
@@ -175,11 +177,13 @@
     SLAB_CACHE_DMA | \
     SLAB_MUST_HWCACHE_ALIGN | SLAB_STORE_USER | \
     SLAB_RECLAIM_ACCOUNT | SLAB_PANIC | \
+    SLAB_BC | SLAB_BC_NOCHARGE | \
     SLAB_DESTROY_BY_RCU | SLAB_MEM_SPREAD)
 #else
 # define CREATE_MASK (SLAB_HWCACHE_ALIGN | \
     SLAB_CACHE_DMA | SLAB_MUST_HWCACHE_ALIGN | \
     SLAB_RECLAIM_ACCOUNT | SLAB_PANIC | \
+    SLAB_BC | SLAB_BC_NOCHARGE | \
     SLAB_DESTROY_BY_RCU | SLAB_MEM_SPREAD)
 #endif

@@ -793,9 +797,33 @@ static struct kmem_cache *kmem_find_gene
  return __find_general_cachep(size, gfpflags);
 }

-static size_t slab_mgmt_size(size_t nr_objs, size_t align)
+static size_t slab_mgmt_size_raw(size_t nr_objs)
 {
- return ALIGN(sizeof(struct slab)+nr_objs*sizeof(kmem_bufctl_t), align);
+ return sizeof(struct slab) + nr_objs * sizeof(kmem_bufctl_t);
```

```
+}
+
+#ifdef CONFIG_BEANCOUNTERS
+#define BC_EXTRASIZE sizeof(struct beancounter *)
+static inline size_t slab_mgmt_size_noalign(int flags, size_t nr_objs)
+{
+ size_t size;
+
+ size = slab_mgmt_size_raw(nr_objs);
+ if (flags & SLAB_BC)
+  size = ALIGN(size, BC_EXTRASIZE) + nr_objs * BC_EXTRASIZE;
+ return size;
+}
+#else
+#define BC_EXTRASIZE 0
+static inline size_t slab_mgmt_size_noalign(int flags, size_t nr_objs)
+{
+ return slab_mgmt_size_raw(nr_objs);
+}
+#endif
+
+static inline size_t slab_mgmt_size(int flags, size_t nr_objs, size_t align)
+{
+ return ALIGN(slab_mgmt_size_noalign(flags, nr_objs), align);
 }

 /*
@@ -840,20 +868,21 @@ static void cache_estimate(unsigned long
    * into account.
    */
   nr_objs = (slab_size - sizeof(struct slab)) /
-     (buffer_size + sizeof(kmem_bufctl_t));
+     (buffer_size + sizeof(kmem_bufctl_t) +
+     (flags & SLAB_BC ? BC_EXTRASIZE : 0));

   /*
    * This calculated number will be either the right
    * amount, or one greater than what we want.
    */
-  if (slab_mgmt_size(nr_objs, align) + nr_objs*buffer_size
+  if (slab_mgmt_size(flags, nr_objs, align) + nr_objs*buffer_size
         > slab_size)
    nr_objs--;

   if (nr_objs > SLAB_LIMIT)
    nr_objs = SLAB_LIMIT;

-  mgmt_size = slab_mgmt_size(nr_objs, align);
```

```
+  mgmt_size = slab_mgmt_size(flags, nr_objs, align);
  }
  *num = nr_objs;
  *left_over = slab_size - nr_objs*buffer_size - mgmt_size;
@@ -1412,7 +1441,8 @@ void __init kmem_cache_init(void)
  sizes[INDEX_AC].cs_cachep = kmem_cache_create(names[INDEX_AC].name,
      sizes[INDEX_AC].cs_size,
      ARCH_KMALLOC_MINALIGN,
-     ARCH_KMALLOC_FLAGS|SLAB_PANIC,
+     ARCH_KMALLOC_FLAGS | SLAB_BC |
+      SLAB_BC_NOCHARGE | SLAB_PANIC,
      NULL, NULL);

  if (INDEX_AC != INDEX_L3) {
@@ -1420,7 +1450,8 @@ void __init kmem_cache_init(void)
    kmem_cache_create(names[INDEX_L3].name,
     sizes[INDEX_L3].cs_size,
     ARCH_KMALLOC_MINALIGN,
-    ARCH_KMALLOC_FLAGS|SLAB_PANIC,
+    ARCH_KMALLOC_FLAGS | SLAB_BC |
+     SLAB_BC_NOCHARGE | SLAB_PANIC,
     NULL, NULL);
  }

@@ -1438,7 +1469,8 @@ void __init kmem_cache_init(void)
    sizes->cs_cachep = kmem_cache_create(names->name,
      sizes->cs_size,
      ARCH_KMALLOC_MINALIGN,
-     ARCH_KMALLOC_FLAGS|SLAB_PANIC,
+     ARCH_KMALLOC_FLAGS | SLAB_BC |
+      SLAB_BC_NOCHARGE | SLAB_PANIC,
      NULL, NULL);
   }

@@ -1941,7 +1973,8 @@ static size_t calculate_slab_order(struc
     * looping condition in cache_grow().
     */
    offslab_limit = size - sizeof(struct slab);
-   offslab_limit /= sizeof(kmem_bufctl_t);
+   offslab_limit /= (sizeof(kmem_bufctl_t) +
+     (flags & SLAB_BC ? BC_EXTRASIZE : 0));

    if (num > offslab_limit)
    break;
@@ -2249,8 +2282,8 @@ kmem_cache_create (const char *name, siz
   cachep = NULL;
   goto oops;
  }
```

```
- slab_size = ALIGN(cachep->num * sizeof(kmem_bufctl_t)
-     + sizeof(struct slab), align);
+
+ slab_size = slab_mgmt_size(flags, cachep->num, align);

  /*
   * If the slab has been placed off-slab, and we have enough space then
@@ -2261,11 +2294,9 @@ kmem_cache_create (const char *name, siz
   left_over -= slab_size;
  }

- if (flags & CFLGS_OFF_SLAB) {
+ if (flags & CFLGS_OFF_SLAB)
   /* really off slab. No need for manual alignment */
-  slab_size =
-      cachep->num * sizeof(kmem_bufctl_t) + sizeof(struct slab);
- }
+  slab_size = slab_mgmt_size_noalign(flags, cachep->num);

  cachep->colour_off = cache_line_size();
  /* Offset must be a multiple of the alignment. */
@@ -2509,6 +2540,30 @@ void kmem_cache_destroy(struct kmem_cach
 }
 EXPORT_SYMBOL(kmem_cache_destroy);

+static inline kmem_bufctl_t *slab_bufctl(struct slab *slabp)
+{
+ return (kmem_bufctl_t *) (slabp + 1);
+}
+
+#ifdef CONFIG_BEANCOUNTERS
+static inline struct beancounter **slab_bc_ptrs(kmem_cache_t *cachep,
+  struct slab *slabp)
+{
+ return (struct beancounter **) ALIGN((unsigned long)
+   (slab_bufctl(slabp) + cachep->num), BC_EXTRASIZE);
+}
+
+struct beancounter **kmem_cache_bcp(kmem_cache_t *cachep, void *objp)
+{
+ struct slab *slabp;
+ struct beancounter **bcs;
+
+ slabp = virt_to_slab(objp);
+ bcs = slab_bc_ptrs(cachep, slabp);
+ return bcs + obj_to_index(cachep, slabp, objp);
+}
+#endif
```

```
+
 /*
  * Get the memory for a slab management obj.
  * For a slab cache when the slab descriptor is off-slab, slab descriptors
@@ -2529,7 +2584,8 @@ static struct slab *alloc_slabmgmt(struc
  if (OFF_SLAB(cachep)) {
   /* Slab management obj is off-slab. */
   slabp = kmem_cache_alloc_node(cachep->slabp_cache,
-           local_flags, nodeid);
+           local_flags & (~__GFP_BC),
+           nodeid);
   if (!slabp)
    return NULL;
  } else {
@@ -2540,14 +2596,14 @@ static struct slab *alloc_slabmgmt(struc
  slabp->colouroff = colour_off;
  slabp->s_mem = objp + colour_off;
  slabp->nodeid = nodeid;
+#ifdef CONFIG_BEANCOUNTERS
+ if (cachep->flags & SLAB_BC)
+  memset(slab_bc_ptrs(cachep, slabp), 0,
+    cachep->num * BC_EXTRASIZE);
+#endif
  return slabp;
 }

-static inline kmem_bufctl_t *slab_bufctl(struct slab *slabp)
-{
- return (kmem_bufctl_t *) (slabp + 1);
-}
-
 static void cache_init_objs(struct kmem_cache *cachep,
        struct slab *slabp, unsigned long ctor_flags)
 {
@@ -2725,7 +2781,7 @@ static int cache_grow(struct kmem_cache
   * Get mem for the objs.  Attempt to allocate a physical page from
   * 'nodeid'.
   */
- objp = kmem_getpages(cachep, flags, nodeid);
+ objp = kmem_getpages(cachep, flags & (~__GFP_BC), nodeid);
  if (!objp)
   goto failed;

@@ -3073,6 +3129,19 @@ static inline void *____cache_alloc(stru
  return objp;
 }

+static inline int bc_should_charge(kmem_cache_t *cachep, gfp_t flags)
```

```
+{
+#ifdef CONFIG_BEANCOUNTERS
+ if (!(cachep->flags & SLAB_BC))
+  return 0;
+ if (flags & __GFP_BC)
+  return 1;
+ if (!(cachep->flags & SLAB_BC_NOCHARGE))
+  return 1;
+#endif
+ return 0;
+}
+
 static __always_inline void *__cache_alloc(struct kmem_cache *cachep,
       gfp_t flags, void *caller)
 {
@@ -3086,6 +3155,12 @@ static __always_inline void *__cache_all
  local_irq_restore(save_flags);
  objp = cache_alloc_debugcheck_after(cachep, flags, objp,
         caller);
+
+ if (objp && bc_should_charge(cachep, flags))
+  if (bc_slab_charge(cachep, objp, flags)) {
+   kmem_cache_free(cachep, objp);
+   objp = NULL;
+  }
  prefetchw(objp);
  return objp;
 }
@@ -3283,6 +3358,8 @@ static inline void __cache_free(struct k
  struct array_cache *ac = cpu_cache_get(cachep);

  check_irq_off();
+ if (cachep->flags & SLAB_BC)
+  bc_slab_uncharge(cachep, objp);
  objp = cache_free_debugcheck(cachep, objp, __builtin_return_address(0));

  if (cache_free_alien(cachep, objp))
--- ./mm/vmalloc.c.bckmemcore 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/vmalloc.c 2006-09-05 12:54:40.000000000 +0400
@@ -520,6 +520,12 @@ void *vmalloc(unsigned long size)
 }
 EXPORT_SYMBOL(vmalloc);

+void *vmalloc_bc(unsigned long size)
+{
+ return __vmalloc(size, GFP_KERNEL_BC | __GFP_HIGHMEM, PAGE_KERNEL);
+}
+EXPORT_SYMBOL(vmalloc_bc);
```

```
+
 /**
  * vmalloc_user  -  allocate virtually contiguous memory which has
  *      been zeroed so it can be mapped to userspace without
```

Subject: [PATCH 7/13] BC: kernel memory (marks)
Posted by dev on Tue, 05 Sep 2006 15:23:00 GMT
View Forum Message <> Reply to Message

Mark some kmem caches with SLAB_BC and some allocations
with __GFP_BC to cause charging/limiting of appropriate
kernel resources.

Signed-off-by: Pavel Emelianov <xemul@sw.ru>
Signed-off-by: Kirill Korotaev <dev@sw.ru>

```
 arch/i386/kernel/ldt.c          |    4 ++--
 arch/i386/mm/init.c             |    4 ++--
 arch/i386/mm/pgtable.c          |    6 ++++--
 drivers/char/tty_io.c           |   10 +++++-----
 fs/file.c                       |    8 ++++----
 fs/locks.c                      |    2 +-
 fs/namespace.c                  |    3 ++-
 fs/select.c                     |    7 ++++---
 include/asm-i386/thread_info.h  |    4 ++--
 include/asm-ia64/pgalloc.h      |   24 ++++++++++++++++++++-------
 include/asm-x86_64/pgalloc.h    |   12 ++++++++----
 include/asm-x86_64/thread_info.h |    5 +++--
 ipc/msgutil.c                   |    4 ++--
 ipc/sem.c                       |    7 ++++---
 ipc/util.c                      |    8 ++++----
 kernel/fork.c                   |   15 ++++++++-------
 kernel/posix-timers.c           |    3 ++-
 kernel/signal.c                 |    2 +-
 kernel/user.c                   |    2 +-
 mm/rmap.c                       |    3 ++-
 mm/shmem.c                      |    3 ++-
 21 files changed, 80 insertions(+), 56 deletions(-)

--- ./arch/i386/kernel/ldt.c.bckmemch 2006-09-05 12:53:51.000000000 +0400
+++ ./arch/i386/kernel/ldt.c 2006-09-05 12:58:17.000000000 +0400
@@ -39,9 +39,9 @@ static int alloc_ldt(mm_context_t *pc, i
  oldsize = pc->size;
  mincount = (mincount+511)&(~511);
  if (mincount*LDT_ENTRY_SIZE > PAGE_SIZE)
```

```
-  newldt = vmalloc(mincount*LDT_ENTRY_SIZE);
+  newldt = vmalloc_bc(mincount*LDT_ENTRY_SIZE);
  else
-  newldt = kmalloc(mincount*LDT_ENTRY_SIZE, GFP_KERNEL);
+  newldt = kmalloc(mincount*LDT_ENTRY_SIZE, GFP_KERNEL_BC);

  if (!newldt)
   return -ENOMEM;
--- ./arch/i386/mm/init.c.bckmemch 2006-09-05 12:53:51.000000000 +0400
+++ ./arch/i386/mm/init.c 2006-09-05 12:58:17.000000000 +0400
@@ -709,7 +709,7 @@ void __init pgtable_cache_init(void)
   pmd_cache = kmem_cache_create("pmd",
      PTRS_PER_PMD*sizeof(pmd_t),
      PTRS_PER_PMD*sizeof(pmd_t),
-     0,
+     SLAB_BC,
      pmd_ctor,
      NULL);
   if (!pmd_cache)
@@ -718,7 +718,7 @@ void __init pgtable_cache_init(void)
  pgd_cache = kmem_cache_create("pgd",
     PTRS_PER_PGD*sizeof(pgd_t),
     PTRS_PER_PGD*sizeof(pgd_t),
-    0,
+    SLAB_BC,
     pgd_ctor,
     PTRS_PER_PMD == 1 ? pgd_dtor : NULL);
  if (!pgd_cache)
--- ./arch/i386/mm/pgtable.c.bckmemch 2006-09-05 12:53:51.000000000 +0400
+++ ./arch/i386/mm/pgtable.c 2006-09-05 12:58:17.000000000 +0400
@@ -186,9 +186,11 @@ struct page *pte_alloc_one(struct mm_str
  struct page *pte;

 #ifdef CONFIG_HIGHPTE
- pte =  alloc_pages(GFP_KERNEL|__GFP_HIGHMEM|__GFP_REPEAT|__GFP_ZERO , 0);
+ pte =  alloc_pages(GFP_KERNEL|__GFP_HIGHMEM|__GFP_REPEAT|__GFP_ZERO |
+    __GFP_BC | __GFP_BC_LIMIT, 0);
 #else
- pte = alloc_pages(GFP_KERNEL|__GFP_REPEAT|__GFP_ZERO, 0);
+ pte = alloc_pages(GFP_KERNEL|__GFP_REPEAT|__GFP_ZERO|
+    __GFP_BC | __GFP_BC_LIMIT, 0);
 #endif
  return pte;
 }
--- ./drivers/char/tty_io.c.bckmemch 2006-09-05 12:53:52.000000000 +0400
+++ ./drivers/char/tty_io.c 2006-09-05 12:58:17.000000000 +0400
@@ -165,7 +165,7 @@ static void release_mem(struct tty_struc
```

```
 static struct tty_struct *alloc_tty_struct(void)
 {
- return kzalloc(sizeof(struct tty_struct), GFP_KERNEL);
+ return kzalloc(sizeof(struct tty_struct), GFP_KERNEL_BC);
 }

 static void tty_buffer_free_all(struct tty_struct *);
@@ -1904,7 +1904,7 @@ static int init_dev(struct tty_driver *d

  if (!*tp_loc) {
   tp = (struct termios *) kmalloc(sizeof(struct termios),
-      GFP_KERNEL);
+      GFP_KERNEL_BC);
   if (!tp)
    goto free_mem_out;
   *tp = driver->init_termios;
@@ -1912,7 +1912,7 @@ static int init_dev(struct tty_driver *d

  if (!*ltp_loc) {
   ltp = (struct termios *) kmalloc(sizeof(struct termios),
-       GFP_KERNEL);
+       GFP_KERNEL_BC);
   if (!ltp)
    goto free_mem_out;
   memset(ltp, 0, sizeof(struct termios));
@@ -1937,7 +1937,7 @@ static int init_dev(struct tty_driver *d

   if (!*o_tp_loc) {
    o_tp = (struct termios *)
-    kmalloc(sizeof(struct termios), GFP_KERNEL);
+    kmalloc(sizeof(struct termios), GFP_KERNEL_BC);
    if (!o_tp)
     goto free_mem_out;
    *o_tp = driver->other->init_termios;
@@ -1945,7 +1945,7 @@ static int init_dev(struct tty_driver *d

   if (!*o_ltp_loc) {
    o_ltp = (struct termios *)
-    kmalloc(sizeof(struct termios), GFP_KERNEL);
+    kmalloc(sizeof(struct termios), GFP_KERNEL_BC);
    if (!o_ltp)
     goto free_mem_out;
    memset(o_ltp, 0, sizeof(struct termios));
--- ./fs/file.c.bckmemch 2006-09-05 12:53:55.000000000 +0400
+++ ./fs/file.c 2006-09-05 12:58:17.000000000 +0400
@@ -44,9 +44,9 @@ struct file ** alloc_fd_array(int num)
  int size = num * sizeof(struct file *);
```

```
  if (size <= PAGE_SIZE)
-  new_fds = (struct file **) kmalloc(size, GFP_KERNEL);
+  new_fds = (struct file **) kmalloc(size, GFP_KERNEL_BC);
  else
-  new_fds = (struct file **) vmalloc(size);
+  new_fds = (struct file **) vmalloc_bc(size);
  return new_fds;
 }

@@ -213,9 +213,9 @@ fd_set * alloc_fdset(int num)
  int size = num / 8;

  if (size <= PAGE_SIZE)
-  new_fdset = (fd_set *) kmalloc(size, GFP_KERNEL);
+  new_fdset = (fd_set *) kmalloc(size, GFP_KERNEL_BC);
  else
-  new_fdset = (fd_set *) vmalloc(size);
+  new_fdset = (fd_set *) vmalloc_bc(size);
  return new_fdset;
 }

--- ./fs/locks.c.bckmemch 2006-09-05 12:53:55.000000000 +0400
+++ ./fs/locks.c 2006-09-05 12:58:17.000000000 +0400
@@ -2228,7 +2228,7 @@ EXPORT_SYMBOL(lock_may_write);
 static int __init filelock_init(void)
 {
  filelock_cache = kmem_cache_create("file_lock_cache",
-   sizeof(struct file_lock), 0, SLAB_PANIC,
+   sizeof(struct file_lock), 0, SLAB_PANIC | SLAB_BC,
    init_once, NULL);
  return 0;
 }
--- ./fs/namespace.c.bckmemch 2006-09-05 12:53:55.000000000 +0400
+++ ./fs/namespace.c 2006-09-05 12:58:17.000000000 +0400
@@ -1812,7 +1812,8 @@ void __init mnt_init(unsigned long mempa
  init_rwsem(&namespace_sem);

  mnt_cache = kmem_cache_create("mnt_cache", sizeof(struct vfsmount),
-   0, SLAB_HWCACHE_ALIGN | SLAB_PANIC, NULL, NULL);
+   0, SLAB_HWCACHE_ALIGN | SLAB_BC | SLAB_PANIC,
+   NULL, NULL);

  mount_hashtable = (struct list_head *)__get_free_page(GFP_ATOMIC);

--- ./fs/select.c.bckmemch 2006-09-05 12:53:55.000000000 +0400
+++ ./fs/select.c 2006-09-05 12:58:17.000000000 +0400
@@ -103,7 +103,8 @@ static struct poll_table_entry *poll_get
  if (!table || POLL_TABLE_FULL(table)) {
```

```
   struct poll_table_page *new_table;

-  new_table = (struct poll_table_page *) __get_free_page(GFP_KERNEL);
+  new_table = (struct poll_table_page *)
+    __get_free_page(GFP_KERNEL_BC);
   if (!new_table) {
    p->error = -ENOMEM;
    __set_current_state(TASK_RUNNING);
@@ -339,7 +340,7 @@ static int core_sys_select(int n, fd_set
  if (size > sizeof(stack_fds) / 6) {
   /* Not enough space in on-stack array; must use kmalloc */
   ret = -ENOMEM;
-  bits = kmalloc(6 * size, GFP_KERNEL);
+  bits = kmalloc(6 * size, GFP_KERNEL_BC);
   if (!bits)
    goto out_nofds;
  }
@@ -693,7 +694,7 @@ int do_sys_poll(struct pollfd __user *uf
   if (!stack_pp)
    stack_pp = pp = (struct poll_list *)stack_pps;
   else {
-   pp = kmalloc(size, GFP_KERNEL);
+   pp = kmalloc(size, GFP_KERNEL_BC);
    if (!pp)
     goto out_fds;
   }
--- ./include/asm-i386/thread_info.h.bckmemch 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-i386/thread_info.h 2006-09-05 12:58:17.000000000 +0400
@@ -99,13 +99,13 @@ static inline struct thread_info *curren
  ({      \
   struct thread_info *ret;   \
        \
-  ret = kmalloc(THREAD_SIZE, GFP_KERNEL);  \
+  ret = kmalloc(THREAD_SIZE, GFP_KERNEL_BC); \
   if (ret)     \
    memset(ret, 0, THREAD_SIZE);  \
   ret;      \
  })
 #else
-#define alloc_thread_info(tsk) kmalloc(THREAD_SIZE, GFP_KERNEL)
+#define alloc_thread_info(tsk) kmalloc(THREAD_SIZE, GFP_KERNEL_BC)
 #endif

 #define free_thread_info(info) kfree(info)
--- ./include/asm-ia64/pgalloc.h.bckmemch 2006-07-10 12:39:19.000000000 +0400
+++ ./include/asm-ia64/pgalloc.h 2006-09-05 12:58:17.000000000 +0400
@@ -19,6 +19,8 @@
 #include <linux/page-flags.h>
```

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

+#include <bc/kmem.h>
+
 #include <asm/mmu_context.h>

 DECLARE_PER_CPU(unsigned long *, __pgtable_quicklist);
@@ -37,7 +39,7 @@ static inline long pgtable_quicklist_tot
  return ql_size;
 }

-static inline void *pgtable_quicklist_alloc(void)
+static inline void *pgtable_quicklist_alloc(int charge)
 {
  unsigned long *ret = NULL;

@@ -45,13 +47,20 @@ static inline void *pgtable_quicklist_al

  ret = pgtable_quicklist;
  if (likely(ret != NULL)) {
+  if (charge && bc_page_charge(virt_to_page(ret),
+      0, __GFP_BC_LIMIT)) {
+   ret = NULL;
+   goto out;
+  }
   pgtable_quicklist = (unsigned long *)(*ret);
   ret[0] = 0;
   --pgtable_quicklist_size;
+out:
   preempt_enable();
  } else {
   preempt_enable();
-  ret = (unsigned long *)__get_free_page(GFP_KERNEL | __GFP_ZERO);
+  ret = (unsigned long *)__get_free_page(GFP_KERNEL |
+      __GFP_ZERO | __GFP_BC | __GFP_BC_LIMIT);
  }

  return ret;
@@ -69,6 +78,7 @@ static inline void pgtable_quicklist_fre
 #endif

  preempt_disable();
+ bc_page_uncharge(virt_to_page(pgtable_entry), 0);
  *(unsigned long *)pgtable_entry = (unsigned long)pgtable_quicklist;
  pgtable_quicklist = (unsigned long *)pgtable_entry;
  ++pgtable_quicklist_size;
@@ -77,7 +87,7 @@ static inline void pgtable_quicklist_fre
```

```
 static inline pgd_t *pgd_alloc(struct mm_struct *mm)
 {
- return pgtable_quicklist_alloc();
+ return pgtable_quicklist_alloc(1);
 }

 static inline void pgd_free(pgd_t * pgd)
@@ -94,7 +104,7 @@ pgd_populate(struct mm_struct *mm, pgd_t

 static inline pud_t *pud_alloc_one(struct mm_struct *mm, unsigned long addr)
 {
- return pgtable_quicklist_alloc();
+ return pgtable_quicklist_alloc(1);
 }

 static inline void pud_free(pud_t * pud)
@@ -112,7 +122,7 @@ pud_populate(struct mm_struct *mm, pud_t

 static inline pmd_t *pmd_alloc_one(struct mm_struct *mm, unsigned long addr)
 {
- return pgtable_quicklist_alloc();
+ return pgtable_quicklist_alloc(1);
 }

 static inline void pmd_free(pmd_t * pmd)
@@ -137,13 +147,13 @@ pmd_populate_kernel(struct mm_struct *mm
 static inline struct page *pte_alloc_one(struct mm_struct *mm,
       unsigned long addr)
 {
- return virt_to_page(pgtable_quicklist_alloc());
+ return virt_to_page(pgtable_quicklist_alloc(1));
 }

 static inline pte_t *pte_alloc_one_kernel(struct mm_struct *mm,
       unsigned long addr)
 {
- return pgtable_quicklist_alloc();
+ return pgtable_quicklist_alloc(0);
 }

 static inline void pte_free(struct page *pte)
--- ./include/asm-x86_64/pgalloc.h.bckmemch 2006-04-21 11:59:36.000000000 +0400
+++ ./include/asm-x86_64/pgalloc.h 2006-09-05 12:58:17.000000000 +0400
@@ -31,12 +31,14 @@ static inline void pmd_free(pmd_t *pmd)

 static inline pmd_t *pmd_alloc_one (struct mm_struct *mm, unsigned long addr)
 {
- return (pmd_t *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT);
```

```
+ return (pmd_t *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT|
+    __GFP_BC | __GFP_BC_LIMIT);
 }

 static inline pud_t *pud_alloc_one(struct mm_struct *mm, unsigned long addr)
 {
- return (pud_t *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT);
+ return (pud_t *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT|
+    __GFP_BC | __GFP_BC_LIMIT);
 }

 static inline void pud_free (pud_t *pud)
@@ -74,7 +76,8 @@ static inline void pgd_list_del(pgd_t *p
 static inline pgd_t *pgd_alloc(struct mm_struct *mm)
 {
  unsigned boundary;
- pgd_t *pgd = (pgd_t *)__get_free_page(GFP_KERNEL|__GFP_REPEAT);
+ pgd_t *pgd = (pgd_t *)__get_free_page(GFP_KERNEL|__GFP_REPEAT|
+    __GFP_BC | __GFP_BC_LIMIT);
  if (!pgd)
   return NULL;
  pgd_list_add(pgd);
@@ -105,7 +108,8 @@ static inline pte_t *pte_alloc_one_kerne

 static inline struct page *pte_alloc_one(struct mm_struct *mm, unsigned long address)
 {
- void *p = (void *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT);
+ void *p = (void *)get_zeroed_page(GFP_KERNEL|__GFP_REPEAT|
+    __GFP_BC | __GFP_BC_LIMIT);
  if (!p)
   return NULL;
  return virt_to_page(p);
--- ./include/asm-x86_64/thread_info.h.bckmemch 2006-09-05 12:53:55.000000000 +0400
+++ ./include/asm-x86_64/thread_info.h 2006-09-05 12:58:17.000000000 +0400
@@ -78,14 +78,15 @@ static inline struct thread_info *stack_
     ({       \
  struct thread_info *ret;    \
         \
- ret = ((struct thread_info *) __get_free_pages(GFP_KERNEL,THREAD_ORDER)); \
+ ret = ((struct thread_info *) __get_free_pages(GFP_KERNEL_BC, \
+    THREAD_ORDER));   \
  if (ret)      \
   memset(ret, 0, THREAD_SIZE);   \
  ret;       \
     })
 #else
 #define alloc_thread_info(tsk) \
- ((struct thread_info *) __get_free_pages(GFP_KERNEL,THREAD_ORDER))
```

```
+ ((struct thread_info *) __get_free_pages(GFP_KERNEL_BC,THREAD_ORDER))
 #endif

 #define free_thread_info(ti) free_pages((unsigned long) (ti), THREAD_ORDER)
--- ./ipc/msgutil.c.bckmemch 2006-04-21 11:59:36.000000000 +0400
+++ ./ipc/msgutil.c 2006-09-05 12:58:17.000000000 +0400
@@ -36,7 +36,7 @@ struct msg_msg *load_msg(const void __us
  if (alen > DATALEN_MSG)
   alen = DATALEN_MSG;

- msg = (struct msg_msg *)kmalloc(sizeof(*msg) + alen, GFP_KERNEL);
+ msg = (struct msg_msg *)kmalloc(sizeof(*msg) + alen, GFP_KERNEL_BC);
  if (msg == NULL)
   return ERR_PTR(-ENOMEM);

@@ -57,7 +57,7 @@ struct msg_msg *load_msg(const void __us
   if (alen > DATALEN_SEG)
    alen = DATALEN_SEG;
   seg = (struct msg_msgseg *)kmalloc(sizeof(*seg) + alen,
-       GFP_KERNEL);
+       GFP_KERNEL_BC);
   if (seg == NULL) {
    err = -ENOMEM;
    goto out_err;
--- ./ipc/sem.c.bckmemch 2006-09-05 12:53:59.000000000 +0400
+++ ./ipc/sem.c 2006-09-05 12:58:17.000000000 +0400
@@ -1006,7 +1006,7 @@ static inline int get_undo_list(struct s

  undo_list = current->sysvsem.undo_list;
  if (!undo_list) {
-  undo_list = kzalloc(sizeof(*undo_list), GFP_KERNEL);
+  undo_list = kzalloc(sizeof(*undo_list), GFP_KERNEL_BC);
   if (undo_list == NULL)
    return -ENOMEM;
   spin_lock_init(&undo_list->lock);
@@ -1069,7 +1069,8 @@ static struct sem_undo *find_undo(struct
  ipc_rcu_getref(sma);
  sem_unlock(sma);

- new = (struct sem_undo *) kmalloc(sizeof(struct sem_undo) + sizeof(short)*nsems,
GFP_KERNEL);
+ new = (struct sem_undo *) kmalloc(sizeof(struct sem_undo) +
+   sizeof(short)*nsems, GFP_KERNEL_BC);
  if (!new) {
   ipc_lock_by_ptr(&sma->sem_perm);
   ipc_rcu_putref(sma);
@@ -1130,7 +1131,7 @@ asmlinkage long sys_semtimedop(int semid
  if (nsops > ns->sc_semopm)
```

```
   return -E2BIG;
  if(nsops > SEMOPM_FAST) {
-  sops = kmalloc(sizeof(*sops)*nsops,GFP_KERNEL);
+  sops = kmalloc(sizeof(*sops)*nsops,GFP_KERNEL_BC);
   if(sops==NULL)
    return -ENOMEM;
  }
--- ./ipc/util.c.bckmemch 2006-09-05 12:53:59.000000000 +0400
+++ ./ipc/util.c 2006-09-05 12:58:17.000000000 +0400
@@ -406,9 +406,9 @@ void* ipc_alloc(int size)
 {
  void* out;
  if(size > PAGE_SIZE)
-  out = vmalloc(size);
+  out = vmalloc_bc(size);
  else
-  out = kmalloc(size, GFP_KERNEL);
+  out = kmalloc(size, GFP_KERNEL_BC);
  return out;
 }

@@ -491,14 +491,14 @@ void* ipc_rcu_alloc(int size)
   * workqueue if necessary (for vmalloc).
   */
  if (rcu_use_vmalloc(size)) {
-  out = vmalloc(HDRLEN_VMALLOC + size);
+  out = vmalloc_bc(HDRLEN_VMALLOC + size);
   if (out) {
    out += HDRLEN_VMALLOC;
    container_of(out, struct ipc_rcu_hdr, data)->is_vmalloc = 1;
    container_of(out, struct ipc_rcu_hdr, data)->refcount = 1;
   }
  } else {
-  out = kmalloc(HDRLEN_KMALLOC + size, GFP_KERNEL);
+  out = kmalloc(HDRLEN_KMALLOC + size, GFP_KERNEL_BC);
   if (out) {
    out += HDRLEN_KMALLOC;
    container_of(out, struct ipc_rcu_hdr, data)->is_vmalloc = 0;
--- ./kernel/fork.c.bckmemch 2006-09-05 12:54:21.000000000 +0400
+++ ./kernel/fork.c 2006-09-05 12:58:17.000000000 +0400
@@ -137,7 +137,7 @@ void __init fork_init(unsigned long memp
  /* create a slab on which task_structs can be allocated */
  task_struct_cachep =
   kmem_cache_create("task_struct", sizeof(struct task_struct),
-   ARCH_MIN_TASKALIGN, SLAB_PANIC, NULL, NULL);
+   ARCH_MIN_TASKALIGN, SLAB_PANIC | SLAB_BC, NULL, NULL);
 #endif
```

```
 /*
@@ -1424,23 +1424,24 @@ void __init proc_caches_init(void)
 {
  sighand_cachep = kmem_cache_create("sighand_cache",
    sizeof(struct sighand_struct), 0,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_DESTROY_BY_RCU,
+   SLAB_HWCACHE_ALIGN | SLAB_PANIC | \
+    SLAB_DESTROY_BY_RCU | SLAB_BC,
    sighand_ctor, NULL);
  signal_cachep = kmem_cache_create("signal_cache",
    sizeof(struct signal_struct), 0,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);
  files_cachep = kmem_cache_create("files_cache",
    sizeof(struct files_struct), 0,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);
  fs_cachep = kmem_cache_create("fs_cache",
    sizeof(struct fs_struct), 0,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);
  vm_area_cachep = kmem_cache_create("vm_area_struct",
    sizeof(struct vm_area_struct), 0,
-   SLAB_PANIC, NULL, NULL);
+   SLAB_PANIC|SLAB_BC, NULL, NULL);
  mm_cachep = kmem_cache_create("mm_struct",
    sizeof(struct mm_struct), ARCH_MIN_MMSTRUCT_ALIGN,
-   SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+   SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);
 }


--- ./kernel/posix-timers.c.bckmemch 2006-09-05 12:53:59.000000000 +0400
+++ ./kernel/posix-timers.c 2006-09-05 12:58:17.000000000 +0400
@@ -242,7 +242,8 @@ static __init int init_posix_timers(void
  register_posix_clock(CLOCK_MONOTONIC, &clock_monotonic);

  posix_timers_cache = kmem_cache_create("posix_timers_cache",
-     sizeof (struct k_itimer), 0, 0, NULL, NULL);
+     sizeof (struct k_itimer), 0, SLAB_BC,
+     NULL, NULL);
  idr_init(&posix_timers_id);
  return 0;
 }
--- ./kernel/signal.c.bckmemch 2006-09-05 12:53:59.000000000 +0400
+++ ./kernel/signal.c 2006-09-05 12:58:17.000000000 +0400
@@ -2748,5 +2748,5 @@ void __init signals_init(void)
   kmem_cache_create("sigqueue",
```

```
       sizeof(struct sigqueue),
       __alignof__(struct sigqueue),
-      SLAB_PANIC, NULL, NULL);
+      SLAB_PANIC | SLAB_BC, NULL, NULL);
 }
--- ./kernel/user.c.bckmemch 2006-09-05 12:54:09.000000000 +0400
+++ ./kernel/user.c 2006-09-05 12:58:17.000000000 +0400
@@ -194,7 +194,7 @@ static int __init uid_cache_init(void)
  int n;

  uid_cachep = kmem_cache_create("uid_cache", sizeof(struct user_struct),
-   0, SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+   0, SLAB_HWCACHE_ALIGN|SLAB_PANIC|SLAB_BC, NULL, NULL);

  for(n = 0; n < UIDHASH_SZ; ++n)
   INIT_LIST_HEAD(uidhash_table + n);
--- ./mm/rmap.c.bckmemch 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/rmap.c 2006-09-05 12:58:17.000000000 +0400
@@ -179,7 +179,8 @@ static void anon_vma_ctor(void *data, st
 void __init anon_vma_init(void)
 {
  anon_vma_cachep = kmem_cache_create("anon_vma", sizeof(struct anon_vma),
-   0, SLAB_DESTROY_BY_RCU|SLAB_PANIC, anon_vma_ctor, NULL);
+   0, SLAB_DESTROY_BY_RCU|SLAB_PANIC|SLAB_BC,
+   anon_vma_ctor, NULL);
 }

 /*
--- ./mm/shmem.c.bckmemch 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/shmem.c 2006-09-05 12:58:17.000000000 +0400
@@ -368,7 +368,8 @@ static swp_entry_t *shmem_swp_alloc(stru
   }

   spin_unlock(&info->lock);
-  page = shmem_dir_alloc(mapping_gfp_mask(inode->i_mapping) | __GFP_ZERO);
+  page = shmem_dir_alloc(mapping_gfp_mask(inode->i_mapping) | \
+    __GFP_ZERO | __GFP_BC);
   if (page)
    set_page_private(page, 0);
   spin_lock(&info->lock);
```

Subject: [PATCH 8/13] BC: locked pages (core)
Posted by dev on Tue, 05 Sep 2006 15:24:18 GMT

View Forum Message <> Reply to Message

Introduce new resource BC_LOCKEDPAGES which stands for accounting
of mlock-ed user pages.

Locked pages are important to be accounted separately
as they are unreclaimable.

Pages are charged to mm_struct BC.

Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
Signed-Off-By: Kirill Korotaev <dev@sw.ru>

```
 include/bc/beancounter.h |    3 -
 include/bc/vmpages.h     |   95 +++++++++++++++++++++++++++++++++++++++++++++++
 include/linux/sched.h    |    3 +
 include/linux/shmem_fs.h |    5 ++
 kernel/bc/Makefile       |    1
 kernel/bc/beancounter.c  |    2
 kernel/bc/vmpages.c      |   75 +++++++++++++++++++++++++++++++++++++
 kernel/fork.c            |   11 +++--
 mm/shmem.c               |    4 +
 9 files changed, 195 insertions(+), 4 deletions(-)

--- ./include/bc/beancounter.h.bclockcore 2006-09-05 12:54:40.000000000 +0400
+++ ./include/bc/beancounter.h 2006-09-05 12:59:27.000000000 +0400
@@ -13,8 +13,9 @@
  */

 #define BC_KMEMSIZE 0
+#define BC_LOCKEDPAGES 1

-#define BC_RESOURCES 1
+#define BC_RESOURCES 2

 struct bc_resource_parm {
  unsigned long barrier; /* A barrier over which resource allocations
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/vmpages.h 2006-09-05 13:04:03.000000000 +0400
@@ -0,0 +1,95 @@
+/*
+ *  include/bc/vmpages.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef __BC_VMPAGES_H_
+#define __BC_VMPAGES_H_
+
```

```
+#include <bc/beancounter.h>
+#include <bc/task.h>
+
+struct mm_struct;
+struct file;
+struct shmem_inode_info;
+
+#ifdef CONFIG_BEANCOUNTERS
+int __must_check bc_memory_charge(struct mm_struct *mm, unsigned long size,
+  unsigned long vm_flags, struct file *vm_file, int strict);
+void bc_memory_uncharge(struct mm_struct *mm, unsigned long size,
+  unsigned long vm_flags, struct file *vm_file);
+
+int __must_check bc_locked_charge(struct mm_struct *mm, unsigned long size);
+void bc_locked_uncharge(struct mm_struct *mm, unsigned long size);
+
+int __must_check bc_locked_shm_charge(struct shmem_inode_info *info,
+  unsigned long size);
+void bc_locked_shm_uncharge(struct shmem_inode_info *info,
+  unsigned long size);
+
+/*
+ * mm's beancounter should be the same as the exec one
+ * of taks using this mm. thus we have two cases of its
+ * initialisation:
+ *  1. new mm is done for fork-ed task
+ *  2. new mm is done for exec-ing task
+ */
+#define mm_init_bc(mm, t) do {     \
+  (mm)->mm_bc = get_beancounter((t)->task_bc.exec_bc); \
+ } while (0)
+#define mm_free_bc(mm)  do {     \
+  put_beancounter((mm)->mm_bc);    \
+ } while (0)
+
+#define shmi_init_bc(info) do {     \
+  (info)->shm_bc = get_beancounter(get_exec_bc()); \
+ } while (0)
+#define shmi_free_bc(info) do {     \
+  put_beancounter((info)->shm_bc);   \
+ } while (0)
+
+#else /* CONFIG_BEANCOUNTERS */
+
+static inline int __must_check bc_memory_charge(struct mm_struct *mm,
+  unsigned long size, unsigned long vm_flags,
+  struct file *vm_file, int strict)
+{
```

```
+ return 0;
+}
+
+static inline void bc_memory_uncharge(struct mm_struct *mm, unsigned long size,
+  unsigned long vm_flags, struct file *vm_file)
+{
+}
+
+static inline int __must_check bc_locked_charge(struct mm_struct *mm,
+  unsigned long size)
+{
+ return 0;
+}
+
+static inline void bc_locked_uncharge(struct mm_struct *mm, unsigned long size)
+{
+}
+
+static inline int __must_check bc_locked_shm_charge(struct shmem_inode_info *i,
+  unsigned long size)
+{
+ return 0;
+}
+
+static inline void bc_locked_shm_uncharge(struct shmem_inode_info *i,
+  unsigned long size)
+{
+}
+
+#define mm_init_bc(mm, t) do { } while (0)
+#define mm_free_bc(mm)  do { } while (0)
+#define shmi_init_bc(info) do { } while (0)
+#define shmi_free_bc(info) do { } while (0)
+
+#endif /* CONFIG_BEANCOUNTERS */
+#endif
+
--- ./include/linux/sched.h.bclockcore 2006-09-05 12:54:21.000000000 +0400
+++ ./include/linux/sched.h 2006-09-05 12:59:27.000000000 +0400
@@ -358,6 +358,9 @@ struct mm_struct {
  /* aio bits */
  rwlock_t  ioctx_list_lock;
  struct kioctx  *ioctx_list;
+#ifdef CONFIG_BEANCOUNTERS
+ struct beancounter *mm_bc;
+#endif
 };
```

```
 struct sighand_struct {
--- ./include/linux/shmem_fs.h.bclockcore 2006-04-21 11:59:36.000000000 +0400
+++ ./include/linux/shmem_fs.h 2006-09-05 12:59:27.000000000 +0400
@@ -8,6 +8,8 @@

 #define SHMEM_NR_DIRECT 16

+struct beancounter;
+
 struct shmem_inode_info {
  spinlock_t  lock;
  unsigned long  flags;
@@ -19,6 +21,9 @@ struct shmem_inode_info {
  swp_entry_t  i_direct[SHMEM_NR_DIRECT]; /* first blocks */
  struct list_head swaplist; /* chain of maybes on swap */
  struct inode  vfs_inode;
+#ifdef CONFIG_BEANCOUNTERS
+ struct beancounter *shm_bc;
+#endif
 };

 struct shmem_sb_info {
--- ./kernel/bc/Makefile.bclockcore 2006-09-05 12:54:50.000000000 +0400
+++ ./kernel/bc/Makefile 2006-09-05 12:59:37.000000000 +0400
@@ -8,3 +8,4 @@ obj-y += beancounter.o
 obj-y += misc.o
 obj-y += sys.o
 obj-y += kmem.o
+obj-y += vmpages.o
--- ./kernel/bc/beancounter.c.bclockcore 2006-09-05 12:55:13.000000000 +0400
+++ ./kernel/bc/beancounter.c 2006-09-05 12:59:45.000000000 +0400
@@ -21,6 +21,7 @@ struct beancounter init_bc;

 const char *bc_rnames[] = {
  "kmemsize", /* 0 */
+ "lockedpages",
 };

 #define BC_HASH_BITS  8
@@ -232,6 +233,7 @@ static void init_beancounter_syslimits(s
  int k;

  bc->bc_parms[BC_KMEMSIZE].limit = 32 * 1024 * 1024;
+ bc->bc_parms[BC_LOCKEDPAGES].limit = 8;

  for (k = 0; k < BC_RESOURCES; k++)
   bc->bc_parms[k].barrier = bc->bc_parms[k].limit;
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
```

```
+++ ./kernel/bc/vmpages.c 2006-09-05 12:59:27.000000000 +0400
@@ -0,0 +1,75 @@
+/*
+ *  kernel/bc/vmpages.c
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/mm.h>
+#include <linux/shmem_fs.h>
+
+#include <bc/beancounter.h>
+#include <bc/vmpages.h>
+
+#include <asm/page.h>
+
+int bc_memory_charge(struct mm_struct *mm, unsigned long size,
+  unsigned long vm_flags, struct file *vm_file, int strict)
+{
+ struct beancounter *bc;
+
+ bc = mm->mm_bc;
+ size >>= PAGE_SHIFT;
+
+ if (vm_flags & VM_LOCKED)
+  if (bc_charge(bc, BC_LOCKEDPAGES, size, strict))
+   return -ENOMEM;
+ return 0;
+}
+
+void bc_memory_uncharge(struct mm_struct *mm, unsigned long size,
+  unsigned long vm_flags, struct file *vm_file)
+{
+ struct beancounter *bc;
+
+ bc = mm->mm_bc;
+ size >>= PAGE_SHIFT;
+
+ if (vm_flags & VM_LOCKED)
+  bc_uncharge(bc, BC_LOCKEDPAGES, size);
+}
+
+static inline int locked_charge(struct beancounter *bc,
+  unsigned long size)
+{
+ size >>= PAGE_SHIFT;
```

```
+ return bc_charge(bc, BC_LOCKEDPAGES, size, BC_BARRIER);
+}
+
+static inline void locked_uncharge(struct beancounter *bc,
+  unsigned long size)
+{
+ size >>= PAGE_SHIFT;
+ bc_uncharge(bc, BC_LOCKEDPAGES, size);
+}
+
+int bc_locked_charge(struct mm_struct *mm, unsigned long size)
+{
+ return locked_charge(mm->mm_bc, size);
+}
+
+void bc_locked_uncharge(struct mm_struct *mm, unsigned long size)
+{
+ locked_uncharge(mm->mm_bc, size);
+}
+
+int bc_locked_shm_charge(struct shmem_inode_info *info, unsigned long size)
+{
+ return locked_charge(info->shm_bc, size);
+}
+
+void bc_locked_shm_uncharge(struct shmem_inode_info *info, unsigned long size)
+{
+ locked_uncharge(info->shm_bc, size);
+}
--- ./kernel/fork.c.bclockcore 2006-09-05 12:58:17.000000000 +0400
+++ ./kernel/fork.c 2006-09-05 12:59:59.000000000 +0400
@@ -49,6 +49,7 @@
 #include <linux/taskstats_kern.h>

 #include <bc/task.h>
+#include <bc/vmpages.h>

 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
@@ -322,7 +323,8 @@ static inline void mm_free_pgd(struct mm

 #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 *tsk)
 {
  atomic_set(&mm->mm_users, 1);
```

```
  atomic_set(&mm->mm_count, 1);
@@ -339,6 +341,7 @@ static struct mm_struct * mm_init(struct
  mm->cached_hole_size = ~0UL;

  if (likely(!mm_alloc_pgd(mm))) {
+  mm_init_bc(mm, tsk);
   mm->def_flags = 0;
   return mm;
  }
@@ -356,7 +359,7 @@ 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;
 }
@@ -371,6 +374,7 @@ void fastcall __mmdrop(struct mm_struct
  BUG_ON(mm == &init_mm);
  mm_free_pgd(mm);
  destroy_context(mm);
+ mm_free_bc(mm);
  free_mm(mm);
 }

@@ -477,7 +481,7 @@ static struct mm_struct *dup_mm(struct t

  memcpy(mm, oldmm, sizeof(*mm));

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

  if (init_new_context(tsk, mm))
@@ -504,6 +508,7 @@ fail_nocontext:
   * because it calls destroy_context()
   */
  mm_free_pgd(mm);
+ mm_free_bc(mm);
  free_mm(mm);
  return NULL;
 }
--- ./mm/shmem.c.bclockcore 2006-09-05 12:58:17.000000000 +0400
+++ ./mm/shmem.c 2006-09-05 12:59:27.000000000 +0400
@@ -47,6 +47,8 @@
 #include <linux/migrate.h>
 #include <linux/highmem.h>
```

```
+#include <bc/vmpages.h>
+
 #include <asm/uaccess.h>
 #include <asm/div64.h>
 #include <asm/pgtable.h>
@@ -698,6 +700,7 @@ static void shmem_delete_inode(struct in
   sbinfo->free_inodes++;
   spin_unlock(&sbinfo->stat_lock);
  }
+ shmi_free_bc(info);
  clear_inode(inode);
 }

@@ -1359,6 +1362,7 @@ shmem_get_inode(struct super_block *sb,
   info = SHMEM_I(inode);
   memset(info, 0, (char *)inode - (char *)info);
   spin_lock_init(&info->lock);
+  shmi_init_bc(info);
   INIT_LIST_HEAD(&info->swaplist);

   switch (mode & S_IFMT) {
```

Subject: [PATCH 9/13] BC: locked pages (charge hooks)
Posted by dev on Tue, 05 Sep 2006 15:25:53 GMT
View Forum Message <> Reply to Message

Introduce calls to BC core over the kernel to charge locked memory.

Normaly new locked piece of memory may appear in insert_vm_struct,
but there are places (do_mmap_pgoff, dup_mmap etc) when new vma
is not inserted by insert_vm_struct(), but either link_vma-ed or
merged with some other - these places call BC code explicitly.

Plus sys_mlock[all] itself has to be patched to charge/uncharge
needed amount of pages.

Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
Signed-Off-By: Kirill Korotaev <dev@sw.ru>

```
 fs/binfmt_elf.c           |    5 ++-
 include/asm-alpha/mman.h  |    1
 include/asm-generic/mman.h |    1
 include/asm-mips/mman.h   |    1
 include/asm-parisc/mman.h |    1
```

```
 include/linux/mm.h         |    1
 mm/mlock.c                 |   21 +++++++++++++---
 mm/mmap.c                  |   59 ++++++++++++++++++++++++++++++++++++++-------
 mm/mremap.c                |   18 +++++++++++++-
 mm/shmem.c                 |   12 ++++++++-
 10 files changed, 104 insertions(+), 16 deletions(-)

--- ./fs/binfmt_elf.c.bclockcharge 2006-09-05 12:53:54.000000000 +0400
+++ ./fs/binfmt_elf.c 2006-09-05 13:08:26.000000000 +0400
@@ -360,7 +360,7 @@ static unsigned long load_elf_interp(str
  eppnt = elf_phdata;
  for (i = 0; i < interp_elf_ex->e_phnum; i++, eppnt++) {
   if (eppnt->p_type == PT_LOAD) {
-   int elf_type = MAP_PRIVATE | MAP_DENYWRITE;
+   int elf_type = MAP_PRIVATE|MAP_DENYWRITE|MAP_EXECPRIO;
    int elf_prot = 0;
    unsigned long vaddr = 0;
    unsigned long k, map_addr;
@@ -846,7 +846,8 @@ static int load_elf_binary(struct linux_
   if (elf_ppnt->p_flags & PF_X)
    elf_prot |= PROT_EXEC;

-  elf_flags = MAP_PRIVATE | MAP_DENYWRITE | MAP_EXECUTABLE;
+  elf_flags = MAP_PRIVATE | MAP_DENYWRITE |
+   MAP_EXECUTABLE | MAP_EXECPRIO;

   vaddr = elf_ppnt->p_vaddr;
   if (loc->elf_ex.e_type == ET_EXEC || load_addr_set) {
--- ./include/asm-alpha/mman.h.mapfx 2006-04-21 11:59:35.000000000 +0400
+++ ./include/asm-alpha/mman.h 2006-09-05 18:13:12.000000000 +0400
@@ -28,6 +28,7 @@
 #define MAP_NORESERVE 0x10000  /* don't check for reservations */
 #define MAP_POPULATE 0x20000  /* populate (prefault) pagetables */
 #define MAP_NONBLOCK 0x40000  /* do not block on IO */
+#define MAP_EXECPRIO 0x80000  /* charge against BC limit */

 #define MS_ASYNC 1  /* sync memory asynchronously */
 #define MS_SYNC  2  /* synchronous memory sync */
--- ./include/asm-generic/mman.h.x 2006-04-21 11:59:35.000000000 +0400
+++ ./include/asm-generic/mman.h 2006-09-05 14:02:04.000000000 +0400
@@ -19,6 +19,7 @@
 #define MAP_TYPE 0x0f  /* Mask for type of mapping */
 #define MAP_FIXED 0x10  /* Interpret addr exactly */
 #define MAP_ANONYMOUS 0x20  /* don't use a file */
+#define MAP_EXECPRIO 0x20000  /* charge agains BC_LIMIT */

 #define MS_ASYNC 1  /* sync memory asynchronously */
 #define MS_INVALIDATE 2  /* invalidate the caches */
```

```
--- ./include/asm-mips/mman.h.mapfx 2006-04-21 11:59:36.000000000 +0400
+++ ./include/asm-mips/mman.h 2006-09-05 18:13:34.000000000 +0400
@@ -46,6 +46,7 @@
 #define MAP_LOCKED 0x8000  /* pages are locked */
 #define MAP_POPULATE 0x10000  /* populate (prefault) pagetables */
 #define MAP_NONBLOCK 0x20000  /* do not block on IO */
+#define MAP_EXECPRIO 0x40000  /* charge against BC limit */

 /*
  * Flags for msync
--- ./include/asm-parisc/mman.h.mapfx 2006-04-21 11:59:36.000000000 +0400
+++ ./include/asm-parisc/mman.h 2006-09-05 18:13:47.000000000 +0400
@@ -22,6 +22,7 @@
 #define MAP_GROWSDOWN 0x8000  /* stack-like segment */
 #define MAP_POPULATE 0x10000  /* populate (prefault) pagetables */
 #define MAP_NONBLOCK 0x20000  /* do not block on IO */
+#define MAP_EXECPRIO 0x40000  /* charge against BC limit */

 #define MS_SYNC  1  /* synchronous memory sync */
 #define MS_ASYNC 2  /* sync memory asynchronously */
--- ./include/linux/mm.h.bclockcharge 2006-09-05 12:55:28.000000000 +0400
+++ ./include/linux/mm.h 2006-09-05 13:06:37.000000000 +0400
@@ -1103,6 +1103,7 @@ out:
 extern int do_munmap(struct mm_struct *, unsigned long, size_t);

 extern unsigned long do_brk(unsigned long, unsigned long);
+extern unsigned long __do_brk(unsigned long, unsigned long, int);

 /* filemap.c */
 extern unsigned long page_unuse(struct page *);
--- ./mm/mlock.c.bclockcharge 2006-04-21 11:59:36.000000000 +0400
+++ ./mm/mlock.c 2006-09-05 13:06:37.000000000 +0400
@@ -11,6 +11,7 @@
 #include <linux/mempolicy.h>
 #include <linux/syscalls.h>

+#include <bc/vmpages.h>

 static int mlock_fixup(struct vm_area_struct *vma, struct vm_area_struct **prev,
  unsigned long start, unsigned long end, unsigned int newflags)
@@ -25,6 +26,14 @@ static int mlock_fixup(struct vm_area_st
   goto out;
  }

+ if (newflags & VM_LOCKED) {
+  ret = bc_locked_charge(mm, end - start);
+  if (ret < 0) {
+   *prev = vma;
```

```
+   goto out;
+  }
+ }
+
  pgoff = vma->vm_pgoff + ((start - vma->vm_start) >> PAGE_SHIFT);
  *prev = vma_merge(mm, *prev, start, end, newflags, vma->anon_vma,
      vma->vm_file, pgoff, vma_policy(vma));
@@ -38,13 +47,13 @@ static int mlock_fixup(struct vm_area_st
  if (start != vma->vm_start) {
   ret = split_vma(mm, vma, start, 1);
   if (ret)
-   goto out;
+   goto out_uncharge;
  }

  if (end != vma->vm_end) {
   ret = split_vma(mm, vma, end, 0);
   if (ret)
-   goto out;
+   goto out_uncharge;
  }

 success:
@@ -63,13 +72,19 @@ success:
   pages = -pages;
   if (!(newflags & VM_IO))
    ret = make_pages_present(start, end);
- }
+ } else
+  bc_locked_uncharge(mm, end - start);

  vma->vm_mm->locked_vm -= pages;
 out:
  if (ret == -ENOMEM)
   ret = -EAGAIN;
  return ret;
+
+out_uncharge:
+ if (newflags & VM_LOCKED)
+  bc_locked_uncharge(mm, end - start);
+ goto out;
 }

 static int do_mlock(unsigned long start, size_t len, int on)
--- ./mm/mmap.c.bclockcharge 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/mmap.c 2006-09-05 13:07:13.000000000 +0400
@@ -26,6 +26,8 @@
 #include <linux/mempolicy.h>
```

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

+#include <bc/vmpages.h>
+
 #include <asm/uaccess.h>
 #include <asm/cacheflush.h>
 #include <asm/tlb.h>
@@ -220,6 +222,10 @@ static struct vm_area_struct *remove_vma
  struct vm_area_struct *next = vma->vm_next;

  might_sleep();
+
+ bc_memory_uncharge(vma->vm_mm, vma->vm_end - vma->vm_start,
+    vma->vm_flags, vma->vm_file);
+
  if (vma->vm_ops && vma->vm_ops->close)
   vma->vm_ops->close(vma);
  if (vma->vm_file)
@@ -267,7 +273,7 @@ asmlinkage unsigned long sys_brk(unsigne
   goto out;

  /* Ok, looks good - let it rip. */
- if (do_brk(oldbrk, newbrk-oldbrk) != oldbrk)
+ if (__do_brk(oldbrk, newbrk-oldbrk, BC_BARRIER) != oldbrk)
   goto out;
 set_brk:
  mm->brk = brk;
@@ -1047,6 +1053,11 @@ munmap_back:
   }
  }

+ error = bc_memory_charge(mm, len, vm_flags, file,
+    flags & MAP_EXECPRIO ? BC_LIMIT : BC_BARRIER);
+ if (error)
+  goto charge_fail;
+
  /*
   * Can we just expand an old private anonymous mapping?
   * The VM_SHARED test is necessary because shmem_zero_setup
@@ -1160,6 +1171,8 @@ unmap_and_free_vma:
 free_vma:
  kmem_cache_free(vm_area_cachep, vma);
 unacct_error:
+ bc_memory_uncharge(mm, len, vm_flags, file);
+charge_fail:
  if (charged)
   vm_unacct_memory(charged);
  return error;
```

```
@@ -1489,12 +1502,16 @@ static int acct_stack_growth(struct vm_a
    return -ENOMEM;
  }

+ if (bc_memory_charge(mm, grow << PAGE_SHIFT,
+     vma->vm_flags, vma->vm_file, BC_LIMIT))
+  goto err_ch;
+
  /*
   * Overcommit..  This must be the final test, as it will
   * update security statistics.
   */
  if (security_vm_enough_memory(grow))
-  return -ENOMEM;
+  goto err_acct;

  /* Ok, everything looks good - let it rip */
  mm->total_vm += grow;
@@ -1502,6 +1519,11 @@ static int acct_stack_growth(struct vm_a
   mm->locked_vm += grow;
  vm_stat_account(mm, vma->vm_flags, vma->vm_file, grow);
  return 0;
+
+err_acct:
+ bc_memory_uncharge(mm, grow << PAGE_SHIFT, vma->vm_flags, vma->vm_file);
+err_ch:
+ return -ENOMEM;
 }

 #if defined(CONFIG_STACK_GROWSUP) || defined(CONFIG_IA64)
@@ -1857,7 +1879,7 @@ static inline void verify_mm_writelocked
  *  anonymous maps.  eventually we may be able to do some
  *  brk-specific accounting here.
  */
-unsigned long do_brk(unsigned long addr, unsigned long len)
+unsigned long __do_brk(unsigned long addr, unsigned long len, int bc_strict)
 {
  struct mm_struct * mm = current->mm;
  struct vm_area_struct * vma, * prev;
@@ -1914,6 +1936,9 @@ unsigned long do_brk(unsigned long addr,

  flags = VM_DATA_DEFAULT_FLAGS | VM_ACCOUNT | mm->def_flags;

+ if (bc_memory_charge(mm, len, flags, NULL, bc_strict))
+  goto out_unacct;
+
  /* Can we just expand an old private anonymous mapping? */
  if (vma_merge(mm, prev, addr, addr + len, flags,
```

```
      NULL, NULL, pgoff, NULL))
@@ -1923,10 +1948,8 @@ unsigned long do_brk(unsigned long addr,
   * create a vma struct for an anonymous mapping
   */
  vma = kmem_cache_zalloc(vm_area_cachep, GFP_KERNEL);
- if (!vma) {
-  vm_unacct_memory(len >> PAGE_SHIFT);
-  return -ENOMEM;
- }
+ if (!vma)
+  goto out_uncharge;

  vma->vm_mm = mm;
  vma->vm_start = addr;
@@ -1943,6 +1966,17 @@ out:
   make_pages_present(addr, addr + len);
  }
  return addr;
+
+out_uncharge:
+ bc_memory_uncharge(mm, len, flags, NULL);
+out_unacct:
+ vm_unacct_memory(len >> PAGE_SHIFT);
+ return -ENOMEM;
+}
+
+unsigned long do_brk(unsigned long addr, unsigned long len)
+{
+ return __do_brk(addr, len, BC_LIMIT);
 }

 EXPORT_SYMBOL(do_brk);
@@ -2005,9 +2039,18 @@ int insert_vm_struct(struct mm_struct *
   return -ENOMEM;
  if ((vma->vm_flags & VM_ACCOUNT) &&
      security_vm_enough_memory(vma_pages(vma)))
-  return -ENOMEM;
+  goto err_acct;
+ if (bc_memory_charge(mm, vma->vm_end - vma->vm_start,
+    vma->vm_flags, vma->vm_file, BC_LIMIT))
+  goto err_charge;
  vma_link(mm, vma, prev, rb_link, rb_parent);
  return 0;
+
+err_charge:
+ if (vma->vm_flags & VM_ACCOUNT)
+  vm_unacct_memory(vma_pages(vma));
+err_acct:
```

```
+ return -ENOMEM;
 }

 /*
--- ./mm/mremap.c.bclockcharge 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/mremap.c 2006-09-05 13:06:37.000000000 +0400
@@ -19,6 +19,8 @@
 #include <linux/security.h>
 #include <linux/syscalls.h>

+#include <bc/vmpages.h>
+
 #include <asm/uaccess.h>
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>
@@ -350,6 +352,13 @@ unsigned long do_mremap(unsigned long ad
    goto out_nc;
  }

+ if (new_len > old_len) {
+  ret = bc_memory_charge(mm, new_len - old_len,
+    vma->vm_flags, vma->vm_file, BC_BARRIER);
+  if (ret)
+   goto out;
+ }
+
  /* old_len exactly to the end of the area..
   * And we're not relocating the area.
   */
@@ -374,7 +383,7 @@ unsigned long do_mremap(unsigned long ad
          addr + new_len);
    }
    ret = addr;
-   goto out;
+   goto out_ch;
   }
  }

@@ -393,10 +402,15 @@ unsigned long do_mremap(unsigned long ad
       vma->vm_pgoff, map_flags);
    ret = new_addr;
    if (new_addr & ~PAGE_MASK)
-    goto out;
+    goto out_ch;
   }
   ret = move_vma(vma, addr, old_len, new_len, new_addr);
  }
+out_ch:
```

```
+ if (ret & ~PAGE_MASK)
+  if (new_len > old_len)
+   bc_memory_uncharge(mm, new_len - old_len,
+     vma->vm_flags, vma->vm_file);
 out:
  if (ret & ~PAGE_MASK)
   vm_unacct_memory(charged);
--- ./mm/shmem.c.bclockcharge 2006-09-05 12:59:27.000000000 +0400
+++ ./mm/shmem.c 2006-09-05 13:06:37.000000000 +0400
@@ -1309,21 +1309,31 @@ int shmem_lock(struct file *file, int lo
  struct inode *inode = file->f_dentry->d_inode;
  struct shmem_inode_info *info = SHMEM_I(inode);
  int retval = -ENOMEM;
+ unsigned long size;
+
+ size = (inode->i_size + PAGE_SIZE - 1) >> PAGE_SHIFT;

  spin_lock(&info->lock);
  if (lock && !(info->flags & VM_LOCKED)) {
-  if (!user_shm_lock(inode->i_size, user))
+  if (bc_locked_shm_charge(info, size))
    goto out_nomem;
+  if (!user_shm_lock(inode->i_size, user))
+   goto out_uncharge;
   info->flags |= VM_LOCKED;
  }
  if (!lock && (info->flags & VM_LOCKED) && user) {
   user_shm_unlock(inode->i_size, user);
+  bc_locked_shm_uncharge(info, size);
   info->flags &= ~VM_LOCKED;
  }
  retval = 0;
 out_nomem:
  spin_unlock(&info->lock);
  return retval;
+
+out_uncharge:
+ bc_locked_shm_uncharge(info, size);
+ goto out_nomem;
 }

 int shmem_mmap(struct file *file, struct vm_area_struct *vma)
```

Subject: [PATCH 10/13] BC: privvm pages
Posted by dev on Tue, 05 Sep 2006 15:26:58 GMT
View Forum Message <> Reply to Message

This patch instroduces new resource - BC_PRIVVMPAGES.
It is an upper estimation of currently used physical memory.

There are different approaches to user pages control:
a) account all the mappings on mmap/brk and reject as
   soon as the sum of VMA's lengths reaches the barrier.

   This approach is very bad as applications always map
   more than they really use, very often MUCH more.

b) account only the really used memory and reject as
   soon as RSS reaches the limit.

   This approach is not good either as user space pages are
   allocated in page fault handler and the only way to reject
   allocation is to kill the task.

   Comparing to previous scenarion this is much worse as
   application won't even be able to terminate gracefully.

c) account a part of memory on mmap/brk and reject there,
   and account the rest of the memory in page fault handlers
   without any rejects.
   This type of accounting is used in UBC.

d) account physical memory and behave like a standalone
   kernel - reclaim user memory when run out of it.

   This type of memory control is to be introduced later
   as an addition to c). UBC provides all the needed
   statistics for this (physical memory, swap pages etc.)

Privvmpages accounting is described in details in
http://wiki.openvz.org/User_pages_accounting

A note about sys_mprotect: as it can change mapping state from
BC_VM_PRIVATE to !BC_VM_PRIVATE and vice-versa appropriate amount of
pages is (un)charged in mprotect_fixup.

Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
Signed-Off-By: Kirill Korotaev <dev@sw.ru>

---

 include/bc/beancounter.h | 3 +-
 include/bc/vmpages.h | 44 ++
 kernel/bc/beancounter.c | 2 +
 kernel/bc/vmpages.c | 53 ++---

```
 kernel/fork.c          |    9 +++++++
 mm/mprotect.c          |   17 +++++++++++++++-
 mm/shmem.c             |    7 ++++++
 7 files changed, 129 insertions(+), 6 deletions(-)

--- ./include/bc/beancounter.h.bcprivvm 2006-09-05 12:59:27.000000000 +0400
+++ ./include/bc/beancounter.h 2006-09-05 13:17:50.000000000 +0400
@@ -14,8 +14,9 @@

 #define BC_KMEMSIZE 0
 #define BC_LOCKEDPAGES 1
+#define BC_PRIVVMPAGES 2

-#define BC_RESOURCES 2
+#define BC_RESOURCES 3

 struct bc_resource_parm {
  unsigned long barrier; /* A barrier over which resource allocations
--- ./include/bc/vmpages.h.bcprivvm 2006-09-05 13:04:03.000000000 +0400
+++ ./include/bc/vmpages.h 2006-09-05 13:38:07.000000000 +0400
@@ -8,6 +8,8 @@
 #ifndef __BC_VMPAGES_H_
 #define __BC_VMPAGES_H_

+#include <linux/mm.h>
+
 #include <bc/beancounter.h>
 #include <bc/task.h>

@@ -15,12 +17,37 @@ struct mm_struct;
 struct file;
 struct shmem_inode_info;

+/*
+ * sys_mprotect() can change mapping state form private to
+ * shared and vice-versa. Thus rescharging is needed, but
+ * with the following rules:
+ * 1. No state change   : nothing to be done at all;
+ * 2. shared -> private : need to charge before operation starts
+ *                        and roll back on error path;
+ * 3. private -> shared : need to uncharge after successfull state
+ *                        change. Uncharging first and charging back
+ *                        on error path isn't good as charge will have
+ *                        to be BC_FORCE and thus can potentially create
+ *                        an overcharged privvmpages.
+ */
+#define BC_NOCHARGE 0
+#define BC_UNCHARGE 1 /* private -> shared */
```

```
+#define BC_CHARGE 2 /* shared -> private */
+
+#define BC_VM_PRIVATE(flags, file) ( ((flags) & VM_WRITE) ? \
+   ( (file) == NULL || !((flags) & VM_SHARED) ) : 0 )
+
 #ifdef CONFIG_BEANCOUNTERS
 int __must_check bc_memory_charge(struct mm_struct *mm, unsigned long size,
   unsigned long vm_flags, struct file *vm_file, int strict);
 void bc_memory_uncharge(struct mm_struct *mm, unsigned long size,
   unsigned long vm_flags, struct file *vm_file);

+int __must_check bc_privvm_recharge(unsigned long old_flags,
+  unsigned long new_flags, struct file *vm_file);
+int __must_check bc_privvm_charge(struct mm_struct *mm, unsigned long size);
+void bc_privvm_uncharge(struct mm_struct *mm, unsigned long size);
+
 int __must_check bc_locked_charge(struct mm_struct *mm, unsigned long size);
 void bc_locked_uncharge(struct mm_struct *mm, unsigned long size);

@@ -64,6 +91,23 @@ static inline void bc_memory_uncharge(st
 {
 }

+static inline int __must_check bc_privvm_recharge(unsigned long old_flags,
+  unsigned long new_flags, struct file *vm_file)
+{
+ return BC_NOCHARGE;
+}
+
+static inline int __must_check bc_privvm_charge(struct mm_struct *mm,
+  unsigned long size)
+{
+ return 0;
+}
+
+static inline void bc_privvm_uncharge(struct mm_struct *mm,
+  unsigned long size)
+{
+}
+
 static inline int __must_check bc_locked_charge(struct mm_struct *mm,
   unsigned long size)
 {
--- ./kernel/bc/beancounter.c.bcprivvm 2006-09-05 12:59:45.000000000 +0400
+++ ./kernel/bc/beancounter.c 2006-09-05 13:17:50.000000000 +0400
@@ -22,6 +22,7 @@ struct beancounter init_bc;
 const char *bc_rnames[] = {
  "kmemsize", /* 0 */
```

```
  "lockedpages",
+ "privvmpages",
 };

 #define BC_HASH_BITS  8
@@ -234,6 +235,7 @@ static void init_beancounter_syslimits(s

  bc->bc_parms[BC_KMEMSIZE].limit = 32 * 1024 * 1024;
  bc->bc_parms[BC_LOCKEDPAGES].limit = 8;
+ bc->bc_parms[BC_PRIVVMPAGES].limit = BC_MAXVALUE;

  for (k = 0; k < BC_RESOURCES; k++)
   bc->bc_parms[k].barrier = bc->bc_parms[k].limit;
--- ./kernel/bc/vmpages.c.bcprivvm 2006-09-05 12:59:27.000000000 +0400
+++ ./kernel/bc/vmpages.c 2006-09-05 13:28:16.000000000 +0400
@@ -18,26 +18,73 @@ int bc_memory_charge(struct mm_struct *m
   unsigned long vm_flags, struct file *vm_file, int strict)
 {
  struct beancounter *bc;
+ unsigned long flags;

  bc = mm->mm_bc;
  size >>= PAGE_SHIFT;

+ spin_lock_irqsave(&bc->bc_lock, flags);
  if (vm_flags & VM_LOCKED)
-  if (bc_charge(bc, BC_LOCKEDPAGES, size, strict))
-   return -ENOMEM;
+  if (bc_charge_locked(bc, BC_LOCKEDPAGES, size, strict))
+   goto err_locked;
+ if (BC_VM_PRIVATE(vm_flags, vm_file))
+  if (bc_charge_locked(bc, BC_PRIVVMPAGES, size, strict))
+   goto err_privvm;
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
  return 0;
+
+err_privvm:
+ bc_uncharge_locked(bc, BC_LOCKEDPAGES, size);
+err_locked:
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+ return -ENOMEM;
 }

 void bc_memory_uncharge(struct mm_struct *mm, unsigned long size,
   unsigned long vm_flags, struct file *vm_file)
 {
  struct beancounter *bc;
+ unsigned long flags;
```

```
  bc = mm->mm_bc;
  size >>= PAGE_SHIFT;

+ spin_lock_irqsave(&bc->bc_lock, flags);
  if (vm_flags & VM_LOCKED)
-  bc_uncharge(bc, BC_LOCKEDPAGES, size);
+  bc_uncharge_locked(bc, BC_LOCKEDPAGES, size);
+ if (BC_VM_PRIVATE(vm_flags, vm_file))
+  bc_uncharge_locked(bc, BC_PRIVVMPAGES, size);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+}
+
+int bc_privvm_recharge(unsigned long vm_flags_old, unsigned long vm_flags_new,
+  struct file *vm_file)
+{
+ int priv_old, priv_new;
+
+ priv_old = (BC_VM_PRIVATE(vm_flags_old, vm_file) ? 1 : 0);
+ priv_new = (BC_VM_PRIVATE(vm_flags_new, vm_file) ? 1 : 0);
+
+ if (priv_old == priv_new)
+  return BC_NOCHARGE;
+
+ return priv_new ? BC_CHARGE : BC_UNCHARGE;
+}
+
+int bc_privvm_charge(struct mm_struct *mm, unsigned long size)
+{
+ struct beancounter *bc;
+
+ bc = mm->mm_bc;
+ bc_charge(bc, BC_PRIVVMPAGES, size >> PAGE_SHIFT);
+}
+
+void bc_privvm_uncharge(struct mm_struct *mm, unsigned long size)
+{
+ struct beancounter *bc;
+
+ bc = mm->mm_bc;
+ bc_uncharge(bc, BC_PRIVVMPAGES, size >> PAGE_SHIFT);
 }

 static inline int locked_charge(struct beancounter *bc,
--- ./kernel/fork.c.bcprivvm 2006-09-05 13:17:15.000000000 +0400
+++ ./kernel/fork.c 2006-09-05 13:23:27.000000000 +0400
@@ -236,9 +236,13 @@ static inline int dup_mmap(struct mm_str
     goto fail_nomem;
```

```
    charge = len;
   }
+  if (bc_memory_charge(mm, mpnt->vm_end - mpnt->vm_start,
+      mpnt->vm_flags & ~VM_LOCKED,
+      mpnt->vm_file, BC_LIMIT) < 0)
+    goto fail_nomem;
   tmp = kmem_cache_alloc(vm_area_cachep, SLAB_KERNEL);
   if (!tmp)
-    goto fail_nomem;
+    goto fail_alloc;
   *tmp = *mpnt;
   pol = mpol_copy(vma_policy(mpnt));
   retval = PTR_ERR(pol);
@@ -292,6 +296,9 @@ out:
  return retval;
 fail_nomem_policy:
  kmem_cache_free(vm_area_cachep, tmp);
+fail_alloc:
+ bc_memory_uncharge(mm, mpnt->vm_end - mpnt->vm_start,
+    mpnt->vm_flags & ~VM_LOCKED, mpnt->vm_file);
 fail_nomem:
  retval = -ENOMEM;
  vm_unacct_memory(charge);
--- ./mm/mprotect.c.bcprivvm 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/mprotect.c 2006-09-05 13:27:40.000000000 +0400
@@ -21,6 +21,7 @@
 #include <linux/syscalls.h>
 #include <linux/swap.h>
 #include <linux/swapops.h>
+#include <bc/vmpages.h>
 #include <asm/uaccess.h>
 #include <asm/pgtable.h>
 #include <asm/cacheflush.h>
@@ -139,12 +140,19 @@ mprotect_fixup(struct vm_area_struct *vm
  pgoff_t pgoff;
  int error;
  int dirty_accountable = 0;
+ int recharge;

  if (newflags == oldflags) {
   *pprev = vma;
   return 0;
  }

+ recharge = bc_privvm_recharge(oldflags, newflags, vma->vm_file);
+ if (recharge == BC_CHARGE) {
+  if (bc_privvm_charge(mm, end - start))
+   return -ENOMEM;
```

```
+ }
+
  /*
   * If we make a private mapping writable we increase our commit;
   * but (without finer accounting) cannot reduce our commit if we
@@ -157,8 +165,9 @@ mprotect_fixup(struct vm_area_struct *vm
  if (newflags & VM_WRITE) {
   if (!(oldflags & (VM_ACCOUNT|VM_WRITE|VM_SHARED))) {
    charged = nrpages;
+   error = -ENOMEM;
    if (security_vm_enough_memory(charged))
-    return -ENOMEM;
+    goto fail_acct;
    newflags |= VM_ACCOUNT;
   }
  }
@@ -205,12 +213,18 @@ success:
   hugetlb_change_protection(vma, start, end, vma->vm_page_prot);
  else
   change_protection(vma, start, end, vma->vm_page_prot, dirty_accountable);
+
+ if (recharge == BC_UNCHARGE)
+  bc_privvm_uncharge(mm, end - start);
  vm_stat_account(mm, oldflags, vma->vm_file, -nrpages);
  vm_stat_account(mm, newflags, vma->vm_file, nrpages);
  return 0;

 fail:
  vm_unacct_memory(charged);
+fail_acct:
+ if (recharge == BC_CHARGE)
+  bc_privvm_uncharge(mm, end - start);
  return error;
 }

--- ./mm/shmem.c.bcprivvm 2006-09-05 13:06:37.000000000 +0400
+++ ./mm/shmem.c 2006-09-05 13:39:26.000000000 +0400
@@ -2363,6 +2363,13 @@ int shmem_zero_setup(struct vm_area_stru

  if (vma->vm_file)
   fput(vma->vm_file);
+ else if (vma->vm_flags & VM_WRITE)
+  /*
+   * this means that mapping was considered to be private
+   * in do_mmap_pgoff, but now it becomes non-private, as
+   * file is attached to the vma.
+   */
+  bc_privvm_uncharge(vma->vm_mm, size);
```

```
  vma->vm_file = file;
  vma->vm_ops = &shmem_vm_ops;
  return 0;
```

Subject: [PATCH 11/13] BC: vmrss (preparations)
Posted by dev on Tue, 05 Sep 2006 15:28:05 GMT
View Forum Message <> Reply to Message

This patch does simple things:
- intruduces an bc_magic field on beancunter to make sure
  union on struct page is correctly used in next patches
- adds nr_beancounters
- adds unused_privvmpages variable (counter of privvm pages
  which are not mapped into VM address space and thus potentially
  can be allocated later)

This is needed by vmrss accounting and is done to make patch reviewing
simpler.

Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
Signed-Off-By: Kirill Korotaev <dev@sw.ru>

```
 include/bc/beancounter.h |   13 ++++++++++++
 include/bc/vmpages.h     |    2 ++
 kernel/bc/beancounter.c  |    5 +++++
 kernel/bc/kmem.c         |    1 +
 kernel/bc/vmpages.c      |   44 ++++++++++++++++++++++++++++++++++++++++++++----
 5 files changed, 61 insertions(+), 4 deletions(-)

--- ./include/bc/beancounter.h.bcvmrssprep 2006-09-05 13:17:50.000000000 +0400
+++ ./include/bc/beancounter.h 2006-09-05 13:44:33.000000000 +0400
@@ -45,6 +45,13 @@ struct bc_resource_parm {
 #define BC_MAXVALUE LONG_MAX

 /*
+ * This magic is used to distinuish user beancounter and pages beancounter
+ * in struct page. page_ub and page_bc are placed in union and MAGIC
+ * ensures us that we don't use pbc as ubc in bc_page_uncharge().
+ */
+#define BC_MAGIC               0x62756275UL
+
+/*
  * Resource management structures
  * Serialization issues:
  *   beancounter list management is protected via bc_hash_lock
```

```
@@ -54,11 +61,13 @@ struct bc_resource_parm {
  */

 struct beancounter {
+ unsigned long  bc_magic;
  atomic_t  bc_refcount;
  spinlock_t  bc_lock;
  bcid_t   bc_id;
  struct hlist_node hash;

+ unsigned long  unused_privvmpages;
  /* resources statistics and settings */
  struct bc_resource_parm bc_parms[BC_RESOURCES];
 };
@@ -74,6 +83,8 @@ enum bc_severity { BC_BARRIER, BC_LIMIT,

 #ifdef CONFIG_BEANCOUNTERS

+extern unsigned int nr_beancounters = 1;
+
 /*
  * These functions tune minheld and maxheld values for a given
  * resource when held value changes
@@ -137,6 +137,8 @@ extern const char *bc_rnames[];

 #else /* CONFIG_BEANCOUNTERS */

+#define nr_beancounters 0
+
 #define beancounter_findcreate(id, f)   (NULL)
 #define get_beancounter(bc)    (NULL)
 #define put_beancounter(bc)    do { } while (0)
--- ./include/bc/vmpages.h.bcvmrssprep 2006-09-05 13:38:07.000000000 +0400
+++ ./include/bc/vmpages.h 2006-09-05 13:40:21.000000000 +0400
@@ -77,6 +77,8 @@ void bc_locked_shm_uncharge(struct shmem
   put_beancounter((info)->shm_bc);   \
  } while (0)

+void bc_update_privvmpages(struct beancounter *bc);
+
 #else /* CONFIG_BEANCOUNTERS */

 static inline int __must_check bc_memory_charge(struct mm_struct *mm,
--- ./kernel/bc/beancounter.c.bcvmrssprep 2006-09-05 13:17:50.000000000 +0400
+++ ./kernel/bc/beancounter.c 2006-09-05 13:44:53.000000000 +0400
@@ -19,6 +19,8 @@ static void init_beancounter_struct(stru

 struct beancounter init_bc;
```

```
+unsigned int nr_beancounters;
+
 const char *bc_rnames[] = {
  "kmemsize", /* 0 */
  "lockedpages",
@@ -88,6 +90,7 @@ retry:

 out_install:
  hlist_add_head(&new_bc->hash, slot);
+ nr_beancounters++;
  spin_unlock_irqrestore(&bc_hash_lock, flags);
 out:
  return new_bc;
@@ -110,6 +113,7 @@ void put_beancounter(struct beancounter
     bc->bc_parms[i].held, bc_rnames[i]);

  hlist_del(&bc->hash);
+ nr_beancounters--;
  spin_unlock_irqrestore(&bc_hash_lock, flags);

  kmem_cache_free(bc_cachep, bc);
@@ -214,6 +218,7 @@ EXPORT_SYMBOL_GPL(bc_uncharge);

 static void init_beancounter_struct(struct beancounter *bc, bcid_t id)
 {
+ bc->bc_magic = BC_MAGIC;
  atomic_set(&bc->bc_refcount, 1);
  spin_lock_init(&bc->bc_lock);
  bc->bc_id = id;
--- ./kernel/bc/kmem.c.bcvmrssprep 2006-09-05 12:54:40.000000000 +0400
+++ ./kernel/bc/kmem.c 2006-09-05 13:40:21.000000000 +0400
@@ -79,6 +79,7 @@ void bc_page_uncharge(struct page *page,
  if (bc == NULL)
   return;

+ BUG_ON(bc->bc_magic != BC_MAGIC);
  bc_uncharge(bc, BC_KMEMSIZE, PAGE_SIZE << order);
  put_beancounter(bc);
  page_bc(page) = NULL;
--- ./kernel/bc/vmpages.c.bcvmrssprep 2006-09-05 13:28:16.000000000 +0400
+++ ./kernel/bc/vmpages.c 2006-09-05 13:45:34.000000000 +0400
@@ -14,6 +14,34 @@

 #include <asm/page.h>

+void bc_update_privvmpages(struct beancounter *bc)
+{
```

```
+ bc->bc_parms[BC_PRIVVMPAGES].held = bc->unused_privvmpages;
+ bc_adjust_minheld(bc, BC_PRIVVMPAGES);
+ bc_adjust_maxheld(bc, BC_PRIVVMPAGES);
+}
+
+static inline int privvm_charge(struct beancounter *bc, unsigned long sz,
+  int strict)
+{
+ if (bc_charge_locked(bc, BC_PRIVVMPAGES, sz, strict))
+  return -ENOMEM;
+
+ bc->unused_privvmpages += sz;
+ return 0;
+}
+
+static inline void privvm_uncharge(struct beancounter *bc, unsigned long sz)
+{
+ if (unlikely(bc->unused_privvmpages < sz)) {
+  printk("BC: overuncharging %d unused pages: val %lu held %lu\n",
+    bc->bc_id, sz, bc->unused_privvmpages);
+  sz = bc->unused_privvmpages;
+ }
+ bc->unused_privvmpages -= sz;
+ bc_update_privvmpages(bc);
+}
+
 int bc_memory_charge(struct mm_struct *mm, unsigned long size,
   unsigned long vm_flags, struct file *vm_file, int strict)
 {
@@ -28,7 +56,7 @@ int bc_memory_charge(struct mm_struct *m
   if (bc_charge_locked(bc, BC_LOCKEDPAGES, size, strict))
    goto err_locked;
  if (BC_VM_PRIVATE(vm_flags, vm_file))
-  if (bc_charge_locked(bc, BC_PRIVVMPAGES, size, strict))
+  if (privvm_charge(bc, size, strict))
    goto err_privvm;
  spin_unlock_irqrestore(&bc->bc_lock, flags);
  return 0;
@@ -53,7 +81,7 @@ void bc_memory_uncharge(struct mm_struct
  if (vm_flags & VM_LOCKED)
   bc_uncharge_locked(bc, BC_LOCKEDPAGES, size);
  if (BC_VM_PRIVATE(vm_flags, vm_file))
-  bc_uncharge_locked(bc, BC_PRIVVMPAGES, size);
+  privvm_uncharge(bc, size);
  spin_unlock_irqrestore(&bc->bc_lock, flags);
 }

@@ -73,18 +101,26 @@ int bc_privvm_recharge(unsigned long vm_
```

```
 int bc_privvm_charge(struct mm_struct *mm, unsigned long size)
 {
+ int ret;
  struct beancounter *bc;
+ unsigned long flags;

  bc = mm->mm_bc;
- bc_charge(bc, BC_PRIVVMPAGES, size >> PAGE_SHIFT);
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ ret = privvm_charge(bc, size >> PAGE_SHIFT, BC_BARRIER);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+ return ret;
 }

 void bc_privvm_uncharge(struct mm_struct *mm, unsigned long size)
 {
  struct beancounter *bc;
+ unsigned long flags;

  bc = mm->mm_bc;
- bc_uncharge(bc, BC_PRIVVMPAGES, size >> PAGE_SHIFT);
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ privvm_uncharge(bc, size >> PAGE_SHIFT);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
 }

 static inline int locked_charge(struct beancounter *bc,
```

Subject: [PATCH 12/13] BC: vmrss (core)

Posted by dev on Tue, 05 Sep 2006 15:28:58 GMT

View Forum Message <> Reply to Message

This is the core of vmrss accounting.

The main introduced object is page_beancounter.
It ties together page and BCs which use the page.
This allows correctly account fractions of memory shared
between BCs (http://wiki.openvz.org/RSS_fractions_accounting)

Accounting API:
1. bc_alloc_rss_counter() allocates a tie between page and BC
2. bc_free_rss_counter frees it.

(1) and (2) must be done each time a page is about
to be added to someone's rss.

3. When page is touched by BC (i.e. by any task which mm belongs to BC)
   page is bc_vmrss_page_add()-ed to that BC. Touching page leads
   to subtracting it from unused_prvvmpages and adding to held_pages.
4. When page is unmapped from BC it is bc_vmrss_page_del()-ed from it.

5. When task forks all it's mapped pages must be bc_vmrss_page_dup()-ed.
   i.e. page beancounter reference counter must be increased.

6. Some pages (former PGReserved) must be added to rss, but without
   having a reference on it. These pages are bc_vmrss_page_add_noref()-ed.

Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
Signed-Off-By: Kirill Korotaev <dev@sw.ru>

```
 include/bc/beancounter.h |    3
 include/bc/vmpages.h     |    4
 include/bc/vmrss.h       |   72 ++++++
 include/linux/mm.h       |    6
 include/linux/shmem_fs.h |    2
 init/main.c              |    2
 kernel/bc/Kconfig        |    9
 kernel/bc/Makefile       |    1
 kernel/bc/beancounter.c  |    9
 kernel/bc/vmpages.c      |    7
 kernel/bc/vmrss.c        |  508 +++++++++++++++++++++++++++++++++++++++++++++++++
 mm/shmem.c               |    6
 12 files changed, 627 insertions(+), 2 deletions(-)

--- ./include/bc/beancounter.h.bcrsscore 2006-09-05 13:44:33.000000000 +0400
+++ ./include/bc/beancounter.h 2006-09-05 13:50:29.000000000 +0400
@@ -68,6 +68,9 @@ struct beancounter {
  struct hlist_node hash;

  unsigned long  unused_privvmpages;
+#ifdef CONFIG_BEANCOUNTERS_RSS
+ unsigned long long rss_pages;
+#endif
  /* resources statistics and settings */
  struct bc_resource_parm bc_parms[BC_RESOURCES];
 };
--- ./include/bc/vmpages.h.bcrsscore 2006-09-05 13:40:21.000000000 +0400
+++ ./include/bc/vmpages.h 2006-09-05 13:46:35.000000000 +0400
@@ -77,6 +77,8 @@ void bc_locked_shm_uncharge(struct shmem
   put_beancounter((info)->shm_bc);   \
  } while (0)
```

```
+#define mm_same_bc(mm1, mm2) ((mm1)->mm_bc == (mm2)->mm_bc)
+
 void bc_update_privvmpages(struct beancounter *bc);

 #else /* CONFIG_BEANCOUNTERS */
@@ -136,6 +138,8 @@ static inline void bc_locked_shm_uncharg
 #define shmi_init_bc(info) do { } while (0)
 #define shmi_free_bc(info) do { } while (0)

+#define mm_same_bc(mm1, mm2) (1)
+
 #endif /* CONFIG_BEANCOUNTERS */
 #endif

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/vmrss.h 2006-09-05 13:50:25.000000000 +0400
@@ -0,0 +1,72 @@
+/*
+ *  include/ub/vmrss.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef __BC_VMRSS_H_
+#define __BC_VMRSS_H_
+
+struct page_beancounter;
+
+struct page;
+struct mm_struct;
+struct vm_area_struct;
+
+/* values that represens page's 'weight' in bc rss accounting */
+#define PB_PAGE_WEIGHT_SHIFT 24
+#define PB_PAGE_WEIGHT (1 << PB_PAGE_WEIGHT_SHIFT)
+/* page obtains one more reference within beancounter */
+#define PB_COPY_SAME ((struct page_beancounter *)-1)
+
+#ifdef CONFIG_BEANCOUNTERS_RSS
+
+struct page_beancounter * __must_check bc_alloc_rss_counter(void);
+struct page_beancounter * __must_check bc_alloc_rss_counter_list(long num,
+  struct page_beancounter *list);
+
+void bc_free_rss_counter(struct page_beancounter *rc);
+
+void bc_vmrss_page_add(struct page *pg, struct mm_struct *mm,
```

```
+  struct vm_area_struct *vma, struct page_beancounter **ppb);
+void bc_vmrss_page_del(struct page *pg, struct mm_struct *mm,
+  struct vm_area_struct *vma);
+void bc_vmrss_page_dup(struct page *pg, struct mm_struct *mm,
+  struct vm_area_struct *vma, struct page_beancounter **ppb);
+void bc_vmrss_page_add_noref(struct page *pg, struct mm_struct *mm,
+  struct vm_area_struct *vma);
+
+unsigned long mm_rss_pages(struct mm_struct *mm, unsigned long start,
+  unsigned long end);
+
+void bc_init_rss(void);
+
+#else /* CONFIG_BEANCOUNTERS_RSS */
+
+static inline struct page_beancounter * __must_check bc_alloc_rss_counter(void)
+{
+ return NULL;
+}
+
+static inline struct page_beancounter * __must_check bc_alloc_rss_counter_list(
+  long num, struct page_beancounter *list)
+{
+ return NULL;
+}
+
+static inline void bc_free_rss_counter(struct page_beancounter *rc)
+{
+}
+
+#define bc_vmrss_page_add(pg, mm, vma, pb) do { } while (0)
+#define bc_vmrss_page_del(pg, mm, vma)  do { } while (0)
+#define bc_vmrss_page_dup(pg, mm, vma, pb) do { } while (0)
+#define bc_vmrss_page_add_noref(pg, mm, vma) do { } while (0)
+#define mm_rss_pages(mm, start, end) (0)
+
+#define bc_init_rss()   do { } while (0)
+
+#endif /* CONFIG_BEANCOUNTERS_RSS */
+
+#endif
--- ./include/linux/mm.h.bcrsscore 2006-09-05 13:06:37.000000000 +0400
+++ ./include/linux/mm.h 2006-09-05 13:47:12.000000000 +0400
@@ -275,11 +275,15 @@ struct page {
  unsigned long trace[8];
 #endif
 #ifdef CONFIG_BEANCOUNTERS
- struct beancounter *page_bc;
```

```
+ union {
+  struct beancounter *page_bc;
+  struct page_beancounter *page_pb;
+ };
 #endif
 };

 #define page_bc(page)   ((page)->page_bc)
+#define page_pb(page)   ((page)->page_pb)
 #define page_private(page)  ((page)->private)
 #define set_page_private(page, v) ((page)->private = (v))

--- ./include/linux/shmem_fs.h.bcrsscore 2006-09-05 12:59:27.000000000 +0400
+++ ./include/linux/shmem_fs.h 2006-09-05 13:50:19.000000000 +0400
@@ -41,4 +41,6 @@ static inline struct shmem_inode_info *S
  return container_of(inode, struct shmem_inode_info, vfs_inode);
 }

+int is_shmem_mapping(struct address_space *mapping);
+
 #endif
--- ./init/main.c.bcrsscore 2006-09-05 12:54:17.000000000 +0400
+++ ./init/main.c 2006-09-05 13:46:35.000000000 +0400
@@ -51,6 +51,7 @@
 #include <linux/lockdep.h>

 #include <bc/beancounter.h>
+#include <bc/vmrss.h>

 #include <asm/io.h>
 #include <asm/bugs.h>
@@ -608,6 +609,7 @@ asmlinkage void __init start_kernel(void
  check_bugs();

  acpi_early_init(); /* before LAPIC and SMP init */
+ bc_init_rss();

  /* Do the rest non-__init'ed, we're now alive */
  rest_init();
--- ./kernel/bc/Kconfig.bcrsscore 2006-09-05 12:54:14.000000000 +0400
+++ ./kernel/bc/Kconfig 2006-09-05 13:50:35.000000000 +0400
@@ -22,4 +22,13 @@ config BEANCOUNTERS
          per-process basis.  Per-process accounting doesn't prevent malicious
          users from spawning a lot of resource-consuming processes.

+config BEANCOUNTERS_RSS
+ bool "Account physical memory usage"
+ default y
```

```
+ depends on BEANCOUNTERS
+ help
+   This allows to estimate per beancounter physical memory usage.
+   Implemented alghorithm accounts shared pages of memory as well,
+   dividing them by number of beancounter which use the page.
+
 endmenu
--- ./kernel/bc/Makefile.bcrsscore 2006-09-05 12:59:37.000000000 +0400
+++ ./kernel/bc/Makefile 2006-09-05 13:50:48.000000000 +0400
@@ -9,3 +9,4 @@ obj-y += misc.o
 obj-y += sys.o
 obj-y += kmem.o
 obj-y += vmpages.o
+obj-$(CONFIG_BEANCOUNTERS_RSS) += vmrss.o
--- ./kernel/bc/beancounter.c.bcrsscore 2006-09-05 13:44:53.000000000 +0400
+++ ./kernel/bc/beancounter.c 2006-09-05 13:49:38.000000000 +0400
@@ -11,6 +11,7 @@
 #include <linux/hash.h>

 #include <bc/beancounter.h>
+#include <bc/vmrss.h>

 static kmem_cache_t *bc_cachep;
 static struct beancounter default_beancounter;
@@ -112,6 +113,14 @@ void put_beancounter(struct beancounter
    printk("BC: %d has %lu of %s held on put", bc->bc_id,
     bc->bc_parms[i].held, bc_rnames[i]);

+ if (bc->unused_privvmpages != 0)
+  printk("BC: %d has %lu of unused pages held on put", bc->bc_id,
+   bc->unused_privvmpages);
+#ifdef CONFIG_BEANCOUNTERS_RSS
+ if (bc->rss_pages != 0)
+  printk("BC: %d hash %llu of rss pages held on put", bc->bc_id,
+   bc->rss_pages);
+#endif
  hlist_del(&bc->hash);
  nr_beancounters--;
  spin_unlock_irqrestore(&bc_hash_lock, flags);
--- ./kernel/bc/vmpages.c.bcrsscore 2006-09-05 13:45:34.000000000 +0400
+++ ./kernel/bc/vmpages.c 2006-09-05 13:48:50.000000000 +0400
@@ -11,12 +11,17 @@

 #include <bc/beancounter.h>
 #include <bc/vmpages.h>
+#include <bc/vmrss.h>

 #include <asm/page.h>
```

```
 void bc_update_privvmpages(struct beancounter *bc)
 {
- bc->bc_parms[BC_PRIVVMPAGES].held = bc->unused_privvmpages;
+ bc->bc_parms[BC_PRIVVMPAGES].held = bc->unused_privvmpages
+#ifdef CONFIG_BEANCOUNTERS_RSS
+  + (bc->rss_pages >> PB_PAGE_WEIGHT_SHIFT)
+#endif
+  ;
  bc_adjust_minheld(bc, BC_PRIVVMPAGES);
  bc_adjust_maxheld(bc, BC_PRIVVMPAGES);
 }
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/vmrss.c 2006-09-05 13:51:21.000000000 +0400
@@ -0,0 +1,508 @@
+/*
+ *  kernel/bc/vmrss.c
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/mm.h>
+#include <linux/list.h>
+#include <linux/slab.h>
+#include <linux/vmalloc.h>
+#include <linux/shmem_fs.h>
+#include <linux/highmem.h>
+
+#include <bc/beancounter.h>
+#include <bc/vmpages.h>
+#include <bc/vmrss.h>
+
+#include <asm/pgtable.h>
+
+/*
+ * Core object of accounting.
+ * page_beancounter (or rss_counter) ties together page an bc.
+ * Page has associated circular list of such pbs. When page is
+ * shared between bcs then it's size is splitted between all of
+ * them in 2^n-s parts.
+ *
+ * E.g. three bcs will share page like 1/2:1/4:1/4
+ * adding one more reference would produce such a change:
+ * 1/2(bc1) : 1/4(bc2) : 1/4(bc3) ->
+ * (1/4(bc1) + 1/4(bc1)) : 1/4(bc2) : 1/4(bc3) ->
+ * 1/4(bc2) : 1/4(bc3) : 1/4(bc4) : 1/4(bc1)
```

```
+ */
+
+#define PB_MAGIC 0x62700001UL
+
+struct page_beancounter {
+ unsigned long magic;
+ struct page *page;
+ struct beancounter *bc;
+ struct page_beancounter *next_hash;
+ unsigned refcount;
+ struct list_head page_list;
+};
+
+#define PB_REFC_BITS 24
+
+#define pb_shift(p) ((p)->refcount >> PB_REFC_BITS)
+#define pb_shift_inc(p) do { ((p)->refcount += (1 << PB_REFC_BITS)); } while (0)
+#define pb_shift_dec(p) do { ((p)->refcount -= (1 << PB_REFC_BITS)); } while (0)
+
+#define pb_count(p) ((p)->refcount & ((1 << PB_REFC_BITS) - 1))
+#define pb_get(p) do { ((p)->refcount++); } while (0)
+#define pb_put(p) do { ((p)->refcount--); } while (0)
+
+#define pb_refcount_init(p, shift) do {     \
+  (p)->refcount = ((shift) << PB_REFC_BITS) + (1); \
+ } while (0)
+
+static spinlock_t pb_lock = SPIN_LOCK_UNLOCKED;
+static struct page_beancounter **pb_hash_table;
+static unsigned int pb_hash_mask;
+
+static inline int pb_hash(struct beancounter *bc, struct page *page)
+{
+ return (page_to_pfn(page) + (bc->bc_id << 10)) & pb_hash_mask;
+}
+
+static kmem_cache_t *pb_cachep;
+#define alloc_pb() kmem_cache_alloc(pb_cachep, GFP_KERNEL)
+#define free_pb(p) kmem_cache_free(pb_cachep, p)
+
+#define next_page_pb(p) list_entry(p->page_list.next, \
+  struct page_beancounter, page_list);
+#define prev_page_pb(p) list_entry(p->page_list.prev, \
+  struct page_beancounter, page_list);
+
+/*
+ * Allocates a new page_beancounter struct and
+ * initialises requred fields.
```

```
+ * pb->next_hash is set to NULL as this field is used
+ * in two ways:
+ * 1. When pb is in hash - it points to the next one in
+ *    the current hash chain;
+ * 2. When pb is not in hash yet - it points to the next pb
+ *    in list just allocated.
+ */
+struct page_beancounter *bc_alloc_rss_counter(void)
+{
+ struct page_beancounter *pb;
+
+ pb = alloc_pb();
+ if (pb == NULL)
+  return ERR_PTR(-ENOMEM);
+
+ pb->magic = PB_MAGIC;
+ pb->next_hash = NULL;
+ return pb;
+}
+
+/*
+ * This function ensures that @list has at least @num elements.
+ * Otherwise needed elements are allocated and new list is
+ * returned. On error old list is freed.
+ *
+ * num == BC_ALLOC_ALL means that lis must contain as many
+ * elements as there are BCCs in hash now.
+ */
+struct page_beancounter *bc_alloc_rss_counter_list(long num,
+  struct page_beancounter *list)
+{
+ struct page_beancounter *pb;
+
+ for (pb = list; pb != NULL && num != 0; pb = pb->next_hash, num--);
+
+ /* need to allocate num more elements */
+ while (num > 0) {
+  pb = alloc_pb();
+  if (pb == NULL)
+   goto err;
+
+  pb->magic = PB_MAGIC;
+  pb->next_hash = list;
+  list = pb;
+  num--;
+ }
+
+ return list;
```

```
+
+err:
+ bc_free_rss_counter(list);
+ return ERR_PTR(-ENOMEM);
+}
+
+/*
+ * Free the list of page_beancounter-s
+ */
+void bc_free_rss_counter(struct page_beancounter *pb)
+{
+ struct page_beancounter *tmp;
+
+ while (pb) {
+  tmp = pb->next_hash;
+  free_pb(pb);
+  pb = tmp;
+ }
+}
+
+/*
+ * Helpers to update rss_pages and unused_privvmpages on BC
+ */
+static void mod_rss_pages(struct beancounter *bc, int val,
+  struct vm_area_struct *vma, int unused)
+{
+ unsigned long flags;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ if (vma && BC_VM_PRIVATE(vma->vm_flags, vma->vm_file)) {
+  if (unused < 0 && unlikely(bc->unused_privvmpages < -unused)) {
+   printk("BC: overuncharging %d unused pages: "
+     "val %i, held %lu\n",
+     bc->bc_id, unused,
+     bc->unused_privvmpages);
+   unused = -bc->unused_privvmpages;
+  }
+  bc->unused_privvmpages += unused;
+ }
+ bc->rss_pages += val;
+ bc_update_privvmpages(bc);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+}
+
+#define __inc_rss_pages(bc, val) mod_rss_pages(bc, val, NULL, 0)
+#define __dec_rss_pages(bc, val) mod_rss_pages(bc, -(val), NULL, 0)
+#define inc_rss_pages(bc, val, vma) mod_rss_pages(bc, val, vma, -1)
+#define dec_rss_pages(bc, val, vma) mod_rss_pages(bc, -(val), vma, 1)
```

```
+
+/*
+ * Routines to manipulate page-to-bc references (page_beancounter)
+ * Reference may be added, removed or duplicated (see descriptions below)
+ */
+
+static int __pb_dup_ref(struct page *pg, struct beancounter *bc, int hash)
+{
+ struct page_beancounter *p;
+
+ for (p = pb_hash_table[hash];
+    p != NULL && (p->page != pg || p->bc != bc);
+    p = p->next_hash);
+ if (p == NULL)
+  return -1;
+
+ pb_get(p);
+ return 0;
+}
+
+static int __pb_add_ref(struct page *pg, struct beancounter *bc,
+  int hash, struct page_beancounter **ppb)
+{
+ struct page_beancounter *head, *p;
+ int shift, ret;
+
+ p = *ppb;
+ *ppb = p->next_hash;
+
+ p->page = pg;
+ p->bc = get_beancounter(bc);
+ p->next_hash = pb_hash_table[hash];
+ pb_hash_table[hash] = p;
+
+ head = page_pb(pg);
+ if (head != NULL) {
+  BUG_ON(head->magic != PB_MAGIC);
+  /*
+   * Move the first element to the end of the list.
+   * List head (pb_head) is set to the next entry.
+   * Note that this code works even if head is the only element
+   * on the list (because it's cyclic).
+   */
+  page_pb(pg) = next_page_pb(head);
+  pb_shift_inc(head);
+  shift = pb_shift(head);
+  /*
+   * Update user beancounter, the share of head has been changed.
```

```
+   * Note that the shift counter is taken after increment.
+   */
+  __dec_rss_pages(head->bc, PB_PAGE_WEIGHT >> shift);
+  /*
+   * Add the new page beancounter to the end of the list.
+   */
+  list_add_tail(&p->page_list, &page_pb(pg)->page_list);
+ } else {
+  page_pb(pg) = p;
+  shift = 0;
+  INIT_LIST_HEAD(&p->page_list);
+ }
+
+ pb_refcount_init(p, shift);
+ ret = PB_PAGE_WEIGHT >> shift;
+ return ret;
+}
+
+static int __pb_remove_ref(struct page *page, struct beancounter *bc)
+{
+ int hash, ret;
+ struct page_beancounter *p, **q;
+ int shift, shiftt;
+
+ ret = 0;
+
+ hash = pb_hash(bc, page);
+
+ BUG_ON(page_pb(page) != NULL && page_pb(page)->magic != PB_MAGIC);
+ for (q = pb_hash_table + hash, p = *q;
+    p != NULL && (p->page != page || p->bc != bc);
+    q = &p->next_hash, p = *q);
+ if (p == NULL)
+  goto out;
+
+ pb_put(p);
+ if (pb_count(p) > 0)
+  goto out;
+
+ /* remove from the hash list */
+ *q = p->next_hash;
+
+ shift = pb_shift(p);
+ ret = PB_PAGE_WEIGHT >> shift;
+
+ if (page_pb(page) == p) {
+  if (list_empty(&p->page_list)) {
+   page_pb(page) = NULL;
```

```
+   put_beancounter(bc);
+   free_pb(p);
+   goto out;
+  }
+  page_pb(page) = next_page_pb(p);
+ }
+
+ list_del(&p->page_list);
+ put_beancounter(bc);
+ free_pb(p);
+
+ /*
+  * Now balance the list.
+  * Move the tail and adjust its shift counter.
+  */
+ p = prev_page_pb(page_pb(page));
+ shiftt = pb_shift(p);
+ pb_shift_dec(p);
+ page_pb(page) = p;
+ __inc_rss_pages(p->bc, PB_PAGE_WEIGHT >> shiftt);
+
+ /*
+  * If the shift counter of the moved beancounter is different from the
+  * removed one's, repeat the procedure for one more tail beancounter
+  */
+ if (shiftt > shift) {
+  p = prev_page_pb(page_pb(page));
+  pb_shift_dec(p);
+  page_pb(page) = p;
+  __inc_rss_pages(p->bc, PB_PAGE_WEIGHT >> shiftt);
+ }
+out:
+ return ret;
+}
+
+/*
+ * bc_vmrss_page_add: Called when page is added to resident set
+ *   of any mm. In this case page is substracted from unused_privvmpages
+ *   (if it is BC_VM_PRIVATE one) and a reference to BC must be set
+ *   with page_beancounter.
+ *
+ * bc_vmrss_page_del: The reverse operation - page is removed from
+ *   resident set and must become unused.
+ *
+ * bc_vmrss_page_dup: This is called on dup_mmap() when all pages
+ *   become shared between two mm structs. This case has one feature:
+ *   some pages (see below) may lack a reference to BC, so setting
+ *   new reference is not needed, but update of unused_privvmpages
```

```
+ *   is required.
+ *
+ * bc_vmrss_page_add_noref: This is called for (former) reserved pages
+ *   like ZERO_PAGE() or some pages set up with insert_page(). These
+ *   pages must not have reference to any BC, but must be accounted in
+ *   rss.
+ */
+
+void bc_vmrss_page_add(struct page *pg, struct mm_struct *mm,
+  struct vm_area_struct *vma, struct page_beancounter **ppb)
+{
+ struct beancounter *bc;
+ int hash, ret;
+
+ if (!PageAnon(pg) && is_shmem_mapping(pg->mapping))
+  return;
+
+ bc = mm->mm_bc;
+ hash = pb_hash(bc, pg);
+
+ ret = 0;
+ spin_lock(&pb_lock);
+ if (__pb_dup_ref(pg, bc, hash))
+  ret = __pb_add_ref(pg, bc, hash, ppb);
+ spin_unlock(&pb_lock);
+
+ inc_rss_pages(bc, ret, vma);
+}
+
+void bc_vmrss_page_del(struct page *pg, struct mm_struct *mm,
+  struct vm_area_struct *vma)
+{
+ struct beancounter *bc;
+ int ret;
+
+ if (!PageAnon(pg) && is_shmem_mapping(pg->mapping))
+  return;
+
+ bc = mm->mm_bc;
+
+ spin_lock(&pb_lock);
+ ret = __pb_remove_ref(pg, bc);
+ spin_unlock(&pb_lock);
+
+ dec_rss_pages(bc, ret, vma);
+}
+
+void bc_vmrss_page_dup(struct page *pg, struct mm_struct *mm,
```

```
+  struct vm_area_struct *vma, struct page_beancounter **ppb)
+{
+ struct beancounter *bc;
+ int hash, ret;
+
+ if (!PageAnon(pg) && is_shmem_mapping(pg->mapping))
+  return;
+
+ bc = mm->mm_bc;
+ hash = pb_hash(bc, pg);
+
+ ret = 0;
+ spin_lock(&pb_lock);
+ if (page_pb(pg) == NULL)
+  /*
+   * pages like ZERO_PAGE must not be accounted in pbc
+   * so on fork we just skip them
+   */
+  goto out_unlock;
+
+ if (*ppb == PB_COPY_SAME) {
+  if (__pb_dup_ref(pg, bc, hash))
+   WARN_ON(1);
+ } else
+  ret = __pb_add_ref(pg, bc, hash, ppb);
+out_unlock:
+ spin_unlock(&pb_lock);
+
+ inc_rss_pages(bc, ret, vma);
+}
+
+void bc_vmrss_page_add_noref(struct page *pg, struct mm_struct *mm,
+  struct vm_area_struct *vma)
+{
+ inc_rss_pages(mm->mm_bc, 0, vma);
+}
+
+/*
+ * Calculate the number of currently resident pages for
+ * given mm_struct in a given range (addr - end).
+ * This is needed for mprotect_fixup() as by the time
+ * it is called some pages can be resident and thus
+ * not accounted in bc->unused_privvmpages. Such pages
+ * must num be uncharged (as they already are).
+ */
+
+static unsigned long pages_in_pte_range(struct mm_struct *mm, pmd_t *pmd,
+    unsigned long addr, unsigned long end,
```

```
+    unsigned long *pages)
+{
+ pte_t *pte;
+ spinlock_t *ptl;
+
+ pte = pte_offset_map_lock(mm, pmd, addr, &ptl);
+ do {
+  pte_t ptent = *pte;
+  if (!pte_none(ptent) && pte_present(ptent))
+   (*pages)++;
+ } while (pte++, addr += PAGE_SIZE, addr != end);
+ pte_unmap_unlock(pte - 1, ptl);
+ return addr;
+}
+
+static inline unsigned long pages_in_pmd_range(struct mm_struct *mm, pud_t *pud,
+    unsigned long addr, unsigned long end,
+    unsigned long *pages)
+{
+ pmd_t *pmd;
+ unsigned long next;
+
+ pmd = pmd_offset(pud, addr);
+ do {
+  next = pmd_addr_end(addr, end);
+  if (pmd_none_or_clear_bad(pmd))
+   continue;
+
+  next = pages_in_pte_range(mm, pmd, addr, next, pages);
+ } while (pmd++, addr = next, addr != end);
+ return addr;
+}
+
+static inline unsigned long pages_in_pud_range(struct mm_struct *mm, pgd_t *pgd,
+    unsigned long addr, unsigned long end,
+    unsigned long *pages)
+{
+ pud_t *pud;
+ unsigned long next;
+
+ pud = pud_offset(pgd, addr);
+ do {
+  next = pud_addr_end(addr, end);
+  if (pud_none_or_clear_bad(pud))
+   continue;
+
+  next = pages_in_pmd_range(mm, pud, addr, next, pages);
+ } while (pud++, addr = next, addr != end);
```

```
+ return addr;
+}
+
+unsigned long mm_rss_pages(struct mm_struct *mm,
+  unsigned long addr, unsigned long end)
+{
+ pgd_t *pgd;
+ unsigned long next;
+ unsigned long pages;
+
+ BUG_ON(addr >= end);
+
+ pages = 0;
+ pgd = pgd_offset(mm, addr);
+ do {
+  next = pgd_addr_end(addr, end);
+  if (pgd_none_or_clear_bad(pgd))
+   continue;
+
+  next = pages_in_pud_range(mm, pgd, addr, next, &pages);
+ } while (pgd++, addr = next, addr != end);
+ return pages;
+}
+
+void __init bc_init_rss(void)
+{
+ unsigned long hash_size;
+
+ pb_cachep = kmem_cache_create("page_beancounter",
+   sizeof(struct page_beancounter), 0,
+   SLAB_HWCACHE_ALIGN | SLAB_PANIC, NULL, NULL);
+
+ hash_size = num_physpages >> 2;
+ for (pb_hash_mask = 1;
+   (hash_size & pb_hash_mask) != hash_size;
+   pb_hash_mask = (pb_hash_mask << 1) + 1);
+
+ hash_size = pb_hash_mask + 1;
+ printk(KERN_INFO "BC: Page beancounter hash is %lu entries.\n",
+   hash_size);
+ pb_hash_table = vmalloc(hash_size * sizeof(struct page_beancounter *));
+ memset(pb_hash_table, 0, hash_size * sizeof(struct page_beancounter *));
+}
--- ./mm/shmem.c.bcrsscore 2006-09-05 13:39:26.000000000 +0400
+++ ./mm/shmem.c 2006-09-05 13:46:35.000000000 +0400
@@ -2236,6 +2236,12 @@ static struct vm_operations_struct shmem
 #endif
 };
```

```
+#ifdef CONFIG_BEANCOUNTERS_RSS
+int is_shmem_mapping(struct address_space *mapping)
+{
+ return (mapping != NULL && mapping->a_ops == &shmem_aops);
+}
+#endif

 static int shmem_get_sb(struct file_system_type *fs_type,
  int flags, const char *dev_name, void *data, struct vfsmount *mnt)
```

Subject: [PATCH 13/13] BC: vmrss (charges)
Posted by dev on Tue, 05 Sep 2006 15:29:59 GMT

View Forum Message <> Reply to Message

Introduce calls to BC code over the kernel to add
accounting of physical pages/privvmpages.

Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
Signed-Off-By: Kirill Korotaev <dev@sw.ru>

```
---

 fs/exec.c          |   11 ++++
 include/linux/mm.h |    3 -
 kernel/fork.c      |    2
 mm/filemap_xip.c   |    2
 mm/fremap.c        |   11 ++++
 mm/memory.c        |  141 ++++++++++++++++++++++++++++++++++++++++++++++------------
 mm/migrate.c       |    3 +
 mm/mprotect.c      |   12 +++-
 mm/rmap.c          |    4 +
 mm/swapfile.c      |   47 ++++++++++++-----
 10 files changed, 186 insertions(+), 50 deletions(-)

--- ./fs/exec.c.bcrssch 2006-09-05 12:53:55.000000000 +0400
+++ ./fs/exec.c 2006-09-05 13:51:55.000000000 +0400
@@ -50,6 +50,8 @@
 #include <linux/cn_proc.h>
 #include <linux/audit.h>

+#include <bc/vmrss.h>
+
 #include <asm/uaccess.h>
 #include <asm/mmu_context.h>

@@ -308,6 +310,11 @@ void install_arg_page(struct vm_area_str
```

```
  struct mm_struct *mm = vma->vm_mm;
  pte_t * pte;
  spinlock_t *ptl;
+ struct page_beancounter *pb;
+
+ pb = bc_alloc_rss_counter();
+ if (IS_ERR(pb))
+  goto out_nopb;

  if (unlikely(anon_vma_prepare(vma)))
   goto out;
@@ -325,11 +332,15 @@ void install_arg_page(struct vm_area_str
  set_pte_at(mm, address, pte, pte_mkdirty(pte_mkwrite(mk_pte(
      page, vma->vm_page_prot))));
  page_add_new_anon_rmap(page, vma, address);
+ bc_vmrss_page_add(page, mm, vma, &pb);
  pte_unmap_unlock(pte, ptl);

  /* no need for flush_tlb */
+ bc_free_rss_counter(pb);
  return;
 out:
+ bc_free_rss_counter(pb);
+out_nopb:
  __free_page(page);
  force_sig(SIGKILL, current);
 }
--- ./include/linux/mm.h.bcrssch 2006-09-05 13:47:12.000000000 +0400
+++ ./include/linux/mm.h 2006-09-05 13:51:55.000000000 +0400
@@ -753,7 +753,8 @@ void free_pgd_range(struct mmu_gather **
 void free_pgtables(struct mmu_gather **tlb, struct vm_area_struct *start_vma,
   unsigned long floor, unsigned long ceiling);
 int copy_page_range(struct mm_struct *dst, struct mm_struct *src,
-   struct vm_area_struct *vma);
+   struct vm_area_struct *vma,
+   struct vm_area_struct *dst_vma);
 int zeromap_page_range(struct vm_area_struct *vma, unsigned long from,
    unsigned long size, pgprot_t prot);
 void unmap_mapping_range(struct address_space *mapping,
--- ./kernel/fork.c.bcrssch 2006-09-05 13:23:27.000000000 +0400
+++ ./kernel/fork.c 2006-09-05 13:51:55.000000000 +0400
@@ -280,7 +280,7 @@ static inline int dup_mmap(struct mm_str
   rb_parent = &tmp->vm_rb;

   mm->map_count++;
-  retval = copy_page_range(mm, oldmm, mpnt);
+  retval = copy_page_range(mm, oldmm, mpnt, tmp);
```

```
   if (tmp->vm_ops && tmp->vm_ops->open)
    tmp->vm_ops->open(tmp);
--- ./mm/filemap_xip.c.bcrssch 2006-07-10 12:39:20.000000000 +0400
+++ ./mm/filemap_xip.c 2006-09-05 13:51:55.000000000 +0400
@@ -13,6 +13,7 @@
 #include <linux/module.h>
 #include <linux/uio.h>
 #include <linux/rmap.h>
+#include <bc/vmrss.h>
 #include <asm/tlbflush.h>
 #include "filemap.h"

@@ -189,6 +190,7 @@ __xip_unmap (struct address_space * mapp
    /* Nuke the page table entry. */
    flush_cache_page(vma, address, pte_pfn(*pte));
    pteval = ptep_clear_flush(vma, address, pte);
+   bc_vmrss_page_del(page, mm, vma);
    page_remove_rmap(page);
    dec_mm_counter(mm, file_rss);
    BUG_ON(pte_dirty(pteval));
--- ./mm/fremap.c.bcrssch 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/fremap.c 2006-09-05 13:51:55.000000000 +0400
@@ -16,6 +16,8 @@
 #include <linux/module.h>
 #include <linux/syscalls.h>

+#include <bc/vmrss.h>
+
 #include <asm/mmu_context.h>
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>
@@ -33,6 +35,7 @@ static int zap_pte(struct mm_struct *mm,
   if (page) {
    if (pte_dirty(pte))
     set_page_dirty(page);
+   bc_vmrss_page_del(page, mm, vma);
    page_remove_rmap(page);
    page_cache_release(page);
   }
@@ -57,6 +60,11 @@ int install_page(struct mm_struct *mm, s
  pte_t *pte;
  pte_t pte_val;
  spinlock_t *ptl;
+ struct page_beancounter *pb;
+
+ pb = bc_alloc_rss_counter();
+ if (IS_ERR(pb))
+  goto out_nopb;
```

```
  pte = get_locked_pte(mm, addr, &ptl);
  if (!pte)
@@ -82,12 +90,15 @@ int install_page(struct mm_struct *mm, s
  pte_val = mk_pte(page, prot);
  set_pte_at(mm, addr, pte, pte_val);
  page_add_file_rmap(page);
+ bc_vmrss_page_add(page, mm, vma, &pb);
  update_mmu_cache(vma, addr, pte_val);
  lazy_mmu_prot_update(pte_val);
  err = 0;
 unlock:
  pte_unmap_unlock(pte, ptl);
 out:
+ bc_free_rss_counter(pb);
+out_nopb:
  return err;
 }
 EXPORT_SYMBOL(install_page);
--- ./mm/memory.c.bcrssch 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/memory.c 2006-09-05 13:51:55.000000000 +0400
@@ -51,6 +51,9 @@
 #include <linux/init.h>
 #include <linux/writeback.h>

+#include <bc/vmpages.h>
+#include <bc/vmrss.h>
+
 #include <asm/pgalloc.h>
 #include <asm/uaccess.h>
 #include <asm/tlb.h>
@@ -427,7 +430,9 @@ struct page *vm_normal_page(struct vm_ar
 static inline void
 copy_one_pte(struct mm_struct *dst_mm, struct mm_struct *src_mm,
   pte_t *dst_pte, pte_t *src_pte, struct vm_area_struct *vma,
-  unsigned long addr, int *rss)
+  unsigned long addr, int *rss,
+  struct vm_area_struct *dst_vma,
+  struct page_beancounter **ppb)
 {
  unsigned long vm_flags = vma->vm_flags;
  pte_t pte = *src_pte;
@@ -481,6 +486,7 @@ copy_one_pte(struct mm_struct *dst_mm, s
  page = vm_normal_page(vma, addr, pte);
  if (page) {
   get_page(page);
+  bc_vmrss_page_dup(page, dst_mm, dst_vma, ppb);
   page_dup_rmap(page);
```

```
   rss[!!PageAnon(page)]++;
  }
@@ -489,20 +495,32 @@ out_set_pte:
  set_pte_at(dst_mm, addr, dst_pte, pte);
 }

+#define pte_ptrs(a)     (PTRS_PER_PTE - ((a >> PAGE_SHIFT)&(PTRS_PER_PTE - 1)))
+
 static int copy_pte_range(struct mm_struct *dst_mm, struct mm_struct *src_mm,
    pmd_t *dst_pmd, pmd_t *src_pmd, struct vm_area_struct *vma,
-   unsigned long addr, unsigned long end)
+   unsigned long addr, unsigned long end,
+   struct vm_area_struct *dst_vma)
 {
  pte_t *src_pte, *dst_pte;
  spinlock_t *src_ptl, *dst_ptl;
  int progress = 0;
- int rss[2];
+ int rss[2], err;
+ struct page_beancounter *pb;

+ err = -ENOMEM;
+ pb = (mm_same_bc(dst_mm, src_mm) ? PB_COPY_SAME : NULL);
 again:
+ if (pb != PB_COPY_SAME) {
+  pb = bc_alloc_rss_counter_list(pte_ptrs(addr), pb);
+  if (IS_ERR(pb))
+   goto out;
+ }
+
  rss[1] = rss[0] = 0;
  dst_pte = pte_alloc_map_lock(dst_mm, dst_pmd, addr, &dst_ptl);
  if (!dst_pte)
-  return -ENOMEM;
+  goto out;
  src_pte = pte_offset_map_nested(src_pmd, addr);
  src_ptl = pte_lockptr(src_mm, src_pmd);
  spin_lock_nested(src_ptl, SINGLE_DEPTH_NESTING);
@@ -524,7 +542,8 @@ again:
    progress++;
    continue;
   }
-  copy_one_pte(dst_mm, src_mm, dst_pte, src_pte, vma, addr, rss);
+  copy_one_pte(dst_mm, src_mm, dst_pte, src_pte, vma, addr, rss,
+    dst_vma, &pb);
   progress += 8;
  } while (dst_pte++, src_pte++, addr += PAGE_SIZE, addr != end);
```

```
@@ -536,12 +555,18 @@ again:
  cond_resched();
  if (addr != end)
   goto again;
- return 0;
+
+ err = 0;
+out:
+ if (pb != PB_COPY_SAME)
+  bc_free_rss_counter(pb);
+ return err;
 }

 static inline int copy_pmd_range(struct mm_struct *dst_mm, struct mm_struct *src_mm,
   pud_t *dst_pud, pud_t *src_pud, struct vm_area_struct *vma,
-  unsigned long addr, unsigned long end)
+  unsigned long addr, unsigned long end,
+  struct vm_area_struct *dst_vma)
 {
  pmd_t *src_pmd, *dst_pmd;
  unsigned long next;
@@ -555,7 +580,7 @@ static inline int copy_pmd_range(struct
   if (pmd_none_or_clear_bad(src_pmd))
    continue;
   if (copy_pte_range(dst_mm, src_mm, dst_pmd, src_pmd,
-      vma, addr, next))
+      vma, addr, next, dst_vma))
    return -ENOMEM;
  } while (dst_pmd++, src_pmd++, addr = next, addr != end);
  return 0;
@@ -563,7 +588,8 @@ static inline int copy_pmd_range(struct

 static inline int copy_pud_range(struct mm_struct *dst_mm, struct mm_struct *src_mm,
   pgd_t *dst_pgd, pgd_t *src_pgd, struct vm_area_struct *vma,
-  unsigned long addr, unsigned long end)
+  unsigned long addr, unsigned long end,
+  struct vm_area_struct *dst_vma)
 {
  pud_t *src_pud, *dst_pud;
  unsigned long next;
@@ -577,14 +603,14 @@ static inline int copy_pud_range(struct
   if (pud_none_or_clear_bad(src_pud))
    continue;
   if (copy_pmd_range(dst_mm, src_mm, dst_pud, src_pud,
-      vma, addr, next))
+      vma, addr, next, dst_vma))
    return -ENOMEM;
  } while (dst_pud++, src_pud++, addr = next, addr != end);
```

```
  return 0;
 }

 int copy_page_range(struct mm_struct *dst_mm, struct mm_struct *src_mm,
-  struct vm_area_struct *vma)
+  struct vm_area_struct *vma, struct vm_area_struct *dst_vma)
 {
  pgd_t *src_pgd, *dst_pgd;
  unsigned long next;
@@ -612,7 +638,7 @@ int copy_page_range(struct mm_struct *ds
   if (pgd_none_or_clear_bad(src_pgd))
    continue;
   if (copy_pud_range(dst_mm, src_mm, dst_pgd, src_pgd,
-      vma, addr, next))
+      vma, addr, next, dst_vma))
    return -ENOMEM;
  } while (dst_pgd++, src_pgd++, addr = next, addr != end);
  return 0;
@@ -681,6 +707,7 @@ static unsigned long zap_pte_range(struc
     mark_page_accessed(page);
    file_rss--;
   }
+   bc_vmrss_page_del(page, mm, vma);
   page_remove_rmap(page);
   tlb_remove_page(tlb, page);
   continue;
@@ -1104,8 +1131,9 @@ int get_user_pages(struct task_struct *t
 }
 EXPORT_SYMBOL(get_user_pages);

-static int zeromap_pte_range(struct mm_struct *mm, pmd_t *pmd,
-   unsigned long addr, unsigned long end, pgprot_t prot)
+static int zeromap_pte_range(struct mm_struct *mm,
+  struct vm_area_struct *vma, pmd_t *pmd,
+  unsigned long addr, unsigned long end, pgprot_t prot)
 {
  pte_t *pte;
  spinlock_t *ptl;
@@ -1118,6 +1146,7 @@ static int zeromap_pte_range(struct mm_s
   struct page *page = ZERO_PAGE(addr);
   pte_t zero_pte = pte_wrprotect(mk_pte(page, prot));
   page_cache_get(page);
+  bc_vmrss_page_add_noref(page, mm, vma);
   page_add_file_rmap(page);
   inc_mm_counter(mm, file_rss);
   BUG_ON(!pte_none(*pte));
@@ -1128,8 +1157,9 @@ static int zeromap_pte_range(struct mm_s
  return 0;
```

```
 }

-static inline int zeromap_pmd_range(struct mm_struct *mm, pud_t *pud,
-   unsigned long addr, unsigned long end, pgprot_t prot)
+static inline int zeromap_pmd_range(struct mm_struct *mm,
+  struct vm_area_struct *vma, pud_t *pud,
+  unsigned long addr, unsigned long end, pgprot_t prot)
 {
  pmd_t *pmd;
  unsigned long next;
@@ -1139,14 +1169,15 @@ static inline int zeromap_pmd_range(stru
   return -ENOMEM;
  do {
   next = pmd_addr_end(addr, end);
-  if (zeromap_pte_range(mm, pmd, addr, next, prot))
+  if (zeromap_pte_range(mm, vma, pmd, addr, next, prot))
    return -ENOMEM;
  } while (pmd++, addr = next, addr != end);
  return 0;
 }

-static inline int zeromap_pud_range(struct mm_struct *mm, pgd_t *pgd,
-   unsigned long addr, unsigned long end, pgprot_t prot)
+static inline int zeromap_pud_range(struct mm_struct *mm,
+  struct vm_area_struct *vma, pgd_t *pgd,
+  unsigned long addr, unsigned long end, pgprot_t prot)
 {
  pud_t *pud;
  unsigned long next;
@@ -1156,7 +1187,7 @@ static inline int zeromap_pud_range(stru
   return -ENOMEM;
  do {
   next = pud_addr_end(addr, end);
-  if (zeromap_pmd_range(mm, pud, addr, next, prot))
+  if (zeromap_pmd_range(mm, vma, pud, addr, next, prot))
    return -ENOMEM;
  } while (pud++, addr = next, addr != end);
  return 0;
@@ -1176,7 +1207,7 @@ int zeromap_page_range(struct vm_area_st
  flush_cache_range(vma, addr, end);
  do {
   next = pgd_addr_end(addr, end);
-  err = zeromap_pud_range(mm, pgd, addr, next, prot);
+  err = zeromap_pud_range(mm, vma, pgd, addr, next, prot);
   if (err)
    break;
  } while (pgd++, addr = next, addr != end);
@@ -1202,12 +1233,15 @@ pte_t * fastcall get_locked_pte(struct m
```

```
  * old drivers should use this, and they needed to mark their
  * pages reserved for the old functions anyway.
  */
-static int insert_page(struct mm_struct *mm, unsigned long addr, struct page *page, pgprot_t
prot)
+static int insert_page(struct vm_area_struct *vma, unsigned long addr, struct page *page,
pgprot_t prot)
 {
+ struct mm_struct *mm;
  int retval;
  pte_t *pte;
  spinlock_t *ptl;

+ mm = vma->vm_mm;
+
  retval = -EINVAL;
  if (PageAnon(page))
   goto out;
@@ -1223,6 +1257,7 @@ static int insert_page(struct mm_struct
  /* Ok, finally just insert the thing.. */
  get_page(page);
  inc_mm_counter(mm, file_rss);
+ bc_vmrss_page_add_noref(page, mm, vma);
  page_add_file_rmap(page);
  set_pte_at(mm, addr, pte, mk_pte(page, prot));

@@ -1262,7 +1297,7 @@ int vm_insert_page(struct vm_area_struct
  if (!page_count(page))
   return -EINVAL;
  vma->vm_flags |= VM_INSERTPAGE;
- return insert_page(vma->vm_mm, addr, page, vma->vm_page_prot);
+ return insert_page(vma, addr, page, vma->vm_page_prot);
 }
 EXPORT_SYMBOL(vm_insert_page);

@@ -1483,6 +1518,7 @@ static int do_wp_page(struct mm_struct *
  pte_t entry;
  int reuse = 0, ret = VM_FAULT_MINOR;
  struct page *dirty_page = NULL;
+ struct page_beancounter *pb;

  old_page = vm_normal_page(vma, address, orig_pte);
  if (!old_page)
@@ -1555,6 +1591,10 @@ static int do_wp_page(struct mm_struct *
 gotten:
  pte_unmap_unlock(page_table, ptl);

+ pb = bc_alloc_rss_counter();
```

```
+ if (IS_ERR(pb))
+  goto oom_nopb;
+
  if (unlikely(anon_vma_prepare(vma)))
   goto oom;
  if (old_page == ZERO_PAGE(address)) {
@@ -1574,6 +1614,7 @@ gotten:
  page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
  if (likely(pte_same(*page_table, orig_pte))) {
   if (old_page) {
+   bc_vmrss_page_del(old_page, mm, vma);
    page_remove_rmap(old_page);
    if (!PageAnon(old_page)) {
     dec_mm_counter(mm, file_rss);
@@ -1589,6 +1630,7 @@ gotten:
   update_mmu_cache(vma, address, entry);
   lru_cache_add_active(new_page);
   page_add_new_anon_rmap(new_page, vma, address);
+  bc_vmrss_page_add(new_page, mm, vma, &pb);

   /* Free the old page.. */
   new_page = old_page;
@@ -1598,6 +1640,7 @@ gotten:
   page_cache_release(new_page);
  if (old_page)
   page_cache_release(old_page);
+ bc_free_rss_counter(pb);
 unlock:
  pte_unmap_unlock(page_table, ptl);
  if (dirty_page) {
@@ -1606,6 +1649,8 @@ unlock:
  }
  return ret;
 oom:
+ bc_free_rss_counter(pb);
+oom_nopb:
  if (old_page)
   page_cache_release(old_page);
  return VM_FAULT_OOM;
@@ -1970,9 +2015,14 @@ static int do_swap_page(struct mm_struct
  swp_entry_t entry;
  pte_t pte;
  int ret = VM_FAULT_MINOR;
+ struct page_beancounter *pb;

  if (!pte_unmap_same(mm, pmd, page_table, orig_pte))
-  goto out;
+  goto out_nopb;
```

```
+
+ pb = bc_alloc_rss_counter();
+ if (IS_ERR(pb))
+  goto out_nopb;

  entry = pte_to_swp_entry(orig_pte);
  if (is_migration_entry(entry)) {
@@ -2030,6 +2080,7 @@ static int do_swap_page(struct mm_struct
  flush_icache_page(vma, page);
  set_pte_at(mm, address, page_table, pte);
  page_add_anon_rmap(page, vma, address);
+ bc_vmrss_page_add(page, mm, vma, &pb);

  swap_free(entry);
  if (vm_swap_full())
@@ -2049,11 +2100,14 @@ static int do_swap_page(struct mm_struct
 unlock:
  pte_unmap_unlock(page_table, ptl);
 out:
+ bc_free_rss_counter(pb);
+out_nopb:
  return ret;
 out_nomap:
  pte_unmap_unlock(page_table, ptl);
  unlock_page(page);
  page_cache_release(page);
+ bc_free_rss_counter(pb);
  return ret;
 }

@@ -2069,11 +2123,16 @@ static int do_anonymous_page(struct mm_s
  struct page *page;
  spinlock_t *ptl;
  pte_t entry;
+ struct page_beancounter *pb;

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

+  pb = bc_alloc_rss_counter();
+  if (IS_ERR(pb))
+   goto oom_nopb;
+
   if (unlikely(anon_vma_prepare(vma)))
    goto oom;
   page = alloc_zeroed_user_highpage(vma, address);
@@ -2089,7 +2148,9 @@ static int do_anonymous_page(struct mm_s
```

```
   inc_mm_counter(mm, anon_rss);
   lru_cache_add_active(page);
   page_add_new_anon_rmap(page, vma, address);
+  bc_vmrss_page_add(page, mm, vma, &pb);
  } else {
+  pb = NULL;
   /* Map the ZERO_PAGE - vm_page_prot is readonly */
   page = ZERO_PAGE(address);
   page_cache_get(page);
@@ -2101,6 +2162,7 @@ static int do_anonymous_page(struct mm_s
    goto release;
   inc_mm_counter(mm, file_rss);
   page_add_file_rmap(page);
+  bc_vmrss_page_add_noref(page, mm, vma);
  }

  set_pte_at(mm, address, page_table, entry);
@@ -2110,11 +2172,14 @@ static int do_anonymous_page(struct mm_s
  lazy_mmu_prot_update(entry);
 unlock:
  pte_unmap_unlock(page_table, ptl);
+ bc_free_rss_counter(pb);
  return VM_FAULT_MINOR;
 release:
  page_cache_release(page);
  goto unlock;
 oom:
+ bc_free_rss_counter(pb);
+oom_nopb:
  return VM_FAULT_OOM;
 }

@@ -2143,6 +2208,7 @@ static int do_no_page(struct mm_struct *
  int ret = VM_FAULT_MINOR;
  int anon = 0;
  struct page *dirty_page = NULL;
+ struct page_beancounter *pb;

  pte_unmap(page_table);
  BUG_ON(vma->vm_flags & VM_PFNMAP);
@@ -2152,6 +2218,10 @@ static int do_no_page(struct mm_struct *
   sequence = mapping->truncate_count;
   smp_rmb(); /* serializes i_size against truncate_count */
  }
+
+ pb = bc_alloc_rss_counter();
+ if (IS_ERR(pb))
+  goto oom_nopb;
```

```
 retry:
  new_page = vma->vm_ops->nopage(vma, address & PAGE_MASK, &ret);
  /*
@@ -2164,9 +2234,9 @@ retry:

  /* no page was available -- either SIGBUS or OOM */
  if (new_page == NOPAGE_SIGBUS)
-  return VM_FAULT_SIGBUS;
+  goto bus_nopg;
  if (new_page == NOPAGE_OOM)
-  return VM_FAULT_OOM;
+  goto oom_nopg;

  /*
   * Should we do an early C-O-W break?
@@ -2190,11 +2260,8 @@ retry:
     * address space wants to know that the page is about
     * to become writable */
    if (vma->vm_ops->page_mkwrite &&
-       vma->vm_ops->page_mkwrite(vma, new_page) < 0
-       ) {
-    page_cache_release(new_page);
-    return VM_FAULT_SIGBUS;
-   }
+       vma->vm_ops->page_mkwrite(vma, new_page) < 0)
+    goto bus;
   }
  }

@@ -2242,6 +2309,8 @@ retry:
     get_page(dirty_page);
    }
   }
+
+  bc_vmrss_page_add(new_page, mm, vma, &pb);
  } else {
   /* One of our sibling threads was faster, back out. */
   page_cache_release(new_page);
@@ -2257,10 +2326,20 @@ unlock:
   set_page_dirty_balance(dirty_page);
   put_page(dirty_page);
  }
+ bc_free_rss_counter(pb);
  return ret;
 oom:
  page_cache_release(new_page);
+oom_nopg:
+ bc_free_rss_counter(pb);
```

```
+oom_nopb:
  return VM_FAULT_OOM;
+
+bus:
+ page_cache_release(new_page);
+bus_nopg:
+ bc_free_rss_counter(pb);
+ return VM_FAULT_SIGBUS;
 }

 /*
--- ./mm/migrate.c.bcrssch 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/migrate.c 2006-09-05 13:51:55.000000000 +0400
@@ -29,6 +29,8 @@
 #include <linux/vmalloc.h>
 #include <linux/security.h>

+#include <bc/vmrss.h>
+
 #include "internal.h"

 #define lru_to_page(_head) (list_entry((_head)->prev, struct page, lru))
@@ -179,6 +181,7 @@ static void remove_migration_pte(struct
  else
   page_add_file_rmap(new);

+ bc_vmrss_page_del(new, mm, vma);
  /* No need to invalidate - it was non-present before */
  update_mmu_cache(vma, addr, pte);
  lazy_mmu_prot_update(pte);
--- ./mm/mprotect.c.bcrssch 2006-09-05 13:27:40.000000000 +0400
+++ ./mm/mprotect.c 2006-09-05 13:54:20.000000000 +0400
@@ -22,6 +22,7 @@
 #include <linux/swap.h>
 #include <linux/swapops.h>
 #include <bc/vmpages.h>
+#include <bc/vmrss.h>
 #include <asm/uaccess.h>
 #include <asm/pgtable.h>
 #include <asm/cacheflush.h>
@@ -141,6 +142,7 @@ mprotect_fixup(struct vm_area_struct *vm
  int error;
  int dirty_accountable = 0;
  int recharge;
+ unsigned long rss;

  if (newflags == oldflags) {
   *pprev = vma;
```

```
@@ -148,8 +150,10 @@ mprotect_fixup(struct vm_area_struct *vm
  }

  recharge = bc_privvm_recharge(oldflags, newflags, vma->vm_file);
- if (recharge == BC_CHARGE) {
-  if (bc_privvm_charge(mm, end - start))
+ if (recharge != BC_NOCHARGE) {
+  rss = mm_rss_pages(mm, start, end) << PAGE_SHIFT;
+  if (recharge == BC_CHARGE && bc_privvm_charge(mm,
+      end - start - rss) < 0)
    return -ENOMEM;
  }

@@ -215,7 +219,7 @@ success:
   change_protection(vma, start, end, vma->vm_page_prot, dirty_accountable);

  if (recharge == BC_UNCHARGE)
-  bc_privvm_uncharge(mm, end - start);
+  bc_privvm_uncharge(mm, end - start - rss);
  vm_stat_account(mm, oldflags, vma->vm_file, -nrpages);
  vm_stat_account(mm, newflags, vma->vm_file, nrpages);
  return 0;
@@ -224,7 +228,7 @@ fail:
  vm_unacct_memory(charged);
 fail_acct:
  if (recharge == BC_CHARGE)
-  bc_privvm_uncharge(mm, end - start);
+  bc_privvm_uncharge(mm, end - start - rss);
  return error;
 }

--- ./mm/rmap.c.bcrssch 2006-09-05 12:58:17.000000000 +0400
+++ ./mm/rmap.c 2006-09-05 13:51:55.000000000 +0400
@@ -54,6 +54,8 @@
 #include <linux/rcupdate.h>
 #include <linux/module.h>

+#include <bc/vmrss.h>
+
 #include <asm/tlbflush.h>

 struct kmem_cache *anon_vma_cachep;
@@ -687,6 +689,7 @@ static int try_to_unmap_one(struct page
   dec_mm_counter(mm, file_rss);


+ bc_vmrss_page_del(page, mm, vma);
  page_remove_rmap(page);
```

```
  page_cache_release(page);

@@ -777,6 +780,7 @@ static void try_to_unmap_cluster(unsigne
    if (pte_dirty(pteval))
     set_page_dirty(page);

+  bc_vmrss_page_del(page, mm, vma);
    page_remove_rmap(page);
    page_cache_release(page);
    dec_mm_counter(mm, file_rss);
--- ./mm/swapfile.c.bcrssch 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/swapfile.c 2006-09-05 13:54:59.000000000 +0400
@@ -28,6 +28,9 @@
 #include <linux/capability.h>
 #include <linux/syscalls.h>

+#include <bc/beancounter.h>
+#include <bc/vmrss.h>
+
 #include <asm/pgtable.h>
 #include <asm/tlbflush.h>
 #include <linux/swapops.h>
@@ -487,13 +490,15 @@ unsigned int count_swap_pages(int type,
  * force COW, vm_page_prot omits write permission from any private vma.
  */
 static void unuse_pte(struct vm_area_struct *vma, pte_t *pte,
-  unsigned long addr, swp_entry_t entry, struct page *page)
+  unsigned long addr, swp_entry_t entry, struct page *page,
+  struct page_beancounter **ppb)
 {
  inc_mm_counter(vma->vm_mm, anon_rss);
  get_page(page);
  set_pte_at(vma->vm_mm, addr, pte,
      pte_mkold(mk_pte(page, vma->vm_page_prot)));
  page_add_anon_rmap(page, vma, addr);
+ bc_vmrss_page_add(page, vma->vm_mm, vma, ppb);
  swap_free(entry);
  /*
   * Move the page to the active list so it is not
@@ -504,7 +509,8 @@ static void unuse_pte(struct vm_area_str

 static int unuse_pte_range(struct vm_area_struct *vma, pmd_t *pmd,
     unsigned long addr, unsigned long end,
-    swp_entry_t entry, struct page *page)
+    swp_entry_t entry, struct page *page,
+    struct page_beancounter **ppb)
 {
  pte_t swp_pte = swp_entry_to_pte(entry);
```

```
  pte_t *pte;
@@ -518,7 +524,7 @@ 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);
+    unuse_pte(vma, pte++, addr, entry, page, ppb);
     found = 1;
     break;
    }
@@ -529,7 +535,8 @@ static int unuse_pte_range(struct vm_are

 static inline int unuse_pmd_range(struct vm_area_struct *vma, pud_t *pud,
      unsigned long addr, unsigned long end,
-     swp_entry_t entry, struct page *page)
+     swp_entry_t entry, struct page *page,
+     struct page_beancounter **ppb)
 {
  pmd_t *pmd;
  unsigned long next;
@@ -539,7 +546,7 @@ static inline int unuse_pmd_range(struct
    next = pmd_addr_end(addr, end);
    if (pmd_none_or_clear_bad(pmd))
     continue;
-   if (unuse_pte_range(vma, pmd, addr, next, entry, page))
+   if (unuse_pte_range(vma, pmd, addr, next, entry, page, ppb))
     return 1;
  } while (pmd++, addr = next, addr != end);
  return 0;
@@ -547,7 +554,8 @@ static inline int unuse_pmd_range(struct

 static inline int unuse_pud_range(struct vm_area_struct *vma, pgd_t *pgd,
      unsigned long addr, unsigned long end,
-     swp_entry_t entry, struct page *page)
+     swp_entry_t entry, struct page *page,
+     struct page_beancounter **ppb)
 {
  pud_t *pud;
  unsigned long next;
@@ -557,14 +565,15 @@ static inline int unuse_pud_range(struct
    next = pud_addr_end(addr, end);
    if (pud_none_or_clear_bad(pud))
     continue;
-   if (unuse_pmd_range(vma, pud, addr, next, entry, page))
+   if (unuse_pmd_range(vma, pud, addr, next, entry, page, ppb))
     return 1;
  } while (pud++, addr = next, addr != end);
  return 0;
```

```
 }

 static int unuse_vma(struct vm_area_struct *vma,
-    swp_entry_t entry, struct page *page)
+    swp_entry_t entry, struct page *page,
+    struct page_beancounter **ppb)
 {
  pgd_t *pgd;
  unsigned long addr, end, next;
@@ -585,14 +594,15 @@ 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))
+  if (unuse_pud_range(vma, pgd, addr, next, entry, page, ppb))
    return 1;
  } while (pgd++, addr = next, addr != end);
  return 0;
 }

 static int unuse_mm(struct mm_struct *mm,
-    swp_entry_t entry, struct page *page)
+    swp_entry_t entry, struct page *page,
+    struct page_beancounter **ppb)
 {
  struct vm_area_struct *vma;

@@ -607,7 +617,7 @@ 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 && unuse_vma(vma, entry, page, ppb))
    break;
  }
  up_read(&mm->mmap_sem);
@@ -673,6 +683,7 @@ static int try_to_unuse(unsigned int typ
  int retval = 0;
  int reset_overflow = 0;
  int shmem;
+ struct page_beancounter *pb;

  /*
   * When searching mms for an entry, a good strategy is to
@@ -692,6 +703,7 @@ static int try_to_unuse(unsigned int typ
  start_mm = &init_mm;
  atomic_inc(&init_mm.mm_users);
```

```
+ pb = NULL;
  /*
   * Keep on scanning until all entries have gone.  Usually,
   * one pass through swap_map is enough, but not necessarily:
@@ -703,6 +715,12 @@ static int try_to_unuse(unsigned int typ
    break;
   }

+  pb = bc_alloc_rss_counter_list(nr_beancounters, pb);
+  if (IS_ERR(pb)) {
+   retval = PTR_ERR(pb);
+   break;
+  }
+
   /*
    * Get a page for the entry, using the existing swap
    * cache page if there is one.  Otherwise, get a clean
@@ -757,7 +775,7 @@ static int try_to_unuse(unsigned int typ
    if (start_mm == &init_mm)
     shmem = shmem_unuse(entry, page);
    else
-    retval = unuse_mm(start_mm, entry, page);
+    retval = unuse_mm(start_mm, entry, page, &pb);
   }
   if (*swap_map > 1) {
    int set_start_mm = (*swap_map >= swcount);
@@ -787,7 +805,7 @@ static int try_to_unuse(unsigned int typ
      set_start_mm = 1;
      shmem = shmem_unuse(entry, page);
     } else
-     retval = unuse_mm(mm, entry, page);
+     retval = unuse_mm(mm, entry, page, &pb);
     if (set_start_mm && *swap_map < swcount) {
      mmput(new_start_mm);
      atomic_inc(&mm->mm_users);
@@ -878,6 +896,9 @@ static int try_to_unuse(unsigned int typ
   cond_resched();
  }

+ if (!IS_ERR(pb))
+  bc_free_rss_counter(pb);
+
  mmput(start_mm);
  if (reset_overflow) {
   printk(KERN_WARNING "swapoff: cleared swap entry overflow\n");
```

Subject: Re: [ckrm-tech] [PATCH 5/13] BC: user interface (syscalls)
Posted by Balbir Singh on Tue, 05 Sep 2006 16:04:52 GMT
View Forum Message <> Reply to Message

> +
> +asmlinkage long sys_set_bcid(bcid_t id)
> +{
> + int error;
> + struct beancounter *bc;
> + struct task_beancounter *task_bc;
> +
> + task_bc = &current->task_bc;

I was playing around with the bc patches and found that to make use of bc's, I had to actually call set_bcid() and then exec() a task/shell so that the id would stick around. Would you consider changing sys_set_bcid to sys_set_task_bcid() or adding a new system call sys_set_task_bcid()? We could pass the pid that we intend to associate with the new id. This also means we'll need locking around to protect task->task_bc.

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Balbir Singh on Tue, 05 Sep 2006 16:53:21 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
> Core Resource Beancounters (BC) + kernel/user memory control.
>
> BC allows to account and control consumption
> of kernel resources used by group of processes.
>
> Draft UBC description on OpenVZ wiki can be found at
> http://wiki.openvz.org/UBC_parameters
>
> The full BC patch set allows to control:
> - kernel memory. All the kernel objects allocatable
> on user demand should be accounted and limited
> for DoS protection.
> E.g. page tables, task structs, vmas etc.

One of the key requirements of resource management for us is to be able to
migrate tasks across resource groups. Since bean counters do not associate
a list of tasks associated with them, I do not see how this can be done
with the existing bean counters.

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Dave Hansen on Tue, 05 Sep 2006 17:46:32 GMT
View Forum Message <> Reply to Message

On Tue, 2006-09-05 at 19:02 +0400, Kirill Korotaev wrote:
> Core Resource Beancounters (BC) + kernel/user memory control.
>
> BC allows to account and control consumption
> of kernel resources used by group of processes.

Hi Kirill,

I've honestly lost track of these discussions along the way, so I hope
you don't mind summarizing a bit.

Do these patches help with accounting for anything other than memory?
Will we need new user/kernel interfaces for cpu, i/o bandwidth, etc...?

Have you given any thought to the possibility that a task might need to
move between accounting contexts? That has certainly been a
"requirement" pushed on to CKRM for a long time, and the need goes
something like this:

1. A system runs a web server, which services several virtual domains
2. that web server receives a request for foo.com
3. the web server switches into foo.com's accounting context
4. the web server reads things from disk, allocates some memory, and
   makes a database request.
5. the database receives the request, and switches into foo.com's
   accounting context, and charges foo.com for its resource use
etc...

So, the goal is to run _one_ copy of an application on a system, but
account for its resources in a much more fine-grained way than at the

application level.

I think we can probably use beancounters for this, if we do not worry
about migrating _existing_ charges when we change accounting context.
Does that make sense?

-- Dave

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Balbir Singh on Tue, 05 Sep 2006 18:28:49 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Tue, 2006-09-05 at 19:02 +0400, Kirill Korotaev wrote:
>> Core Resource Beancounters (BC) + kernel/user memory control.
>>
>> BC allows to account and control consumption
>> of kernel resources used by group of processes.
>
> Hi Kirill,
>
> I've honestly lost track of these discussions along the way, so I hope
> you don't mind summarizing a bit.
>
> Do these patches help with accounting for anything other than memory?
> Will we need new user/kernel interfaces for cpu, i/o bandwidth, etc...?
>
> Have you given any thought to the possibility that a task might need to
> move between accounting contexts? That has certainly been a
> "requirement" pushed on to CKRM for a long time, and the need goes
> something like this:
>
> 1. A system runs a web server, which services several virtual domains
> 2. that web server receives a request for foo.com
> 3. the web server switches into foo.com's accounting context
> 4. the web server reads things from disk, allocates some memory, and
>   makes a database request.
> 5. the database receives the request, and switches into foo.com's
>   accounting context, and charges foo.com for its resource use
> etc...
>
> So, the goal is to run _one_ copy of an application on a system, but
> account for its resources in a much more fine-grained way than at the
> application level.
>
> I think we can probably use beancounters for this, if we do not worry

> about migrating _existing_ charges when we change accounting context.
> Does that make sense?
>
> -- Dave

This is much better stated than I did. Thanks!

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [PATCH 11/13] BC: vmrss (preparations)
Posted by Cedric Le Goater on Tue, 05 Sep 2006 22:09:41 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:

<snip>

> --- ./include/bc/beancounter.h.bcvmrssprep 2006-09-05
> 13:17:50.000000000 +0400
> +++ ./include/bc/beancounter.h 2006-09-05 13:44:33.000000000 +0400
> @@ -45,6 +45,13 @@ struct bc_resource_parm {
> #define BC_MAXVALUE LONG_MAX
>
> /*
> + * This magic is used to distinuish user beancounter and pages beancounter
> + * in struct page. page_ub and page_bc are placed in union and MAGIC
> + * ensures us that we don't use pbc as ubc in bc_page_uncharge().
> + */
> +#define BC_MAGIC 0x62756275UL
> +
> +/*
> * Resource management structures
> * Serialization issues:
> * beancounter list management is protected via bc_hash_lock
> @@ -54,11 +61,13 @@ struct bc_resource_parm {
> */
>
> struct beancounter {
> + unsigned long bc_magic;
> atomic_t bc_refcount;
> spinlock_t bc_lock;
> bcid_t bc_id;
> struct hlist_node hash;

>
> +   unsigned long   unused_privvmpages;
>   /* resources statistics and settings */
>   struct bc_resource_parm   bc_parms[BC_RESOURCES];
> };
> @@ -74,6 +83,8 @@ enum bc_severity { BC_BARRIER, BC_LIMIT,
>
> #ifdef CONFIG_BEANCOUNTERS
>
> +extern unsigned int nr_beancounters = 1;
> +

my gcc doesn't like this one ...

regards,

C.

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>

---
 include/bc/beancounter.h | 2 +-
 kernel/bc/beancounter.c | 2 +-
 2 files changed, 2 insertions(+), 2 deletions(-)

Index: 2.6.18-rc5-mm1/include/bc/beancounter.h
 === =======
--- 2.6.18-rc5-mm1.orig/include/bc/beancounter.h
+++ 2.6.18-rc5-mm1/include/bc/beancounter.h
@@ -86,7 +86,7 @@ enum bc_severity { BC_BARRIER, BC_LIMIT,

 #ifdef CONFIG_BEANCOUNTERS

-extern unsigned int nr_beancounters = 1;
+extern unsigned int nr_beancounters;

 /*
  * These functions tune minheld and maxheld values for a given
Index: 2.6.18-rc5-mm1/kernel/bc/beancounter.c
 === =======
--- 2.6.18-rc5-mm1.orig/kernel/bc/beancounter.c
+++ 2.6.18-rc5-mm1/kernel/bc/beancounter.c
@@ -20,7 +20,7 @@ static void init_beancounter_struct(stru

 struct beancounter init_bc;

-unsigned int nr_beancounters;
+unsigned int nr_beancounters = 1;

```
const char *bc_rnames[] = {
 "kmemsize", /* 0 */
```

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Rohit Seth on Wed, 06 Sep 2006 00:17:58 GMT
View Forum Message <> Reply to Message

On Tue, 2006-09-05 at 10:46 -0700, Dave Hansen wrote:
> On Tue, 2006-09-05 at 19:02 +0400, Kirill Korotaev wrote:
> > Core Resource Beancounters (BC) + kernel/user memory control.
> >
> > BC allows to account and control consumption
> > of kernel resources used by group of processes.
>
> Hi Kirill,
>
> I've honestly lost track of these discussions along the way, so I hope
> you don't mind summarizing a bit.
>
> Do these patches help with accounting for anything other than memory?
> Will we need new user/kernel interfaces for cpu, i/o bandwidth, etc...?
>
> Have you given any thought to the possibility that a task might need to
> move between accounting contexts? That has certainly been a
> "requirement" pushed on to CKRM for a long time, and the need goes
> something like this:
>
> 1. A system runs a web server, which services several virtual domains
> 2. that web server receives a request for foo.com
> 3. the web server switches into foo.com's accounting context
> 4. the web server reads things from disk, allocates some memory, and
> makes a database request.
> 5. the database receives the request, and switches into foo.com's
> accounting context, and charges foo.com for its resource use
> etc...
>

I'm wondering why not have different processes to serve different domains on the same physical server...particularly when they have different database to work on. Is the amount of memory that you save by having a single copy that much useful that you are even okay to serialize the whole operation (What would happen, while the request for foo.com is getting worked on, there is another request for foo_bar.com...does it need to wait for foo.com request to get done before it can be served).

> So, the goal is to run _one_ copy of an application on a system, but
> account for its resources in a much more fine-grained way than at the
> application level.
>

What is that fine grained way. If not process based then can it be
associated with file system location?

-rohit

Subject: Re: [PATCH 9/13] BC: locked pages (charge hooks)
Posted by Nick Piggin on Wed, 06 Sep 2006 03:43:35 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:

> Introduce calls to BC core over the kernel to charge locked memory.
>
> Normaly new locked piece of memory may appear in insert_vm_struct,
> but there are places (do_mmap_pgoff, dup_mmap etc) when new vma
> is not inserted by insert_vm_struct(), but either link_vma-ed or
> merged with some other - these places call BC code explicitly.
>
> Plus sys_mlock[all] itself has to be patched to charge/uncharge
> needed amount of pages.

I still haven't heard your good reasons why such a complex scheme is
required when my really simple proposal of unconditionally charging
the page to the container it was allocated by.

That has the benefit of not being full of user explotable holes and
also not putting such a huge burden on mm/ and the wider kernel in
general.

--

Subject: Re: [ckrm-tech] [PATCH 5/13] BC: user interface (syscalls)
Posted by Pavel Emelianov on Wed, 06 Sep 2006 08:29:35 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:

>> +
>> +asmlinkage long sys_set_bcid(bcid_t id)
>> +{
>> +   int error;
>> +   struct beancounter *bc;
>> +   struct task_beancounter *task_bc;
>> +
>> +   task_bc = &current->task_bc;
>
> I was playing around with the bc patches and found that to make
> use of bc's, I had to actually call set_bcid() and then exec() a
> task/shell so that the id would stick around. Would you consider
That sounds very strange as sys_set_bcid() actually changes current's
exec_bc.
One note is about mm's bc - mm obtains new bc only after fork or exec -
that's
true. But kmemsize starts charging right after the sys_set_bcid.
> changing sys_set_bcid to sys_set_task_bcid() or adding a new
> system call sys_set_task_bcid()? We could pass the pid that we
> intend to associate with the new id. This also means we'll need
> locking around to protect task->task_bc.

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Wed, 06 Sep 2006 08:34:33 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Kirill Korotaev wrote:
>> Core Resource Beancounters (BC) + kernel/user memory control.
>>
>> BC allows to account and control consumption
>> of kernel resources used by group of processes.
>>
>> Draft UBC description on OpenVZ wiki can be found at
>> http://wiki.openvz.org/UBC_parameters
>>
>> The full BC patch set allows to control:
>> - kernel memory. All the kernel objects allocatable
>> on user demand should be accounted and limited
>> for DoS protection.
>> E.g. page tables, task structs, vmas etc.
>
> One of the key requirements of resource management for us is to be
> able to
> migrate tasks across resource groups. Since bean counters do not
> associate

Then could you tell me please what to do with all the resources allocated
by the task you are moving to another group?
> a list of tasks associated with them, I do not see how this can be done
> with the existing bean counters.
>
Associating a list of tasks with beancounter is not so hard actually.
The question is wether this is usefull (regarding my previous comment).

Subject: Re: [PATCH 9/13] BC: locked pages (charge hooks)
Posted by Pavel Emelianov on Wed, 06 Sep 2006 08:45:01 GMT
View Forum Message <> Reply to Message

Nick Piggin wrote:
> Kirill Korotaev wrote:
>
>> Introduce calls to BC core over the kernel to charge locked memory.
>>
>> Normaly new locked piece of memory may appear in insert_vm_struct,
>> but there are places (do_mmap_pgoff, dup_mmap etc) when new vma
>> is not inserted by insert_vm_struct(), but either link_vma-ed or
>> merged with some other - these places call BC code explicitly.
>>
>> Plus sys_mlock[all] itself has to be patched to charge/uncharge
>> needed amount of pages.
>
>
> I still haven't heard your good reasons why such a complex scheme is
> required when my really simple proposal of unconditionally charging
> the page to the container it was allocated by.
Charging the page to the container it was allocated in is a possible and
correct way, we agree, but how does this comment refer to locked pages
accounting?
>
> That has the benefit of not being full of user explotable holes and
> also not putting such a huge burden on mm/ and the wider kernel in
> general.

Subject: Re: [ckrm-tech] [PATCH 5/13] BC: user interface (syscalls)
Posted by Balbir Singh on Wed, 06 Sep 2006 08:57:38 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Balbir Singh wrote:
>>> +
>>> +asmlinkage long sys_set_bcid(bcid_t id)

>>> +{
>>> +   int error;
>>> +   struct beancounter *bc;
>>> +   struct task_beancounter *task_bc;
>>> +
>>> +   task_bc = &current->task_bc;
>> I was playing around with the bc patches and found that to make
>> use of bc's, I had to actually call set_bcid() and then exec() a
>> task/shell so that the id would stick around. Would you consider
> That sounds very strange as sys_set_bcid() actually changes current's
> exec_bc.
> One note is about mm's bc - mm obtains new bc only after fork or exec -
> that's
> true. But kmemsize starts charging right after the sys_set_bcid.

I was playing around only with kmemsize. I think the reason for my observation
is this

bash --> (my utility) --> set_bcid()

Since bash spawns my utility in a separate process, it creates and assigns
a bean counter to it and then my utility exits. Unless it spawns/exec()'s a
new shell, the beancounter is freed when the task exits (my utility).

>> changing sys_set_bcid to sys_set_task_bcid() or adding a new
>> system call sys_set_task_bcid()? We could pass the pid that we
>> intend to associate with the new id. This also means we'll need
>> locking around to protect task->task_bc.
>

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [PATCH 9/13] BC: locked pages (charge hooks)
Posted by Nick Piggin on Wed, 06 Sep 2006 09:41:05 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:

>Nick Piggin wrote:
>
>>Kirill Korotaev wrote:
>>

>>
>>>Introduce calls to BC core over the kernel to charge locked memory.
>>>
>>>Normaly new locked piece of memory may appear in insert_vm_struct,
>>>but there are places (do_mmap_pgoff, dup_mmap etc) when new vma
>>>is not inserted by insert_vm_struct(), but either link_vma-ed or
>>>merged with some other - these places call BC code explicitly.
>>>
>>>Plus sys_mlock[all] itself has to be patched to charge/uncharge
>>>needed amount of pages.
>>>
>>
>>I still haven't heard your good reasons why such a complex scheme is
>>required when my really simple proposal of unconditionally charging
>>the page to the container it was allocated by.
>>
>Charging the page to the container it was allocated in is a possible and
>correct way, we agree, but how does this comment refer to locked pages
>

If it is a possible and correct way, I'd must rather see *that* way
get tried first, and then made more complex or discarded if it is
found to be insufficient.

>accounting?
>

That's where I'd looked at enough mm/ stuff to decide that it wasn't
just my usual unjustified whining. Complexity of this approach is
quite... high.

Sorry if that wasn't clear.

--

Subject: Re: [ckrm-tech] [PATCH 5/13] BC: user interface (syscalls)
Posted by Pavel Emelianov on Wed, 06 Sep 2006 10:42:48 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelianov wrote:
>> Balbir Singh wrote:
>>>> +
>>>> +asmlinkage long sys_set_bcid(bcid_t id)
>>>> +{

>>>> +   int error;
>>>> +   struct beancounter *bc;
>>>> +   struct task_beancounter *task_bc;
>>>> +
>>>> +   task_bc = &current->task_bc;
>>> I was playing around with the bc patches and found that to make
>>> use of bc's, I had to actually call set_bcid() and then exec() a
>>> task/shell so that the id would stick around. Would you consider
>> That sounds very strange as sys_set_bcid() actually changes current's
>> exec_bc.
>> One note is about mm's bc - mm obtains new bc only after fork or exec -
>> that's
>> true. But kmemsize starts charging right after the sys_set_bcid.
>
> I was playing around only with kmemsize. I think the reason for my
> observation
> is this
>
> bash --> (my utility) --> set_bcid()
>
> Since bash spawns my utility in a separate process, it creates and
> assigns
> a bean counter to it and then my utility exits. Unless it
> spawns/exec()'s a
> new shell, the beancounter is freed when the task exits (my utility).
Well, beancounter is not "inherited" by parent task :)
After setting bcid you need to spawn/exec a new shell.
But seeting limits and getting stats is possible from the old shell
as well as from the new one.
>
>>> changing sys_set_bcid to sys_set_task_bcid() or adding a new
>>> system call sys_set_task_bcid()? We could pass the pid that we
>>> intend to associate with the new id. This also means we'll need
>>> locking around to protect task->task_bc.
>>
>
>

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by dev on Wed, 06 Sep 2006 13:04:20 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Kirill Korotaev wrote:
>
>> Core Resource Beancounters (BC) + kernel/user memory control.

>>
>> BC allows to account and control consumption
>> of kernel resources used by group of processes.
>>
>> Draft UBC description on OpenVZ wiki can be found at
>> http://wiki.openvz.org/UBC_parameters
>>
>> The full BC patch set allows to control:
>> - kernel memory. All the kernel objects allocatable
>> on user demand should be accounted and limited
>> for DoS protection.
>> E.g. page tables, task structs, vmas etc.
>
>
> One of the key requirements of resource management for us is to be able to
> migrate tasks across resource groups. Since bean counters do not associate
> a list of tasks associated with them, I do not see how this can be done
> with the existing bean counters.
It was discussed multiple times already.
The key problem here is the objects which do not _belong_ to tasks.
e.g. IPC objects. They exist in global namespace and can't be reaccounted.
At least no one proposed the policy to reaccount.
And please note, IPCs are not the only such objects.

But I guess your comment mostly concerns user pages, yeah?
In this case reaccounting can be easily done using page beancounters
which are introduced in this patch set.
So if it is a requirement, then lets cooperate and create such functionality.

So for now I see 2 main requirements from people:
- memory reclamation
- tasks moving across beancounters

I agree with these requirements and lets move into this direction.
But moving so far can't be done without accepting:
1. core functionality
2. accounting

Thanks,
Kirill

Subject: Re: [ckrm-tech] [PATCH 5/13] BC: user interface (syscalls)
Posted by Balbir Singh on Wed, 06 Sep 2006 13:23:27 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Balbir Singh wrote:

>> Pavel Emelianov wrote:
>>> Balbir Singh wrote:
>>>>> +
>>>>> +asmlinkage long sys_set_bcid(bcid_t id)
>>>>> +{
>>>>> +   int error;
>>>>> +   struct beancounter *bc;
>>>>> +   struct task_beancounter *task_bc;
>>>>> +
>>>>> +   task_bc = &current->task_bc;
>>>> I was playing around with the bc patches and found that to make
>>>> use of bc's, I had to actually call set_bcid() and then exec() a
>>>> task/shell so that the id would stick around. Would you consider
>>> That sounds very strange as sys_set_bcid() actually changes current's
>>> exec_bc.
>>> One note is about mm's bc - mm obtains new bc only after fork or exec -
>>> that's
>>> true. But kmemsize starts charging right after the sys_set_bcid.
>> I was playing around only with kmemsize. I think the reason for my
>> observation
>> is this
>>
>> bash --> (my utility) --> set_bcid()
>>
>> Since bash spawns my utility in a separate process, it creates and
>> assigns
>> a bean counter to it and then my utility exits. Unless it
>> spawns/exec()'s a
>> new shell, the beancounter is freed when the task exits (my utility).
> Well, beancounter is not "inherited" by parent task :)
> After setting bcid you need to spawn/exec a new shell.
> But seeting limits and getting stats is possible from the old shell
> as well as from the new one.

That's what I suspected. I suggest changing the system call to allow adding any task to a particular id (not necessarily only the current one). It would help us group tasks to a particular id. It would also solve my problem of spawning a shell each time I decide to use a task with a beancounter and limits.

>>>> changing sys_set_bcid to sys_set_task_bcid() or adding a new
>>>> system call sys_set_task_bcid()? We could pass the pid that we
>>>> intend to associate with the new id. This also means we'll need
>>>> locking around to protect task->task_bc.
>>

--

Balbir Singh,

Subject: Re: [ckrm-tech] [PATCH 5/13] BC: user interface (syscalls)
Posted by Balbir Singh on Wed, 06 Sep 2006 13:45:44 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
> Add the following system calls for BC management:
> 1. sys_get_bcid   - get current BC id
> 2. sys_set_bcid   - change exec_ and fork_ BCs on current
> 3. sys_set_bclimit - set limits for resources consumtions
> 4. sys_get_bcstat   - return br_resource_parm on resource
>
> Signed-off-by: Pavel Emelianov <xemul@sw.ru>
> Signed-off-by: Kirill Korotaev <dev@sw.ru>
>
> --- ./include/asm-powerpc/systbl.h.bcsys 2006-07-10 12:39:19.000000000 +0400
> +++ ./include/asm-powerpc/systbl.h 2006-09-05 12:47:21.000000000 +0400
> @@ -304,3 +304,7 @@ SYSCALL_SPU(fchmodat)
> SYSCALL_SPU(faccessat)
> COMPAT_SYS_SPU(get_robust_list)
> COMPAT_SYS_SPU(set_robust_list)
> +SYSCALL(sys_get_bcid)
> +SYSCALL(sys_set_bcid)
> +SYSCALL(sys_set_bclimit)
> +SYSCALL(sys_get_bcstat)

Fix a build error for powerpc boxes. While compiling on powerpc, Vaidyanathan Srinivasan caught this error. System calls on powerpc do not need sys_ prefix.

Signed-off-by: Balbir Singh <balbir@in.ibm.com>
Signed-off-by: Vaidyanathan Srinivasan <svaidy@in.ibm.com>
---

 include/asm-powerpc/systbl.h | 8 ++++----
 1 files changed, 4 insertions(+), 4 deletions(-)

diff -puN include/asm-powerpc/systbl.h~fix-powerpc-build
include/asm-powerpc/systbl.h
--- linux-2.6.18-rc5/include/asm-powerpc/systbl.h~fix-powerpc-bu ild 2006-09-06
19:03:18.000000000 +0530
+++ linux-2.6.18-rc5-balbir/include/asm-powerpc/systbl.h 2006-09-06
19:03:38.000000000 +0530
@@ -304,7 +304,7 @@ SYSCALL_SPU(fchmodat)
 SYSCALL_SPU(faccessat)

```
  COMPAT_SYS_SPU(get_robust_list)
  COMPAT_SYS_SPU(set_robust_list)
-SYSCALL(sys_get_bcid)
-SYSCALL(sys_set_bcid)
-SYSCALL(sys_set_bclimit)
-SYSCALL(sys_get_bcstat)
+SYSCALL(get_bcid)
+SYSCALL(set_bcid)
+SYSCALL(set_bclimit)
+SYSCALL(get_bcstat)
```

_

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by dev on Wed, 06 Sep 2006 13:54:13 GMT
View Forum Message <> Reply to Message

> On Tue, 2006-09-05 at 19:02 +0400, Kirill Korotaev wrote:
>
>>Core Resource Beancounters (BC) + kernel/user memory control.
>>
>>BC allows to account and control consumption
>>of kernel resources used by group of processes.
>
>
> Hi Kirill,
>
> I've honestly lost track of these discussions along the way, so I hope
> you don't mind summarizing a bit.
I think we need to create wiki to summarize it once and forever.
http://wiki.openvz.org/UBC_discussion

> Do these patches help with accounting for anything other than memory?
this patch set - no, but the complete one - does:
* numfile
* numptys
* numsocks (TCP, other, etc.)
* numtasks
* numflocks
...
this list of resources was chosen to make sure that no DoS from the container

is possible.
This list is extensible easily and if resource is out of interest than
its limits can be set to unlimited.

> Will we need new user/kernel interfaces for cpu, i/o bandwidth, etc...?
no. no new interfaces are required.

BUT: I remind you the talks at OKS/OLS and in previous UBC discussions.
It was noted that having a separate interfaces for CPU, I/O bandwidth
and memory maybe worthwhile. BTW, I/O bandwidth already has a separate
interface :/

> Have you given any thought to the possibility that a task might need to
> move between accounting contexts? That has certainly been a
> "requirement" pushed on to CKRM for a long time, and the need goes
> something like this:
Yes we thought about this and this is no more problematic for BC
than for CKRM. See my explanation below.

> 1. A system runs a web server, which services several virtual domains
> 2. that web server receives a request for foo.com
> 3. the web server switches into foo.com's accounting context
> 4. the web server reads things from disk, allocates some memory, and
>   makes a database request.
> 5. the database receives the request, and switches into foo.com's
>   accounting context, and charges foo.com for its resource use
> etc...
The question is - whether web server is multithreaded or not...
If it is not - then no problem here, you can change current
context and new resources will be charged accordingly.

And current BC code is _able_ to handle it with _minor_ changes.
(One just need to save bc not on mm struct, but rather on vma struct
and change mm->bc on set_bc_id()).

However, no one (can some one from CKRM team please?) explained so far
what to do with threads. Consider the following example.

1. Threaded web server spawns a child to serve a client.
2. child thread touches some pages and they are charged to child BC
   (which differs from parent's one)
3. child exits, but since its mm is shared with parent, these pages
   stay mapped and charged to child BC.

So the question is: what to do with these pages?
- should we recharge them to another BC?
- leave them charged?

> So, the goal is to run _one_ copy of an application on a system, but
> account for its resources in a much more fine-grained way than at the
> application level.
Yes.

> I think we can probably use beancounters for this, if we do not worry
> about migrating _existing_ charges when we change accounting context.
> Does that make sense?
exactly. thats what I'm saying. we can use beancounters for this
if charges are kept for creator.

Thanks,
Kirill

Subject: Re: [PATCH 11/13] BC: vmrss (preparations)
Posted by dev on Wed, 06 Sep 2006 13:56:03 GMT
View Forum Message <> Reply to Message

Thanks a lot!!!

> Kirill Korotaev wrote:
>
> <snip>
>
>>--- ./include/bc/beancounter.h.bcvmrssprep 2006-09-05
>>13:17:50.000000000 +0400
>>+++ ./include/bc/beancounter.h 2006-09-05 13:44:33.000000000 +0400
>>@@ -45,6 +45,13 @@ struct bc_resource_parm {
>>#define BC_MAXVALUE LONG_MAX
>>
>>/*
>>+ * This magic is used to distinuish user beancounter and pages beancounter
>>+ * in struct page. page_ub and page_bc are placed in union and MAGIC
>>+ * ensures us that we don't use pbc as ubc in bc_page_uncharge().
>>+ */
>>+#define BC_MAGIC 0x62756275UL
>>+
>>+/*
>> * Resource management structures
>> * Serialization issues:
>> * beancounter list management is protected via bc_hash_lock
>>@@ -54,11 +61,13 @@ struct bc_resource_parm {
>> */
>>
>>struct beancounter {
>>+ unsigned long bc_magic;
>> atomic_t bc_refcount;

>>   spinlock_t   bc_lock;
>>   bcid_t   bc_id;
>>   struct hlist_node   hash;
>>
>>+   unsigned long   unused_privvmpages;
>>   /* resources statistics and settings */
>>   struct bc_resource_parm   bc_parms[BC_RESOURCES];
>>};
>>@@ -74,6 +83,8 @@ enum bc_severity { BC_BARRIER, BC_LIMIT,
>>
>>#ifdef CONFIG_BEANCOUNTERS
>>
>>+extern unsigned int nr_beancounters = 1;
>>+
>
>
> my gcc doesn't like this one ...
>
> regards,
>
> C.
>
> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
>
> ---
> include/bc/beancounter.h | 2 +-
> kernel/bc/beancounter.c | 2 +-
> 2 files changed, 2 insertions(+), 2 deletions(-)
>
> Index: 2.6.18-rc5-mm1/include/bc/beancounter.h
> === =======
> --- 2.6.18-rc5-mm1.orig/include/bc/beancounter.h
> +++ 2.6.18-rc5-mm1/include/bc/beancounter.h
> @@ -86,7 +86,7 @@ enum bc_severity { BC_BARRIER, BC_LIMIT,
>
> #ifdef CONFIG_BEANCOUNTERS
>
> -extern unsigned int nr_beancounters = 1;
> +extern unsigned int nr_beancounters;
>
> /*
>   * These functions tune minheld and maxheld values for a given
> Index: 2.6.18-rc5-mm1/kernel/bc/beancounter.c
> === =======
> --- 2.6.18-rc5-mm1.orig/kernel/bc/beancounter.c
> +++ 2.6.18-rc5-mm1/kernel/bc/beancounter.c
> @@ -20,7 +20,7 @@ static void init_beancounter_struct(stru
>

> struct beancounter init_bc;
>
> -unsigned int nr_beancounters;
> +unsigned int nr_beancounters = 1;
>
> const char *bc_rnames[] = {
>   "kmemsize", /* 0 */
>

Subject: Re: [ckrm-tech] [PATCH 9/13] BC: locked pages (charge hooks)
Posted by dev on Wed, 06 Sep 2006 14:16:42 GMT
View Forum Message <> Reply to Message

Nick,

> Kirill Korotaev wrote:
>
>
>>Introduce calls to BC core over the kernel to charge locked memory.
>>
>>Normaly new locked piece of memory may appear in insert_vm_struct,
>>but there are places (do_mmap_pgoff, dup_mmap etc) when new vma
>>is not inserted by insert_vm_struct(), but either link_vma-ed or
>>merged with some other - these places call BC code explicitly.
>>
>>Plus sys_mlock[all] itself has to be patched to charge/uncharge
>>needed amount of pages.
>
>
>
> I still haven't heard your good reasons why such a complex scheme is
> required when my really simple proposal of unconditionally charging
> the page to the container it was allocated by.
Nick can you elaborate what your proposal is?
Probably I missed it somewhere...

> That has the benefit of not being full of user explotable holes and
> also not putting such a huge burden on mm/ and the wider kernel in
> general.
I guess you will have to account locked pages still and
thus complexity won't be reduced much in this regard...

Thanks,
Kirill

Subject: Re: [PATCH 7/13] BC: kernel memory (marks)
Posted by Cedric Le Goater on Wed, 06 Sep 2006 14:19:15 GMT
View Forum Message <> Reply to Message

Minor issue bellow in arch/ia64/mm/init.c. I'm not sure what the charge
argument should be. Please check.

Regards,

C.

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>

---
 arch/ia64/mm/init.c | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

Index: 2.6.18-rc5-mm1/arch/ia64/mm/init.c
 === =======
--- 2.6.18-rc5-mm1.orig/arch/ia64/mm/init.c
+++ 2.6.18-rc5-mm1/arch/ia64/mm/init.c
@@ -95,7 +95,7 @@ check_pgt_cache(void)
        preempt_disable();
        while (unlikely((pages_to_free = min_pages_to_free()) > 0)) {
                while (pages_to_free--) {
-                       free_page((unsigned long)pgtable_quicklist_alloc());
+   free_page((unsigned long)pgtable_quicklist_alloc(0));
                }
                preempt_enable();
                preempt_disable();

Subject: Re: [ckrm-tech] [PATCH 5/13] BC: user interface (syscalls)
Posted by dev on Wed, 06 Sep 2006 14:20:34 GMT
View Forum Message <> Reply to Message

thanks a lot!

> Kirill Korotaev wrote:
>
>>Add the following system calls for BC management:
>> 1. sys_get_bcid   - get current BC id
>> 2. sys_set_bcid   - change exec_ and fork_ BCs on current
>> 3. sys_set_bclimit - set limits for resources consumtions
>> 4. sys_get_bcstat   - return br_resource_parm on resource
>>
>>Signed-off-by: Pavel Emelianov <xemul@sw.ru>
>>Signed-off-by: Kirill Korotaev <dev@sw.ru>

>>
>>--- ./include/asm-powerpc/systbl.h.bcsys 2006-07-10 12:39:19.000000000 +0400
>>+++ ./include/asm-powerpc/systbl.h 2006-09-05 12:47:21.000000000 +0400
>>@@ -304,3 +304,7 @@ SYSCALL_SPU(fchmodat)
>> SYSCALL_SPU(faccessat)
>> COMPAT_SYS_SPU(get_robust_list)
>> COMPAT_SYS_SPU(set_robust_list)
>>+SYSCALL(sys_get_bcid)
>>+SYSCALL(sys_set_bcid)
>>+SYSCALL(sys_set_bclimit)
>>+SYSCALL(sys_get_bcstat)
>
>
>
> Fix a build error for powerpc boxes. While compiling on powerpc, Vaidyanathan
> Srinivasan caught this error. System calls on powerpc do not need sys_ prefix.
>
> Signed-off-by: Balbir Singh <balbir@in.ibm.com>
> Signed-off-by: Vaidyanathan Srinivasan <svaidy@in.ibm.com>
> ---
>
> include/asm-powerpc/systbl.h | 8 ++++----
> 1 files changed, 4 insertions(+), 4 deletions(-)
>
> diff -puN include/asm-powerpc/systbl.h~fix-powerpc-build
> include/asm-powerpc/systbl.h
> --- linux-2.6.18-rc5/include/asm-powerpc/systbl.h~fix-powerpc-bu ild 2006-09-06
> 19:03:18.000000000 +0530
> +++ linux-2.6.18-rc5-balbir/include/asm-powerpc/systbl.h 2006-09-06
> 19:03:38.000000000 +0530
> @@ -304,7 +304,7 @@ SYSCALL_SPU(fchmodat)
> SYSCALL_SPU(faccessat)
> COMPAT_SYS_SPU(get_robust_list)
> COMPAT_SYS_SPU(set_robust_list)
> -SYSCALL(sys_get_bcid)
> -SYSCALL(sys_set_bcid)
> -SYSCALL(sys_set_bclimit)
> -SYSCALL(sys_get_bcstat)
> +SYSCALL(get_bcid)
> +SYSCALL(set_bcid)
> +SYSCALL(set_bclimit)
> +SYSCALL(get_bcstat)
> _
>

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user

memory)
Posted by Balbir Singh on Wed, 06 Sep 2006 19:17:56 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
> Balbir Singh wrote:
>> Kirill Korotaev wrote:
>>
>>> Core Resource Beancounters (BC) + kernel/user memory control.
>>>
>>> BC allows to account and control consumption
>>> of kernel resources used by group of processes.
>>>
>>> Draft UBC description on OpenVZ wiki can be found at
>>> http://wiki.openvz.org/UBC_parameters
>>>
>>> The full BC patch set allows to control:
>>> - kernel memory. All the kernel objects allocatable
>>> on user demand should be accounted and limited
>>> for DoS protection.
>>> E.g. page tables, task structs, vmas etc.
>>
>> One of the key requirements of resource management for us is to be able to
>> migrate tasks across resource groups. Since bean counters do not associate
>> a list of tasks associated with them, I do not see how this can be done
>> with the existing bean counters.
> It was discussed multiple times already.
> The key problem here is the objects which do not _belong_ to tasks.
> e.g. IPC objects. They exist in global namespace and can't be reaccounted.
> At least no one proposed the policy to reaccount.
> And please note, IPCs are not the only such objects.
>
> But I guess your comment mostly concerns user pages, yeah?

Yes.

> In this case reaccounting can be easily done using page beancounters
> which are introduced in this patch set.
> So if it is a requirement, then lets cooperate and create such functionality.
>

Sure, let's cooperate and talk.

> So for now I see 2 main requirements from people:
> - memory reclamation
> - tasks moving across beancounters
>

Some not quite so urgent ones - like support for guarantees. I think this can

be worked out as we make progress.

> I agree with these requirements and lets move into this direction.
> But moving so far can't be done without accepting:
> 1. core functionality
> 2. accounting
>

Some of the core functionality might be a limiting factor for the requirements.
Lets agree on the requirements, I think its a great step forward and then
build the core functionality with these requirements in mind.

> Thanks,
> Kirill
>
--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Wed, 06 Sep 2006 21:47:34 GMT
View Forum Message <> Reply to Message

On Wed, 2006-09-06 at 17:06 +0400, Kirill Korotaev wrote:
> Balbir Singh wrote:
> > Kirill Korotaev wrote:
> >
> >> Core Resource Beancounters (BC) + kernel/user memory control.
> >>
> >> BC allows to account and control consumption
> >> of kernel resources used by group of processes.
> >>
> >> Draft UBC description on OpenVZ wiki can be found at
> >> http://wiki.openvz.org/UBC_parameters
> >>
> >> The full BC patch set allows to control:
> >> - kernel memory. All the kernel objects allocatable
> >> on user demand should be accounted and limited
> >> for DoS protection.
> >> E.g. page tables, task structs, vmas etc.
> >
> >
> > One of the key requirements of resource management for us is to be able to
> > migrate tasks across resource groups. Since bean counters do not associate

> > a list of tasks associated with them, I do not see how this can be done
> > with the existing bean counters.
> It was discussed multiple times already.
> The key problem here is the objects which do not _belong_ to tasks.
> e.g. IPC objects. They exist in global namespace and can't be reaccounted.
> At least no one proposed the policy to reaccount.
> And please note, IPCs are not the only such objects.

>From implementation point of view I do not see it to be any different
than how it can be done under UBC.

AFAICS, beancounters are associated with tasks not those "objects".
Those "objects" get their bc through some association with a task. The
same can be done in the other case also.

If my understanding is wrong, please tell me how one can associate such
"object" to a bc.

>
> But I guess your comment mostly concerns user pages, yeah?
> In this case reaccounting can be easily done using page beancounters
> which are introduced in this patch set.
> So if it is a requirement, then lets cooperate and create such functionality.

hmm... that is what I thought I was doing when I was replying on these
threads. May be I should have waited for this "call for co-operation"
before jumping on it :)

>
> So for now I see 2 main requirements from people:
> - memory reclamation
> - tasks moving across beancounters

Please consider the requirements I listed before
 http://marc.theaimsgroup.com/?l=ckrm-tech&m=115593001810 616&w=2

>
> I agree with these requirements and lets move into this direction.
> But moving so far can't be done without accepting:
> 1. core functionality
> 2. accounting

I agree that discussion need to happen on the core functionality and
interface.
>
> Thanks,
> Kirill
>

>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Wed, 06 Sep 2006 21:54:01 GMT
View Forum Message <> Reply to Message

On Wed, 2006-09-06 at 17:57 +0400, Kirill Korotaev wrote:
> > On Tue, 2006-09-05 at 19:02 +0400, Kirill Korotaev wrote:
> >
> >>Core Resource Beancounters (BC) + kernel/user memory control.
> >>
> >>BC allows to account and control consumption
> >>of kernel resources used by group of processes.
> >
> >
> > Hi Kirill,
> >
> > I've honestly lost track of these discussions along the way, so I hope
> > you don't mind summarizing a bit.
> I think we need to create wiki to summarize it once and forever.
> http://wiki.openvz.org/UBC_discussion
>
> > Do these patches help with accounting for anything other than memory?
> this patch set - no, but the complete one - does:
> * numfile
> * numptys
> * numsocks (TCP, other, etc.)
> * numtasks
> * numflocks
> ...
> this list of resources was chosen to make sure that no DoS from the container
> is possible.

> This list is extensible easily and if resource is out of interest than
> its limits can be set to unlimited.
>
> > Will we need new user/kernel interfaces for cpu, i/o bandwidth, etc...?
> no. no new interfaces are required.

Good to know that.

Your CPU controller supports guarantee ?

Do you have a i/o controller ?

>
> BUT: I remind you the talks at OKS/OLS and in previous UBC discussions.
> It was noted that having a separate interfaces for CPU, I/O bandwidth

But, it will be lot simpler for the user to configure/use if they are
together. We should discuss this also.

> and memory maybe worthwhile. BTW, I/O bandwidth already has a separate
> interface :/
>
> > Have you given any thought to the possibility that a task might need to
> > move between accounting contexts? That has certainly been a
> > "requirement" pushed on to CKRM for a long time, and the need goes
> > something like this:
> Yes we thought about this and this is no more problematic for BC
> than for CKRM. See my explanation below.
>
> > 1. A system runs a web server, which services several virtual domains
> > 2. that web server receives a request for foo.com
> > 3. the web server switches into foo.com's accounting context
> > 4. the web server reads things from disk, allocates some memory, and
> > makes a database request.
> > 5. the database receives the request, and switches into foo.com's
> > accounting context, and charges foo.com for its resource use
> > etc...
> The question is - whether web server is multithreaded or not...
> If it is not - then no problem here, you can change current
> context and new resources will be charged accordingly.
>
> And current BC code is _able_ to handle it with _minor_ changes.
> (One just need to save bc not on mm struct, but rather on vma struct
> and change mm->bc on set_bc_id()).
>
> However, no one (can some one from CKRM team please?) explained so far
> what to do with threads. Consider the following example.
>

> 1. Threaded web server spawns a child to serve a client.
> 2. child thread touches some pages and they are charged to child BC
>   (which differs from parent's one)
> 3. child exits, but since its mm is shared with parent, these pages
>   stay mapped and charged to child BC.
>
> So the question is: what to do with these pages?
> - should we recharge them to another BC?
> - leave them charged?

Leave them charged. It will be charged to the appropriate UBC when they
touch it again.

>
> > So, the goal is to run _one_ copy of an application on a system, but
> > account for its resources in a much more fine-grained way than at the
> > application level.
> Yes.
>
> > I think we can probably use beancounters for this, if we do not worry
> > about migrating _existing_ charges when we change accounting context.
> > Does that make sense?
> exactly. thats what I'm saying. we can use beancounters for this
> if charges are kept for creator.
>
> Thanks,
> Kirill
>
>
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Wed, 06 Sep 2006 22:06:11 GMT

View Forum Message <> Reply to Message

On Thu, 2006-09-07 at 00:47 +0530, Balbir Singh wrote:

<snip>
>
> Some not quite so urgent ones - like support for guarantees. I think this can

IMO, guarantee support should be considered to be part of the
infrastructure. Controller functionalities/implementation will be
different with/without guarantee support. In other words, adding
guarantee feature later will cause re-implementations.

> be worked out as we make progress.
>
> > I agree with these requirements and lets move into this direction.
> > But moving so far can't be done without accepting:
> > 1. core functionality
> > 2. accounting
> >
>
> Some of the core functionality might be a limiting factor for the requirements.
> Lets agree on the requirements, I think its a great step forward and then
> build the core functionality with these requirements in mind.
>
> > Thanks,
> > Kirill
> >
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Balbir Singh on Thu, 07 Sep 2006 03:08:48 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:
> On Thu, 2006-09-07 at 00:47 +0530, Balbir Singh wrote:
>
> <snip>
>> Some not quite so urgent ones - like support for guarantees. I think this can
>
> IMO, guarantee support should be considered to be part of the

> infrastructure. Controller functionalities/implementation will be
> different with/without guarantee support. In other words, adding
> guarantee feature later will cause re-implementations.

Thanks for pointing this out. Thats what I implied in the comment below.

>
>> be worked out as we make progress.
>>
>>> I agree with these requirements and lets move into this direction.
>>> But moving so far can't be done without accepting:
>>> 1. core functionality
>>> 2. accounting
>>>
>> Some of the core functionality might be a limiting factor for the requirements.
>> Lets agree on the requirements, I think its a great step forward and then
>> build the core functionality with these requirements in mind.
>>
>>> Thanks,
>>> Kirill
>>>

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Thu, 07 Sep 2006 07:29:17 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:

[snip]
>>> Will we need new user/kernel interfaces for cpu, i/o bandwidth, etc...?
>>>
>> no. no new interfaces are required.
>>
>
> Good to know that.
>
> Your CPU controller supports guarantee ?
>
It does, but CPU controller is not so simple as memory one.

> Do you have a i/o controller ?
>
>
>> BUT: I remind you the talks at OKS/OLS and in previous UBC discussions.
>> It was noted that having a separate interfaces for CPU, I/O bandwidth
>>
>
> But, it will be lot simpler for the user to configure/use if they are
> together. We should discuss this also.
>
IMHO such unification may only imply that one syscall is used to pass
configuration info into kernel.
Each controller has specific configurating parameters different from the
other ones. E.g. CPU controller must assign a "weight" to each group to
share CPU time accordingly, but what is a "weight" for memory controller?
IO may operate on "bandwidth" and it's not clear what is a "bandwidth" in
Kb/sec for CPU controller and so on.

[snip]
>> The question is - whether web server is multithreaded or not...
>> If it is not - then no problem here, you can change current
>> context and new resources will be charged accordingly.
>>
>> And current BC code is _able_ to handle it with _minor_ changes.
>> (One just need to save bc not on mm struct, but rather on vma struct
>> and change mm->bc on set_bc_id()).
>>
>> However, no one (can some one from CKRM team please?) explained so far
>> what to do with threads. Consider the following example.
>>
>> 1. Threaded web server spawns a child to serve a client.
>> 2. child thread touches some pages and they are charged to child BC
>>   (which differs from parent's one)
>> 3. child exits, but since its mm is shared with parent, these pages
>>   stay mapped and charged to child BC.
>>
>> So the question is: what to do with these pages?
>> - should we recharge them to another BC?
>> - leave them charged?
>>
>
> Leave them charged. It will be charged to the appropriate UBC when they
> touch it again.
>
Do you mean that page must be re-charged each time someone touches it?

Subject: Re: [ckrm-tech] [PATCH 11/13] BC: vmrss (preparations)
Posted by Balbir Singh on Thu, 07 Sep 2006 16:28:45 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:
> This patch does simple things:
> - intruduces an bc_magic field on beancunter to make sure
> union on struct page is correctly used in next patches
> - adds nr_beancounters
> - adds unused_privvmpages variable (counter of privvm pages
> which are not mapped into VM address space and thus potentially
> can be allocated later)
>
> +static inline void privvm_uncharge(struct beancounter *bc, unsigned long sz)
> +{
> + if (unlikely(bc->unused_privvmpages < sz)) {
> + printk("BC: overuncharging %d unused pages: val %lu held %lu\n",
> +   bc->bc_id, sz, bc->unused_privvmpages);

I hit this path, when I do not enable CONFIG_BEANCOUNTERS_RSS. I suspect it has something to do with the code in mod_rss_pages(). I suspect the that CONFIG_BEANCOUNTERS_RSS needs to be enabled to get the accounting right.

In addition, Could you please make this a warning with KERN_WARNING.

> + sz = bc->unused_privvmpages;
> + }
> + bc->unused_privvmpages -= sz;
> + bc_update_privvmpages(bc);
> +}
> +
--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Thu, 07 Sep 2006 19:16:56 GMT
View Forum Message <> Reply to Message

On Thu, 2006-09-07 at 11:29 +0400, Pavel Emelianov wrote:
> Chandra Seetharaman wrote:
>
> [snip]
> >>> Will we need new user/kernel interfaces for cpu, i/o bandwidth, etc...?

> >>>
> >> no. no new interfaces are required.
> >>
> >
> > Good to know that.
> >
> > Your CPU controller supports guarantee ?
> >
> It does, but CPU controller is not so simple as memory one.

Hmm... the reason I asked is that the UBC infrastructure doesn't provide
guarantee support and Kirill mentioned there is no changes required to
UBC if you have to move your CPU controller to be under UBC.

>From your reply it does look like you need to make some changes (add
guarantee support) to UBC, if you want to move the CPU controller to be
under UBC.

> > Do you have a i/o controller ?
> >
> >
> >> BUT: I remind you the talks at OKS/OLS and in previous UBC discussions.
> >> It was noted that having a separate interfaces for CPU, I/O bandwidth
> >>
> >
> > But, it will be lot simpler for the user to configure/use if they are
> > together. We should discuss this also.
> >
> IMHO such unification may only imply that one syscall is used to pass
> configuration info into kernel.
> Each controller has specific configurating parameters different from the
> other ones. E.g. CPU controller must assign a "weight" to each group to
> share CPU time accordingly, but what is a "weight" for memory controller?
> IO may operate on "bandwidth" and it's not clear what is a "bandwidth" in
> Kb/sec for CPU controller and so on.
>
> [snip]
> >> The question is - whether web server is multithreaded or not...
> >> If it is not - then no problem here, you can change current
> >> context and new resources will be charged accordingly.
> >>
> >> And current BC code is _able_ to handle it with _minor_ changes.
> >> (One just need to save bc not on mm struct, but rather on vma struct
> >> and change mm->bc on set_bc_id()).
> >>
> >> However, no one (can some one from CKRM team please?) explained so far
> >> what to do with threads. Consider the following example.
> >>

> >> 1. Threaded web server spawns a child to serve a client.
> >> 2. child thread touches some pages and they are charged to child BC
> >>   (which differs from parent's one)
> >> 3. child exits, but since its mm is shared with parent, these pages
> >>   stay mapped and charged to child BC.
> >>
> >> So the question is: what to do with these pages?
> >> - should we recharge them to another BC?
> >> - leave them charged?
> >>
> >
> > Leave them charged. It will be charged to the appropriate UBC when they
> > touch it again.
> >
> Do you mean that page must be re-charged each time someone touches it?

What I meant is that to leave them charged, and if when they are
ummapped and mapped later, charge it to the appropriate BC.

--

 -- ----------
   Chandra Seetharaman | Be careful what you choose....
           - sekharan@us.ibm.com | .....you may get it.
 -- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Thu, 07 Sep 2006 19:29:15 GMT
View Forum Message <> Reply to Message

On Thu, 2006-09-07 at 11:29 +0400, Pavel Emelianov wrote:
<snip>

> >> BUT: I remind you the talks at OKS/OLS and in previous UBC discussions.
> >> It was noted that having a separate interfaces for CPU, I/O bandwidth
> >>
> >
> > But, it will be lot simpler for the user to configure/use if they are
> > together. We should discuss this also.
> >
> IMHO such unification may only imply that one syscall is used to pass
> configuration info into kernel.
> Each controller has specific configurating parameters different from the
> other ones. E.g. CPU controller must assign a "weight" to each group to
> share CPU time accordingly, but what is a "weight" for memory controller?
> IO may operate on "bandwidth" and it's not clear what is a "bandwidth" in

> Kb/sec for CPU controller and so on.

CKRM/RG handles this by eliminating the units from the interface and abstracting them to be "shares". Each resource controller converts the shares to its own units and handles properly.

User can specify the quantities simply as a percentage. CPU controller would treat it as cycles/ticks (within a time), memory controller would treat it as number of pages, and I/O controller would treat it as bandwidth, and so on...

<snip>
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Fri, 08 Sep 2006 07:22:47 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:

[snip]
>>>> The question is - whether web server is multithreaded or not...
>>>> If it is not - then no problem here, you can change current
>>>> context and new resources will be charged accordingly.
>>>>
>>>> And current BC code is _able_ to handle it with _minor_ changes.
>>>> (One just need to save bc not on mm struct, but rather on vma struct
>>>> and change mm->bc on set_bc_id()).
>>>>
>>>> However, no one (can some one from CKRM team please?) explained so far
>>>> what to do with threads. Consider the following example.
>>>>
>>>> 1. Threaded web server spawns a child to serve a client.
>>>> 2. child thread touches some pages and they are charged to child BC
>>>> (which differs from parent's one)
>>>> 3. child exits, but since its mm is shared with parent, these pages
>>>> stay mapped and charged to child BC.
>>>>
>>>> So the question is: what to do with these pages?
>>>> - should we recharge them to another BC?
>>>> - leave them charged?

>>>>
>>>>
>>> Leave them charged. It will be charged to the appropriate UBC when they
>>> touch it again.
>>>
>>>
>> Do you mean that page must be re-charged each time someone touches it?
>>
>
> What I meant is that to leave them charged, and if when they are
> ummapped and mapped later, charge it to the appropriate BC.
>
In this case multithreaded apache that tries to serve each domain in
separate BC will fill the memory with BC-s, held by pages allocated
and mapped in threads.

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Fri, 08 Sep 2006 07:26:34 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:
> On Thu, 2006-09-07 at 11:29 +0400, Pavel Emelianov wrote:
> <snip>
>
>
>>>> BUT: I remind you the talks at OKS/OLS and in previous UBC discussions.
>>>> It was noted that having a separate interfaces for CPU, I/O bandwidth
>>>>
>>>>
>>> But, it will be lot simpler for the user to configure/use if they are
>>> together. We should discuss this also.
>>>
>>>
>> IMHO such unification may only imply that one syscall is used to pass
>> configuration info into kernel.
>> Each controller has specific configurating parameters different from the
>> other ones. E.g. CPU controller must assign a "weight" to each group to
>> share CPU time accordingly, but what is a "weight" for memory controller?
>> IO may operate on "bandwidth" and it's not clear what is a "bandwidth" in
>> Kb/sec for CPU controller and so on.
>>
>
> CKRM/RG handles this by eliminating the units from the interface and
> abstracting them to be "shares". Each resource controller converts the
> shares to its own units and handles properly.
>

That's what I'm talking about - common syscall/ioct/etc and each controller
parses its input itself. That's OK for us.

[snip]

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Fri, 08 Sep 2006 07:33:00 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:
> On Thu, 2006-09-07 at 00:47 +0530, Balbir Singh wrote:
>
> <snip>
>> Some not quite so urgent ones - like support for guarantees. I think
>> this can
>
> IMO, guarantee support should be considered to be part of the
> infrastructure. Controller functionalities/implementation will be
> different with/without guarantee support. In other words, adding
> guarantee feature later will cause re-implementations.
I'm afraid we have different understandings of what a "guarantee" is.
Don't we?
Guarantee may be one of

1. container will be able to touch that number of pages
2. container will be able to sys_mmap() that number of pages
3. container will not be killed unless it touches that number of pages
4. anything else

Let's decide what kind of a guarantee we want.
>> be worked out as we make progress.
>>
>>> I agree with these requirements and lets move into this direction.
>>> But moving so far can't be done without accepting:
>>> 1. core functionality
>>> 2. accounting
>>>
>> Some of the core functionality might be a limiting factor for the requirements.
>> Lets agree on the requirements, I think its a great step forward and then
>> build the core functionality with these requirements in mind.
>>
>>> Thanks,
>>> Kirill
>>>

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Dave Hansen on Fri, 08 Sep 2006 15:30:50 GMT
View Forum Message <> Reply to Message

On Tue, 2006-09-05 at 17:17 -0700, Rohit Seth wrote:
> I'm wondering why not have different processes to serve different
> domains on the same physical server...particularly when they have
> different database to work on.

This is largely because this is I think how it is done today, and it has a lot of disadvantages. They also want to be able to account for traffic on the same database. Think of a large web hosting environment where you charged everyone (hundreds or thousands of users) by CPU and I/O bandwidth used at all levels of a given transaction.

> Is the amount of memory that you save by
> having a single copy that much useful that you are even okay to
> serialize the whole operation (What would happen, while the request for
> foo.com is getting worked on, there is another request for
> foo_bar.com...does it need to wait for foo.com request to get done
> before it can be served).

Let's put it this way. Enterprise databases can be memory pigs. It isn't feasible to run hundreds or thousands of copies on each machine.

-- Dave

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Dave Hansen on Fri, 08 Sep 2006 15:33:53 GMT
View Forum Message <> Reply to Message

On Wed, 2006-09-06 at 17:06 +0400, Kirill Korotaev wrote:
> It was discussed multiple times already.
> The key problem here is the objects which do not _belong_ to tasks.

Heh. The original CKRM patches didn't have a strong binding to tasks. They took it away to make them more mergeable. ;)

-- Dave

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Dave Hansen on Fri, 08 Sep 2006 15:43:41 GMT

On Fri, 2006-09-08 at 11:33 +0400, Pavel Emelianov wrote:
> I'm afraid we have different understandings of what a "guarantee" is.

It appears so.

> Don't we?
> Guarantee may be one of
>
> 1. container will be able to touch that number of pages
> 2. container will be able to sys_mmap() that number of pages
> 3. container will not be killed unless it touches that number of pages

A "death sentence" guarantee? I like it. :)

> 4. anything else
>
> Let's decide what kind of a guarantee we want.

I think of it as: "I will be allowed to use this many total pages, and they are guaranteed not to fail." (1), I think. The sum of all of the system's guarantees must be less than or equal to the amount of free memory on the machine.

If we knew to which NUMA node the memory was going to go, we might as well take the pages out of the allocator.

-- Dave

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Rohit Seth on Fri, 08 Sep 2006 17:10:37 GMT

On Fri, 2006-09-08 at 08:30 -0700, Dave Hansen wrote:
> On Tue, 2006-09-05 at 17:17 -0700, Rohit Seth wrote:
> > I'm wondering why not have different processes to serve different
> > domains on the same physical server...particularly when they have
> > different database to work on.
>
> This is largely because this is I think how it is done today, and it has
> a lot of disadvantages.

If it has lot of disadvantages then we should try to avoid that mechanism. Though I think it is okay to allow processes to be moved around with the clear expectation that it is a very heavy operation (as

I think at least all the anon pages should be moved too along with task)
and should not be generally done.

> They also want to be able to account for
> traffic on the same database. Think of a large web hosting environment
> where you charged everyone (hundreds or thousands of users) by CPU and
> I/O bandwidth used at all levels of a given transaction.
>
> > Is the amount of memory that you save by
> > having a single copy that much useful that you are even okay to
> > serialize the whole operation (What would happen, while the request for
> > foo.com is getting worked on, there is another request for
> > foo_bar.com...does it need to wait for foo.com request to get done
> > before it can be served).
>
> Let's put it this way. Enterprise databases can be memory pigs. It
> isn't feasible to run hundreds or thousands of copies on each machine.
>

The extra cost is probably the stack and private data segment...yes
there could be trade offs there depending on how big these segments are.
Though if there are big shared segments then that can be charged to a
single container.

Thanks,
-rohit

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Shailabh Nagar on Fri, 08 Sep 2006 17:26:53 GMT
View Forum Message <> Reply to Message

Rohit Seth wrote:
> On Fri, 2006-09-08 at 08:30 -0700, Dave Hansen wrote:
>> On Tue, 2006-09-05 at 17:17 -0700, Rohit Seth wrote:
>>> I'm wondering why not have different processes to serve different
>>> domains on the same physical server...particularly when they have
>>> different database to work on.
>> This is largely because this is I think how it is done today, and it has
>> a lot of disadvantages.
>
> If it has lot of disadvantages then we should try to avoid that
> mechanism. Though I think it is okay to allow processes to be moved
> around with the clear expectation that it is a very heavy operation (as
> I think at least all the anon pages should be moved too along with task)
> and should not be generally done.

>
>> They also want to be able to account for
>> traffic on the same database. Think of a large web hosting environment
>> where you charged everyone (hundreds or thousands of users) by CPU and
>> I/O bandwidth used at all levels of a given transaction.
>>
>>> Is the amount of memory that you save by
>>> having a single copy that much useful that you are even okay to
>>> serialize the whole operation (What would happen, while the request for
>>> foo.com is getting worked on, there is another request for
>>> foo_bar.com...does it need to wait for foo.com request to get done
>>> before it can be served).
>> Let's put it this way. Enterprise databases can be memory pigs. It
>> isn't feasible to run hundreds or thousands of copies on each machine.
>>
>
>
> The extra cost is probably the stack and private data segment...

Also maintenability, licensing, blah, blah.
Replicating the software stack for each service level one
wishes to provide, if avoidable as it seems to be, isn't such a good idea.
Same sort of reasoning for why containers make sense compared to Xen/VMWare
instances.

Memory resources, by their very nature, will be tougher to account when a
single database/app server services multiple clients and we can essentially
give up on that (taking the approach that only limited recharging can ever
be achieved). But cpu atleast is easy to charge correctly and since that will
also indirectly influence the requests for memory & I/O, its useful to allow
middleware to change the accounting base for a thread/task.

--Shailabh

> yes
> there could be trade offs there depending on how big these segments are.
> Though if there are big shared segments then that can be charged to a
> single container.

>
> Thanks,
> -rohit
>
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo

> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Balbir Singh on Fri, 08 Sep 2006 18:26:56 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Fri, 2006-09-08 at 11:33 +0400, Pavel Emelianov wrote:
>> I'm afraid we have different understandings of what a "guarantee" is.
>
> It appears so.
>
>> Don't we?
>> Guarantee may be one of
>>
>> 1. container will be able to touch that number of pages
>> 2. container will be able to sys_mmap() that number of pages
>> 3. container will not be killed unless it touches that number of pages
>
> A "death sentence" guarantee? I like it. :)
>
>> 4. anything else
>>
>> Let's decide what kind of a guarantee we want.

I think of guarantees w.r.t resources as the lower limit on the resource.
Guarantees and limits can be thought of as the range (guarantee, limit]
for the usage of the resource.

>
> I think of it as: "I will be allowed to use this many total pages, and
> they are guaranteed not to fail." (1), I think. The sum of all of the
> system's guarantees must be less than or equal to the amount of free
> memory on the machine.
>

Yes, totally agree.

> If we knew to which NUMA node the memory was going to go, we might as
> well take the pages out of the allocator.
>
> -- Dave
>

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Fri, 08 Sep 2006 19:07:22 GMT
View Forum Message <> Reply to Message

On Fri, 2006-09-08 at 11:22 +0400, Pavel Emelianov wrote:
> Chandra Seetharaman wrote:
>
> [snip]
> >>>> The question is - whether web server is multithreaded or not...
> >>>> If it is not - then no problem here, you can change current
> >>>> context and new resources will be charged accordingly.
> >>>>
> >>>> And current BC code is _able_ to handle it with _minor_ changes.
> >>>> (One just need to save bc not on mm struct, but rather on vma struct
> >>>> and change mm->bc on set_bc_id()).
> >>>>
> >>>> However, no one (can some one from CKRM team please?) explained so far
> >>>> what to do with threads. Consider the following example.
> >>>>
> >>>> 1. Threaded web server spawns a child to serve a client.
> >>>> 2. child thread touches some pages and they are charged to child BC
> >>>>   (which differs from parent's one)
> >>>> 3. child exits, but since its mm is shared with parent, these pages
> >>>>   stay mapped and charged to child BC.
> >>>>
> >>>> So the question is: what to do with these pages?
> >>>> - should we recharge them to another BC?
> >>>> - leave them charged?
> >>>>
> >>>>
> >>> Leave them charged. It will be charged to the appropriate UBC when they
> >>> touch it again.
> >>>
> >>>
> >> Do you mean that page must be re-charged each time someone touches it?
> >>
> >
> > What I meant is that to leave them charged, and if when they are
> > ummapped and mapped later, charge it to the appropriate BC.
> >

> In this case multithreaded apache that tries to serve each domain in
> separate BC will fill the memory with BC-s, held by pages allocated
> and mapped in threads.

I do not understand how the memory will be filled with BCs. Can you
explain, please.
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Fri, 08 Sep 2006 19:10:41 GMT
View Forum Message <> Reply to Message

On Fri, 2006-09-08 at 11:26 +0400, Pavel Emelianov wrote:
> Chandra Seetharaman wrote:
> > On Thu, 2006-09-07 at 11:29 +0400, Pavel Emelianov wrote:
> > <snip>
> >
> >
> >>>> BUT: I remind you the talks at OKS/OLS and in previous UBC discussions.
> >>>> It was noted that having a separate interfaces for CPU, I/O bandwidth
> >>>>
> >>>>
> >>> But, it will be lot simpler for the user to configure/use if they are
> >>> together. We should discuss this also.
> >>>
> >>>
> >> IMHO such unification may only imply that one syscall is used to pass
> >> configuration info into kernel.
> >> Each controller has specific configurating parameters different from the
> >> other ones. E.g. CPU controller must assign a "weight" to each group to
> >> share CPU time accordingly, but what is a "weight" for memory controller?
> >> IO may operate on "bandwidth" and it's not clear what is a "bandwidth" in
> >> Kb/sec for CPU controller and so on.
> >>
> >
> > CKRM/RG handles this by eliminating the units from the interface and
> > abstracting them to be "shares". Each resource controller converts the
> > shares to its own units and handles properly.
> >
> That's what I'm talking about - common syscall/ioct/etc and each controller
> parses its input itself. That's OK for us.

Yes, we can eliminate the "units"(KBs, cycles/ticks, pages etc.,) from
the interface and use a (unitless) number to specify the amount of
resource a resource group/container uses.
>
> [snip]
>
> --- ------------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

-- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
-- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Fri, 08 Sep 2006 19:23:44 GMT
View Forum Message <> Reply to Message

On Fri, 2006-09-08 at 11:33 +0400, Pavel Emelianov wrote:
> Chandra Seetharaman wrote:
> > On Thu, 2006-09-07 at 00:47 +0530, Balbir Singh wrote:
> >
> > <snip>
> >> Some not quite so urgent ones - like support for guarantees. I think
> >> this can
> >
> > IMO, guarantee support should be considered to be part of the
> > infrastructure. Controller functionalities/implementation will be
> > different with/without guarantee support. In other words, adding
> > guarantee feature later will cause re-implementations.
> I'm afraid we have different understandings of what a "guarantee" is.
> Don't we?

may be (I am not sure :), lets get it clarified.

> Guarantee may be one of
>
> 1. container will be able to touch that number of pages

>   2. container will be able to sys_mmap() that number of pages
>   3. container will not be killed unless it touches that number of pages
>   4. anything else

I would say (1) with slight modification
  "container will be able to touch _at least_ that number of pages"

Note that it is not only in the context of memory alone, it is generic
across resources.

For CPU it will be, "container will get _at least_ X ticks in Y seconds"

For number of tasks it will be, "container will get _at least_ X active
tasks at any point of time" and so on.

And as Dave pointed, sum of guarantees of all containers _can not_
exceed the total amount of that resource available at the system level.

>
> Let's decide what kind of a guarantee we want.
> >> be worked out as we make progress.
> >>
> >>> I agree with these requirements and lets move into this direction.
> >>> But moving so far can't be done without accepting:
> >>> 1. core functionality
> >>> 2. accounting
> >>>
> >> Some of the core functionality might be a limiting factor for the requirements.
> >> Lets agree on the requirements, I think its a great step forward and then
> >> build the core functionality with these requirements in mind.
> >>
> >>> Thanks,
> >>> Kirill
> >>>
>
>
>
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....

- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Rohit Seth on Fri, 08 Sep 2006 21:43:54 GMT
View Forum Message <> Reply to Message

On Fri, 2006-09-08 at 12:23 -0700, Chandra Seetharaman wrote:
> On Fri, 2006-09-08 at 11:33 +0400, Pavel Emelianov wrote:
> > Chandra Seetharaman wrote:
> > > On Thu, 2006-09-07 at 00:47 +0530, Balbir Singh wrote:
> > >
> > > <snip>
> > >> Some not quite so urgent ones - like support for guarantees. I think
> > >> this can
> > >
> > > IMO, guarantee support should be considered to be part of the
> > > infrastructure. Controller functionalities/implementation will be
> > > different with/without guarantee support. In other words, adding
> > > guarantee feature later will cause re-implementations.
> > I'm afraid we have different understandings of what a "guarantee" is.
> > Don't we?
>
> may be (I am not sure :), lets get it clarified.
>
> > Guarantee may be one of
> >
> > 1. container will be able to touch that number of pages
> > 2. container will be able to sys_mmap() that number of pages
> > 3. container will not be killed unless it touches that number of pages
> > 4. anything else
>
> I would say (1) with slight modification
> "container will be able to touch _at least_ that number of pages"
>

Does this scheme support running of tasks outside of containers on the
same platform where you have tasks running inside containers. If so
then how will you ensure processes running out side any container will
not leave less than the total guaranteed memory to different containers.

-rohit

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Mon, 11 Sep 2006 06:56:59 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Dave Hansen wrote:
>> On Fri, 2006-09-08 at 11:33 +0400, Pavel Emelianov wrote:
>>> I'm afraid we have different understandings of what a "guarantee" is.
>>
>> It appears so.
>>
>>> Don't we?
>>> Guarantee may be one of
>>>
>>> 1. container will be able to touch that number of pages
>>> 2. container will be able to sys_mmap() that number of pages
>>> 3. container will not be killed unless it touches that number of
>>> pages
>>
>> A "death sentence" guarantee? I like it. :)
>>
>>> 4. anything else
>>>
>>> Let's decide what kind of a guarantee we want.
>
> I think of guarantees w.r.t resources as the lower limit on the resource.
> Guarantees and limits can be thought of as the range (guarantee, limit]
> for the usage of the resource.
>
>>
>> I think of it as: "I will be allowed to use this many total pages, and
>> they are guaranteed not to fail." (1), I think. The sum of all of the
>> system's guarantees must be less than or equal to the amount of free
>> memory on the machine.
>
> Yes, totally agree.

Such a guarantee is really a limit and this limit is even harder than
BC's one :)

E.g. I have a node with 1Gb of ram and 10 containers with 100Mb
guarantee each.
I want to start one more. What shall I do not to break guarantees?

>
>> If we knew to which NUMA node the memory was going to go, we might as
>> well take the pages out of the allocator.
>>

>> -- Dave
>>

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Mon, 11 Sep 2006 07:02:06 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:
> On Fri, 2006-09-08 at 11:22 +0400, Pavel Emelianov wrote:
>
>> Chandra Seetharaman wrote:
>>
>> [snip]
>>
>>>>>> The question is - whether web server is multithreaded or not...
>>>>>> If it is not - then no problem here, you can change current
>>>>>> context and new resources will be charged accordingly.
>>>>>>
>>>>>> And current BC code is _able_ to handle it with _minor_ changes.
>>>>>> (One just need to save bc not on mm struct, but rather on vma struct
>>>>>> and change mm->bc on set_bc_id()).
>>>>>>
>>>>>> However, no one (can some one from CKRM team please?) explained so far
>>>>>> what to do with threads. Consider the following example.
>>>>>>
>>>>>> 1. Threaded web server spawns a child to serve a client.
>>>>>> 2. child thread touches some pages and they are charged to child BC
>>>>>>   (which differs from parent's one)
>>>>>> 3. child exits, but since its mm is shared with parent, these pages
>>>>>>   stay mapped and charged to child BC.
>>>>>>
>>>>>> So the question is: what to do with these pages?
>>>>>> - should we recharge them to another BC?
>>>>>> - leave them charged?
>>>>>>
>>>>>>
>>>>>>
>>>>> Leave them charged. It will be charged to the appropriate UBC when they
>>>>> touch it again.
>>>>>
>>>>>
>>>>>
>>>> Do you mean that page must be re-charged each time someone touches it?
>>>>
>>>>
>>> What I meant is that to leave them charged, and if when they are

>>> ummapped and mapped later, charge it to the appropriate BC.
>>>
>>>
>> In this case multithreaded apache that tries to serve each domain in
>> separate BC will fill the memory with BC-s, held by pages allocated
>> and mapped in threads.
>>
>
> I do not understand how the memory will be filled with BCs. Can you
> explain, please.
>
Sure. At the beginning I have one task with one BC. Then
1. A thread is spawned and new BC is created;
2. New thread touches a new page (e.g. maps a new file) which is charged
to new BC
   (and this means that this BC's must stay in memory till page is
uncharged);
3. Thread exits after serving the request, but since it's mm is shared
with parent
   all the touched pages stay resident and, thus, the new BC is still
pinned in memory.
Steps 1-3 are done multiple times for new pages (new files).
Remember that we're discussing the case when pages are not recharged.

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Balbir Singh on Mon, 11 Sep 2006 07:54:06 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Balbir Singh wrote:
>> Dave Hansen wrote:
>>> On Fri, 2006-09-08 at 11:33 +0400, Pavel Emelianov wrote:
>>>> I'm afraid we have different understandings of what a "guarantee" is.
>>> It appears so.
>>>
>>>> Don't we?
>>>> Guarantee may be one of
>>>>
>>>> 1. container will be able to touch that number of pages
>>>> 2. container will be able to sys_mmap() that number of pages
>>>> 3. container will not be killed unless it touches that number of
>>>> pages
>>> A "death sentence" guarantee? I like it. :)
>>>
>>>> 4. anything else
>>>>

>>>> Let's decide what kind of a guarantee we want.
>> I think of guarantees w.r.t resources as the lower limit on the resource.
>> Guarantees and limits can be thought of as the range (guarantee, limit]
>> for the usage of the resource.
>>
>>> I think of it as: "I will be allowed to use this many total pages, and
>>> they are guaranteed not to fail." (1), I think. The sum of all of the
>>> system's guarantees must be less than or equal to the amount of free
>>> memory on the machine.
>> Yes, totally agree.
>
> Such a guarantee is really a limit and this limit is even harder than
> BC's one :)
>
> E.g. I have a node with 1Gb of ram and 10 containers with 100Mb
> guarantee each.
> I want to start one more. What shall I do not to break guarantees?

Don't start the new container or change the guarantees of the existing ones to accommodate this one :) The QoS design (done by the administrator) should take care of such use-cases. It would be perfectly ok to have a container that does not care about guarantees to set their guarantee to 0 and set their limit to the desired value. As Chandra has been stating we need two parameters (guarantee, limit), either can be optional, but not both.

>
>>> If we knew to which NUMA node the memory was going to go, we might as
>>> well take the pages out of the allocator.
>>>
>>> -- Dave
>>>

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Mon, 11 Sep 2006 08:13:59 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelianov wrote:

>> Balbir Singh wrote:
>>> Dave Hansen wrote:
>>>> On Fri, 2006-09-08 at 11:33 +0400, Pavel Emelianov wrote:
>>>>> I'm afraid we have different understandings of what a "guarantee" is.
>>>> It appears so.
>>>>
>>>>> Don't we?
>>>>> Guarantee may be one of
>>>>>
>>>>> 1. container will be able to touch that number of pages
>>>>> 2. container will be able to sys_mmap() that number of pages
>>>>> 3. container will not be killed unless it touches that number of
>>>>> pages
>>>> A "death sentence" guarantee? I like it. :)
>>>>
>>>>> 4. anything else
>>>>>
>>>>> Let's decide what kind of a guarantee we want.
>>> I think of guarantees w.r.t resources as the lower limit on the
>>> resource.
>>> Guarantees and limits can be thought of as the range (guarantee, limit]
>>> for the usage of the resource.
>>>
>>>> I think of it as: "I will be allowed to use this many total pages, and
>>>> they are guaranteed not to fail." (1), I think. The sum of all of
>>>> the
>>>> system's guarantees must be less than or equal to the amount of free
>>>> memory on the machine.
>>> Yes, totally agree.
>>
>> Such a guarantee is really a limit and this limit is even harder than
>> BC's one :)
>>
>> E.g. I have a node with 1Gb of ram and 10 containers with 100Mb
>> guarantee each.
>> I want to start one more. What shall I do not to break guarantees?
>
> Don't start the new container or change the guarantees of the existing
> ones
> to accommodate this one :) The QoS design (done by the administrator)
> should
> take care of such use-cases. It would be perfectly ok to have a container
> that does not care about guarantees to set their guarantee to 0 and set
> their limit to the desired value. As Chandra has been stating we need two
> parameters (guarantee, limit), either can be optional, but not both.
If I set up 9 groups to have 100Mb limit then I have 100Mb assured (on
1Gb node)
for the 10th one exactly. And I do not have to set up any guarantee as

it won't affect
anything. So what a guarantee parameter is needed for?

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Balbir Singh on Mon, 11 Sep 2006 08:19:22 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Balbir Singh wrote:
>> Pavel Emelianov wrote:
>>> Balbir Singh wrote:
>>>> Dave Hansen wrote:
>>>>> On Fri, 2006-09-08 at 11:33 +0400, Pavel Emelianov wrote:
>>>>>> I'm afraid we have different understandings of what a "guarantee" is.
>>>>> It appears so.
>>>>>
>>>>>> Don't we?
>>>>>> Guarantee may be one of
>>>>>>
>>>>>> 1. container will be able to touch that number of pages
>>>>>> 2. container will be able to sys_mmap() that number of pages
>>>>>> 3. container will not be killed unless it touches that number of
>>>>>> pages
>>>>> A "death sentence" guarantee? I like it. :)
>>>>>
>>>>>> 4. anything else
>>>>>>
>>>>>> Let's decide what kind of a guarantee we want.
>>>> I think of guarantees w.r.t resources as the lower limit on the
>>>> resource.
>>>> Guarantees and limits can be thought of as the range (guarantee, limit]
>>>> for the usage of the resource.
>>>>
>>> I think of it as: "I will be allowed to use this many total pages, and
>>> they are guaranteed not to fail." (1), I think. The sum of all of
>>> the
>>> system's guarantees must be less than or equal to the amount of free
>>> memory on the machine.
>> Yes, totally agree.
>>> Such a guarantee is really a limit and this limit is even harder than
>>> BC's one :)
>>>
>>> E.g. I have a node with 1Gb of ram and 10 containers with 100Mb
>>> guarantee each.
>>> I want to start one more. What shall I do not to break guarantees?
>> Don't start the new container or change the guarantees of the existing

>> ones
>> to accommodate this one :) The QoS design (done by the administrator)
>> should
>> take care of such use-cases. It would be perfectly ok to have a container
>> that does not care about guarantees to set their guarantee to 0 and set
>> their limit to the desired value. As Chandra has been stating we need two
>> parameters (guarantee, limit), either can be optional, but not both.
> If I set up 9 groups to have 100Mb limit then I have 100Mb assured (on
> 1Gb node)
> for the 10th one exactly. And I do not have to set up any guarantee as
> it won't affect
> anything. So what a guarantee parameter is needed for?

This use case works well for providing guarantee to one container. What if
I want guarantees of 100Mb and 200Mb for two containers? How do I setup
the system using limits?

Even I restrict everyone else to 700Mb. With this I cannot be sure that
the remaining 300Mb will be distributed as 100Mb and 200Mb.

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Srivatsa Vaddagiri on Mon, 11 Sep 2006 10:21:52 GMT
View Forum Message <> Reply to Message

On Mon, Sep 11, 2006 at 12:13:59PM +0400, Pavel Emelianov wrote:
> If I set up 9 groups to have 100Mb limit then I have 100Mb assured (on
> 1Gb node)
> for the 10th one exactly. And I do not have to set up any guarantee as
> it won't affect
> anything. So what a guarantee parameter is needed for?

I presume you are talking of hard-limiting each group to 100 MB here. In
which case, wont the 100MB (reserved for 10th group) be unutilized
untill 10th group is started (it may never be started for that matter!).

IMO it would be better to go and use that free 100 MB for reclaimable memory
and give that up when 10th group is started.

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Srivatsa Vaddagiri on Mon, 11 Sep 2006 13:04:28 GMT
View Forum Message <> Reply to Message

On Mon, Sep 11, 2006 at 11:02:06AM +0400, Pavel Emelianov wrote:
> Sure. At the beginning I have one task with one BC. Then
> 1. A thread is spawned and new BC is created;

Why do we have to create a BC for every new thread? A new BC is needed
for every new service level instead IMO. And typically there wont be
unlimited service levels.

> 2. New thread touches a new page (e.g. maps a new file) which is charged
> to new BC
> (and this means that this BC's must stay in memory till page is
> uncharged);
> 3. Thread exits after serving the request, but since it's mm is shared
> with parent
> all the touched pages stay resident and, thus, the new BC is still
> pinned in memory.
> Steps 1-3 are done multiple times for new pages (new files).
> Remember that we're discussing the case when pages are not recharged.

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Mon, 11 Sep 2006 18:25:07 GMT
View Forum Message <> Reply to Message

On Fri, 2006-09-08 at 14:43 -0700, Rohit Seth wrote:
<snip>

> > > Guarantee may be one of
> > >
> > > 1. container will be able to touch that number of pages
> > > 2. container will be able to sys_mmap() that number of pages

> > > 3. container will not be killed unless it touches that number of pages
> > > 4. anything else
> >
> > I would say (1) with slight modification
> > "container will be able to touch _at least_ that number of pages"
> >
>
> Does this scheme support running of tasks outside of containers on the
> same platform where you have tasks running inside containers. If so
> then how will you ensure processes running out side any container will
> not leave less than the total guaranteed memory to different containers.
>

There could be a default container which doesn't have any guarantee or limit. When you create containers and assign guarantees to each of them make sure that you leave some amount of resource unassigned. That unassigned resources can be used by the default container or can be used by containers that want more than their guarantee (and less than their limit). This is how CKRM/RG handles this issue.

>
>
> -rohit
>
>
> -- -------------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

 -- ----------
   Chandra Seetharaman | Be careful what you choose....
           - sekharan@us.ibm.com | .....you may get it.
 -- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Mon, 11 Sep 2006 18:44:22 GMT
View Forum Message <> Reply to Message

On Mon, 2006-09-11 at 10:56 +0400, Pavel Emelianov wrote:

<snip>

> >> I think of it as: "I will be allowed to use this many total pages, and
> >> they are guaranteed not to fail." (1), I think. The sum of all of the
> >> system's guarantees must be less than or equal to the amount of free
> >> memory on the machine.
> >
> > Yes, totally agree.
>
> Such a guarantee is really a limit and this limit is even harder than
> BC's one :)
>
> E.g. I have a node with 1Gb of ram and 10 containers with 100Mb
> guarantee each.

In the first place system administrator should not be configuring it
that way, Then they are using it as a strict hard limit than guarantee
(as the resources guaranteed to one container is _not_ available to
others).

Besides, the above configuration is clearly _not_ work conservative.

They should use both guarantee and limit to associate resources to a
container/RG.

> I want to start one more. What shall I do not to break guarantees?

CKRM/RG handles it this way:

Amount of a resource a child RG gets is the ratio of its share value to
the parent's total # of shares. Children's resource allocation can be
changed just by changing the parent's total # of shares.

If you case about initial situation would be:
  Total memory in the system 100MB
  parent's total # of shares: 100 (1 share == 1MB)
  10 children with # of shares: 10 (i.e each children has 10MB)

When I want to add another child, just change parent's total # of shares
to be say 125:
  Total memory in the system 100MB
  parent's total # of shares: 125 (1 share == 0.8MB)
  10 children with # of shares: 10 (i.e each children has 8MB)
Now you are left with 25 shares (or 20MB) that you can assign to new
child(ren) as you please.

<snip>

--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Mon, 11 Sep 2006 18:47:09 GMT
View Forum Message <> Reply to Message

On Mon, 2006-09-11 at 11:02 +0400, Pavel Emelianov wrote:
<snip>

> >> In this case multithreaded apache that tries to serve each domain in
> >> separate BC will fill the memory with BC-s, held by pages allocated
> >> and mapped in threads.
> >>
> >
> > I do not understand how the memory will be filled with BCs. Can you
> > explain, please.
> >
> Sure. At the beginning I have one task with one BC. Then
> 1. A thread is spawned and new BC is created;

You do not have to create a new BC for each new thread, just associate
the thread to an existing appropriate BC.

> 2. New thread touches a new page (e.g. maps a new file) which is charged
> to new BC
> (and this means that this BC's must stay in memory till page is
> uncharged);
> 3. Thread exits after serving the request, but since it's mm is shared
> with parent
> all the touched pages stay resident and, thus, the new BC is still
> pinned in memory.
> Steps 1-3 are done multiple times for new pages (new files).
> Remember that we're discussing the case when pages are not recharged.
>

> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Mon, 11 Sep 2006 18:49:50 GMT
View Forum Message <> Reply to Message

On Mon, 2006-09-11 at 12:13 +0400, Pavel Emelianov wrote:

<snip>
> >
> > Don't start the new container or change the guarantees of the existing
> > ones
> > to accommodate this one :) The QoS design (done by the administrator)
> > should
> > take care of such use-cases. It would be perfectly ok to have a container
> > that does not care about guarantees to set their guarantee to 0 and set
> > their limit to the desired value. As Chandra has been stating we need two
> > parameters (guarantee, limit), either can be optional, but not both.
> If I set up 9 groups to have 100Mb limit then I have 100Mb assured (on
> 1Gb node)
> for the 10th one exactly. And I do not have to set up any guarantee as
> it won't affect
> anything. So what a guarantee parameter is needed for?

I do not think it is that simple since
- there is typically more than one class I want to set guarantee to
- I will not able to use both limit and guarantee
- Implementation will not be work-conserving.

Also, How would you configure the following in your model ?

5 classes: Class A(10, 40), Class B(20, 100), Class C (30, 100), Class D
(5, 100), Class E(15, 50); (class_name(guarantee, limit))

"Limit only" approach works for DoS prevention. But for providing QoS
you would need guarantee.
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....

- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Rohit Seth on Mon, 11 Sep 2006 19:10:31 GMT
View Forum Message <> Reply to Message

On Mon, 2006-09-11 at 11:25 -0700, Chandra Seetharaman wrote:
> On Fri, 2006-09-08 at 14:43 -0700, Rohit Seth wrote:
> <snip>
>
> > > > Guarantee may be one of
> > > >
> > > > 1. container will be able to touch that number of pages
> > > > 2. container will be able to sys_mmap() that number of pages
> > > > 3. container will not be killed unless it touches that number of pages
> > > > 4. anything else
> > >
> > > I would say (1) with slight modification
> > > "container will be able to touch _at least_ that number of pages"
> > >
> >
> > Does this scheme support running of tasks outside of containers on the
> > same platform where you have tasks running inside containers. If so
> > then how will you ensure processes running out side any container will
> > not leave less than the total guaranteed memory to different containers.
> >
>
> There could be a default container which doesn't have any guarantee or
> limit.

First, I think it is critical that we allow processes to run outside of
any container (unless we know for sure that the penalty of running a
process inside a container is very very minimal).

And anything running outside a container should be limited by default
Linux settings.

> When you create containers and assign guarantees to each of them
> make sure that you leave some amount of resource unassigned.
^^^^^ This will force the "default" container
with limits (indirectly). IMO, the whole guarantee feature gets defeated
the moment you bring in this fuzziness.

> That
> unassigned resources can be used by the default container or can be used

> by containers that want more than their guarantee (and less than their
> limit). This is how CKRM/RG handles this issue.
>
>

It seems that a single notion of limit should suffice, and that limit
should more be treated as something beyond which that resource
consumption in the container will be throttled/not_allowed.

-rohit

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4)
(added user memory)
Posted by Chandra Seetharaman on Mon, 11 Sep 2006 19:42:05 GMT
View Forum Message <> Reply to Message

On Mon, 2006-09-11 at 12:10 -0700, Rohit Seth wrote:
> On Mon, 2006-09-11 at 11:25 -0700, Chandra Seetharaman wrote:
> > On Fri, 2006-09-08 at 14:43 -0700, Rohit Seth wrote:
> > <snip>
> >
> > > > > > Guarantee may be one of
> > > > > >
> > > > > > 1. container will be able to touch that number of pages
> > > > > > 2. container will be able to sys_mmap() that number of pages
> > > > > > 3. container will not be killed unless it touches that number of pages
> > > > > > 4. anything else
> > > > >
> > > > > I would say (1) with slight modification
> > > > > "container will be able to touch _at least_ that number of pages"
> > > >
> > >
> > > Does this scheme support running of tasks outside of containers on the
> > > same platform where you have tasks running inside containers. If so
> > > then how will you ensure processes running out side any container will
> > > not leave less than the total guaranteed memory to different containers.
> > >
> >
> > There could be a default container which doesn't have any guarantee or
> > limit.
>
> First, I think it is critical that we allow processes to run outside of
> any container (unless we know for sure that the penalty of running a
> process inside a container is very very minimal).

When I meant a default container I meant a default "resource group". In
case of container that would be the default environment. I do not see

any additional overhead associated with it, it is only associated with
how resource are allocated/accounted.

>
> And anything running outside a container should be limited by default
> Linux settings.

note that the resource available to the default RG will be (total system
resource - allocated to RGs).
>
> > When you create containers and assign guarantees to each of them
> > make sure that you leave some amount of resource unassigned.
>   ^^^^^ This will force the "default" container
> with limits (indirectly). IMO, the whole guarantee feature gets defeated

You _will_ have limits for the default RG even if we don't have
guarantees.

> the moment you bring in this fuzziness.

Not really.
 - Each RG will have a guarantee and limit of each resource.
 - default RG will have (system resource - sum of guarantees)
 - Every RG will be guaranteed some amount of resource to provide QoS
 - Every RG will be limited at "limit" to prevent DoS attacks.
 - Whoever doesn't care either of those set them to don't care values.

>
> > That
> > unassigned resources can be used by the default container or can be used
> > by containers that want more than their guarantee (and less than their
> > limit). This is how CKRM/RG handles this issue.
> >
> >
>
> It seems that a single notion of limit should suffice, and that limit
> should more be treated as something beyond which that resource
> consumption in the container will be throttled/not_allowed.

As I stated in an earlier email "Limit only" approach can prevent a
system from DoS attacks (and also fits the container model nicely),
whereas to provide QoS one would need guarantee.

Without guarantee, a RG that the admin cares about can starve if
all/most of the other RGs consume upto their limits.

>
> -rohit

>
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by kir on Mon, 11 Sep 2006 19:47:24 GMT
View Forum Message <> Reply to Message

Rohit Seth wrote:
> On Mon, 2006-09-11 at 11:25 -0700, Chandra Seetharaman wrote:
>
>> On Fri, 2006-09-08 at 14:43 -0700, Rohit Seth wrote:
>> <snip>
>>
>>
>>>>>> Guarantee may be one of
>>>>>>
>>>>>> 1. container will be able to touch that number of pages
>>>>>> 2. container will be able to sys_mmap() that number of pages
>>>>>> 3. container will not be killed unless it touches that number of pages
>>>>>> 4. anything else
>>>>>>
>>>> I would say (1) with slight modification
>>>> "container will be able to touch _at least_ that number of pages"
>>>>
>>>>
>>> Does this scheme support running of tasks outside of containers on the
>>> same platform where you have tasks running inside containers. If so
>>> then how will you ensure processes running out side any container will
>>> not leave less than the total guaranteed memory to different containers.
>>>
>>>
>> There could be a default container which doesn't have any guarantee or

>> limit.
>>
>
> First, I think it is critical that we allow processes to run outside of
> any container (unless we know for sure that the penalty of running a
> process inside a container is very very minimal).
>
(1) there is a set of processes running outside of any container. In
OpenVZ we call that "VE0" or "host system", probably Chandra meant that
by "default container".
(2) The host system is used to manage the containers (start/stop/set
parameters/create/destroy).
(3) the penalty of running a process inside a container is indeed very low.

> And anything running outside a container should be limited by default
> Linux settings.
>
(4) due to (2), it is not recommended to run anything but the tasks used
to manage the containers -- otherwise your gonna have security problems
(5) "Default Linux settings" do not cover everything (for example --
dentry cache), thus the need for beancounters.
>> When you create containers and assign guarantees to each of them
>> make sure that you leave some amount of resource unassigned.
>>
>   ^^^^^ This will force the "default" container
> with limits (indirectly). IMO, the whole guarantee feature gets defeated
> the moment you bring in this fuzziness.
>
>
>> That
>> unassigned resources can be used by the default container or can be used
>> by containers that want more than their guarantee (and less than their
>> limit). This is how CKRM/RG handles this issue.
>>
>>
>>
>
> It seems that a single notion of limit should suffice, and that limit
> should more be treated as something beyond which that resource
> consumption in the container will be throttled/not_allowed.
>
Beancounters have a notion of "barrier" and "limit". For some parameters
they are the same, but for some parameters they differ -- and there is
some "safety gap" between the barrier and the limit. The problem is for
some types of resources you can not throttle or deny -- the only way is
to kill the process. The one (but not the only one) example is process
stack expansion. See more at http://wiki.openvz.org/UBC (and follow the
menu at the right side).

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Rohit Seth on Mon, 11 Sep 2006 23:58:19 GMT
View Forum Message <> Reply to Message

On Mon, 2006-09-11 at 12:42 -0700, Chandra Seetharaman wrote:
> On Mon, 2006-09-11 at 12:10 -0700, Rohit Seth wrote:
> > On Mon, 2006-09-11 at 11:25 -0700, Chandra Seetharaman wrote:

> > > There could be a default container which doesn't have any guarantee or
> > > limit.
> >
> > First, I think it is critical that we allow processes to run outside of
> > any container (unless we know for sure that the penalty of running a
> > process inside a container is very very minimal).
>
> When I meant a default container I meant a default "resource group". In
> case of container that would be the default environment. I do not see
> any additional overhead associated with it, it is only associated with
> how resource are allocated/accounted.
>

There should be some cost when you do atomic inc/dec accounting and
locks for add/remove resources from any container (including default
resource group). No?

> >
> > And anything running outside a container should be limited by default
> > Linux settings.
>
> note that the resource available to the default RG will be (total system
> resource - allocated to RGs).

I think it will be preferable to not change the existing behavior for
applications that are running outside any container (in your case
default resource group).

> >
> > > When you create containers and assign guarantees to each of them
> > > make sure that you leave some amount of resource unassigned.
> >   ^^^^^ This will force the "default" container
> > with limits (indirectly). IMO, the whole guarantee feature gets defeated
>
> You _will_ have limits for the default RG even if we don't have
> guarantees.
>
> > the moment you bring in this fuzziness.
>
> Not really.

> - Each RG will have a guarantee and limit of each resource.
> - default RG will have (system resource - sum of guarantees)
> - Every RG will be guaranteed some amount of resource to provide QoS
> - Every RG will be limited at "limit" to prevent DoS attacks.
> - Whoever doesn't care either of those set them to don't care values.
>

For the cases that put this don't care, do you depend on existing
reclaim algorithm (for memory) in kernel?

> >
> > > That
> > > unassigned resources can be used by the default container or can be used
> > > by containers that want more than their guarantee (and less than their
> > > limit). This is how CKRM/RG handles this issue.
> > >
> > >
> > >
> > It seems that a single notion of limit should suffice, and that limit
> > should more be treated as something beyond which that resource
> > consumption in the container will be throttled/not_allowed.
>
> As I stated in an earlier email "Limit only" approach can prevent a
> system from DoS attacks (and also fits the container model nicely),
> whereas to provide QoS one would need guarantee.
>
> Without guarantee, a RG that the admin cares about can starve if
> all/most of the other RGs consume upto their limits.
>
> >

If the limits are set appropriately so that containers total memory
consumption does not exceed the system memory then there shouldn't be
any QoS issue (to whatever extent it is applicable for specific
scenario).

-rohit

Subject: Re: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Rohit Seth on Tue, 12 Sep 2006 00:28:58 GMT
View Forum Message <> Reply to Message

On Mon, 2006-09-11 at 23:48 +0400, Kir Kolyshkin wrote:
> Rohit Seth wrote:
> > On Mon, 2006-09-11 at 11:25 -0700, Chandra Seetharaman wrote:
> >

> >> On Fri, 2006-09-08 at 14:43 -0700, Rohit Seth wrote:
> >> <snip>
> >>
> >>
> >>>>> Guarantee may be one of
> >>>>>
> >>>>> 1. container will be able to touch that number of pages
> >>>>> 2. container will be able to sys_mmap() that number of pages
> >>>>> 3. container will not be killed unless it touches that number of pages
> >>>>> 4. anything else
> >>>>>
> >>>> I would say (1) with slight modification
> >>>> "container will be able to touch _at least_ that number of pages"
> >>>>
> >>>>
> >>> Does this scheme support running of tasks outside of containers on the
> >>> same platform where you have tasks running inside containers. If so
> >>> then how will you ensure processes running out side any container will
> >>> not leave less than the total guaranteed memory to different containers.
> >>>
> >>>
> >> There could be a default container which doesn't have any guarantee or
> >> limit.
> >>
> >
> > First, I think it is critical that we allow processes to run outside of
> > any container (unless we know for sure that the penalty of running a
> > process inside a container is very very minimal).
> >
> (1) there is a set of processes running outside of any container. In
> OpenVZ we call that "VE0" or "host system", probably Chandra meant that
> by "default container".
> (2) The host system is used to manage the containers (start/stop/set
> parameters/create/destroy).
> (3) the penalty of running a process inside a container is indeed very low.
>
> > And anything running outside a container should be limited by default
> > Linux settings.
> >
> (4) due to (2), it is not recommended to run anything but the tasks used
> to manage the containers -- otherwise your gonna have security problems

Just like you want to run those special threads outside of any
container, some sysadmin might be interested in running different
processes that they don't want to bind to any container limits.

I think it is critical that you provide the capability to have tasks
running outside any container. Whether sysadmin wants to do it or not

for a system is a different thing.

-rohit

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Balbir Singh on Tue, 12 Sep 2006 09:53:38 GMT
View Forum Message <> Reply to Message

Rohit Seth wrote:

> If the limits are set appropriately so that containers total memory
> consumption does not exceed the system memory then there shouldn't be
> any QoS issue (to whatever extent it is applicable for specific
> scenario).
>
> -rohit
>

What if the guarantee and limits are subject to change? Consider many groups,
with changing limits - how do we provide guarantees then?

Limit is the upper bound on resource utilization and guarantee is the lower
bound. In a dynamic system, how can we provide a lower bound on a resource
for a group by manipulating the upper bounds on the rest of the groups?

Consider a system with 1GB of ram and two groups such that they need a guarantee
of 100MB and 200MB of memory. How would you setup limits to ensure that
the guarantees are met? The remaining groups will be limited to 700MB, but
how do we ensure that these classes get 100MB and 200MB of the remaining 300MB
respectively?

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Tue, 12 Sep 2006 10:24:25 GMT
View Forum Message <> Reply to Message

Srivatsa Vaddagiri wrote:
> On Mon, Sep 11, 2006 at 11:02:06AM +0400, Pavel Emelianov wrote:

>
>> Sure. At the beginning I have one task with one BC. Then
>> 1. A thread is spawned and new BC is created;
>>
>
> Why do we have to create a BC for every new thread? A new BC is needed
> for every new service level instead IMO. And typically there wont be
> unlimited service levels.
>
That's the scenario we started from - each domain is served in a separate
BC with *threaded* Apache.
>
>> 2. New thread touches a new page (e.g. maps a new file) which is charged
>> to new BC
>> (and this means that this BC's must stay in memory till page is
>> uncharged);
>> 3. Thread exits after serving the request, but since it's mm is shared
>> with parent
>> all the touched pages stay resident and, thus, the new BC is still
>> pinned in memory.
>> Steps 1-3 are done multiple times for new pages (new files).
>> Remember that we're discussing the case when pages are not recharged.
>>
>
>
>

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Srivatsa Vaddagiri on Tue, 12 Sep 2006 10:29:43 GMT
View Forum Message <> Reply to Message

On Tue, Sep 12, 2006 at 02:24:25PM +0400, Pavel Emelianov wrote:
> Srivatsa Vaddagiri wrote:
> > On Mon, Sep 11, 2006 at 11:02:06AM +0400, Pavel Emelianov wrote:
> >
> >> Sure. At the beginning I have one task with one BC. Then
> >> 1. A thread is spawned and new BC is created;
> >>
> >
> > Why do we have to create a BC for every new thread? A new BC is needed
> > for every new service level instead IMO. And typically there wont be
> > unlimited service levels.
> >
> That's the scenario we started from - each domain is served in a separate
> BC with *threaded* Apache.

Sure ..but you can still meet that requirement by creating fixed set of
BCs (for each domain) and let each new thread be associated with a
corresponding BC (w/o requiring to create BC for every new thread),
depending on which domain's request it is serving?

> >
> >> 2. New thread touches a new page (e.g. maps a new file) which is charged
> >> to new BC
> >> (and this means that this BC's must stay in memory till page is
> >> uncharged);
> >> 3. Thread exits after serving the request, but since it's mm is shared
> >> with parent
> >> all the touched pages stay resident and, thus, the new BC is still
> >> pinned in memory.
> >> Steps 1-3 are done multiple times for new pages (new files).
> >> Remember that we're discussing the case when pages are not recharged.
> >>
> >
> >
> >
>
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Tue, 12 Sep 2006 10:35:45 GMT
View Forum Message <> Reply to Message

Srivatsa Vaddagiri wrote:
> On Mon, Sep 11, 2006 at 12:13:59PM +0400, Pavel Emelianov wrote:
>
>> If I set up 9 groups to have 100Mb limit then I have 100Mb assured (on
>> 1Gb node)
>> for the 10th one exactly. And I do not have to set up any guarantee as
>> it won't affect

>> anything. So what a guarantee parameter is needed for?
>>
>
> I presume you are talking of hard-limiting each group to 100 MB here. In
> which case, wont the 100MB (reserved for 10th group) be unutilized
> untill 10th group is started (it may never be started for that matter!).
>
> IMO it would be better to go and use that free 100 MB for reclaimable memory
> and give that up when 10th group is started.
>
Sure. I've talked about the unreclaimable memory.
Sorry, for not specifying it explicitly.

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Tue, 12 Sep 2006 10:39:56 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelianov wrote:
>> Balbir Singh wrote:
>>> Pavel Emelianov wrote:
>>>> Balbir Singh wrote:
>>>>> Dave Hansen wrote:
>>>>>> On Fri, 2006-09-08 at 11:33 +0400, Pavel Emelianov wrote:
>>>>>>> I'm afraid we have different understandings of what a
>>>>>>> "guarantee" is.
>>>>>> It appears so.
>>>>>>
>>>>>>> Don't we?
>>>>>>> Guarantee may be one of
>>>>>>>
>>>>>>> 1. container will be able to touch that number of pages
>>>>>>> 2. container will be able to sys_mmap() that number of pages
>>>>>>> 3. container will not be killed unless it touches that number of
>>>>>>> pages
>>>>>> A "death sentence" guarantee? I like it. :)
>>>>>>
>>>>>>> 4. anything else
>>>>>>>
>>>>>>> Let's decide what kind of a guarantee we want.
>>>>> I think of guarantees w.r.t resources as the lower limit on the
>>>>> resource.
>>>>> Guarantees and limits can be thought of as the range (guarantee,
>>>>> limit]
>>>>> for the usage of the resource.
>>>>>

>>>>>> I think of it as: "I will be allowed to use this many total
>>>>>> pages, and
>>>>>> they are guaranteed not to fail." (1), I think. The sum of all of
>>>>>> the
>>>>>> system's guarantees must be less than or equal to the amount of free
>>>>>> memory on the machine.
>>>>> Yes, totally agree.
>>>> Such a guarantee is really a limit and this limit is even harder than
>>>> BC's one :)
>>>>
>>>> E.g. I have a node with 1Gb of ram and 10 containers with 100Mb
>>>> guarantee each.
>>>> I want to start one more. What shall I do not to break guarantees?
>>> Don't start the new container or change the guarantees of the existing
>>> ones
>>> to accommodate this one :) The QoS design (done by the administrator)
>>> should
>>> take care of such use-cases. It would be perfectly ok to have a
>>> container
>>> that does not care about guarantees to set their guarantee to 0 and set
>>> their limit to the desired value. As Chandra has been stating we
>>> need two
>>> parameters (guarantee, limit), either can be optional, but not both.
>> If I set up 9 groups to have 100Mb limit then I have 100Mb assured (on
>> 1Gb node)
>> for the 10th one exactly. And I do not have to set up any guarantee as
>> it won't affect
>> anything. So what a guarantee parameter is needed for?
>
> This use case works well for providing guarantee to one container.
> What if
> I want guarantees of 100Mb and 200Mb for two containers? How do I setup
> the system using limits?
You may set any value from 100 up to 800 Mb for the first one and
200-900Mb for
the second. In case of no other groups first will receive its 100Mb for
sure and
so does the second. If there are other groups - their guarantees should
be concerned.
>
> Even I restrict everyone else to 700Mb. With this I cannot be sure that
> the remaining 300Mb will be distributed as 100Mb and 200Mb.
There's no "everyone else" here - we're talking about a "static" case.
When new group arrives we need to recalculate guarantees as you said.
And here's my next question - what to do if the new guarantee would become
lower that current amount of unreclaimable memory in BC?

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Tue, 12 Sep 2006 10:40:03 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelianov wrote:
>> Balbir Singh wrote:
>>> Pavel Emelianov wrote:
>>>> Balbir Singh wrote:
>>>>> Dave Hansen wrote:
>>>>>> On Fri, 2006-09-08 at 11:33 +0400, Pavel Emelianov wrote:
>>>>>>> I'm afraid we have different understandings of what a
>>>>>>> "guarantee" is.
>>>>>> It appears so.
>>>>>>
>>>>>>> Don't we?
>>>>>>> Guarantee may be one of
>>>>>>>
>>>>>>> 1. container will be able to touch that number of pages
>>>>>>> 2. container will be able to sys_mmap() that number of pages
>>>>>>> 3. container will not be killed unless it touches that number of
>>>>>>> pages
>>>>>> A "death sentence" guarantee? I like it. :)
>>>>>>
>>>>>>> 4. anything else
>>>>>>>
>>>>>>> Let's decide what kind of a guarantee we want.
>>>>> I think of guarantees w.r.t resources as the lower limit on the
>>>>> resource.
>>>>> Guarantees and limits can be thought of as the range (guarantee,
>>>>> limit]
>>>>> for the usage of the resource.
>>>>>
>>>> I think of it as: "I will be allowed to use this many total
>>>> pages, and
>>>> they are guaranteed not to fail." (1), I think. The sum of all of
>>>> the
>>>> system's guarantees must be less than or equal to the amount of free
>>>> memory on the machine.
>>> Yes, totally agree.
>>>> Such a guarantee is really a limit and this limit is even harder than
>>>> BC's one :)
>>>>
>>>> E.g. I have a node with 1Gb of ram and 10 containers with 100Mb
>>>> guarantee each.
>>>> I want to start one more. What shall I do not to break guarantees?
>>> Don't start the new container or change the guarantees of the existing
>>> ones

>>> to accommodate this one :) The QoS design (done by the administrator)
>>> should
>>> take care of such use-cases. It would be perfectly ok to have a
>>> container
>>> that does not care about guarantees to set their guarantee to 0 and set
>>> their limit to the desired value. As Chandra has been stating we
>>> need two
>>> parameters (guarantee, limit), either can be optional, but not both.
>> If I set up 9 groups to have 100Mb limit then I have 100Mb assured (on
>> 1Gb node)
>> for the 10th one exactly. And I do not have to set up any guarantee as
>> it won't affect
>> anything. So what a guarantee parameter is needed for?
>
> This use case works well for providing guarantee to one container.
> What if
> I want guarantees of 100Mb and 200Mb for two containers? How do I setup
> the system using limits?
You may set any value from 100 up to 800 Mb for the first one and
200-900Mb for
the second. In case of no other groups first will receive its 100Mb for
sure and
so does the second. If there are other groups - their guarantees should
be concerned.
>
> Even I restrict everyone else to 700Mb. With this I cannot be sure that
> the remaining 300Mb will be distributed as 100Mb and 200Mb.
There's no "everyone else" here - we're talking about a "static" case.
When new group arrives we need to recalculate guarantees as you said.
And here's my next question - what to do if the new guarantee would become
lower that current amount of unreclaimable memory in BC?

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Srivatsa Vaddagiri on Tue, 12 Sep 2006 10:44:10 GMT
View Forum Message <> Reply to Message

On Mon, Sep 11, 2006 at 12:10:31PM -0700, Rohit Seth wrote:
> It seems that a single notion of limit should suffice, and that limit
> should more be treated as something beyond which that resource
> consumption in the container will be throttled/not_allowed.

The big question is : are containers/RG allowed to use *upto* their
limit always? In other words, will you typically setup limits such that
sum of all limits = max resource capacity?

If it is setup like that, then what you are considering as limit is

actually guar no?

If it wont be setup like that, then I dont see how one can provide QoS.

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Tue, 12 Sep 2006 10:48:18 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:
> On Mon, 2006-09-11 at 12:13 +0400, Pavel Emelianov wrote:
>
> <snip>
>
>>> Don't start the new container or change the guarantees of the existing
>>> ones
>>> to accommodate this one :) The QoS design (done by the administrator)
>>> should
>>> take care of such use-cases. It would be perfectly ok to have a container
>>> that does not care about guarantees to set their guarantee to 0 and set
>>> their limit to the desired value. As Chandra has been stating we need two
>>> parameters (guarantee, limit), either can be optional, but not both.
>>>
>> If I set up 9 groups to have 100Mb limit then I have 100Mb assured (on
>> 1Gb node)
>> for the 10th one exactly. And I do not have to set up any guarantee as
>> it won't affect
>> anything. So what a guarantee parameter is needed for?
>>
>
> I do not think it is that simple since
> - there is typically more than one class I want to set guarantee to
> - I will not able to use both limit and guarantee
> - Implementation will not be work-conserving.
>
> Also, How would you configure the following in your model ?
>
> 5 classes: Class A(10, 40), Class B(20, 100), Class C (30, 100), Class D
> (5, 100), Class E(15, 50); (class_name(guarantee, limit))
>
What's the total memory amount on the node? Without it it's hard to make
any
guarantee.

> "Limit only" approach works for DoS prevention. But for providing QoS
> you would need guarantee.
>
You may not provide guarantee on physycal resource for a particular group
without limiting its usage by other groups. That's my major idea.

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Tue, 12 Sep 2006 11:06:35 GMT
View Forum Message <> Reply to Message

Srivatsa Vaddagiri wrote:
> On Tue, Sep 12, 2006 at 02:24:25PM +0400, Pavel Emelianov wrote:
>
>> Srivatsa Vaddagiri wrote:
>>
>>> On Mon, Sep 11, 2006 at 11:02:06AM +0400, Pavel Emelianov wrote:
>>>
>>>
>>>> Sure. At the beginning I have one task with one BC. Then
>>>> 1. A thread is spawned and new BC is created;
>>>>
>>>>
>>> Why do we have to create a BC for every new thread? A new BC is needed
>>> for every new service level instead IMO. And typically there wont be
>>> unlimited service levels.
>>>
>>>
>> That's the scenario we started from - each domain is served in a separate
>> BC with *threaded* Apache.
>>
>
> Sure ..but you can still meet that requirement by creating fixed set of
> BCs (for each domain) and let each new thread be associated with a
> corresponding BC (w/o requiring to create BC for every new thread),
> depending on which domain's request it is serving?
>
Hmmm... Beancounters can provide this after trivial changes.
We may schedule them in current set of "pending" features
(http://wiki.openvz.org/UBC_discussion)

But this can create a kind of DoS within an application:
  A thread continuously touches new and new pages to it's BC and
these pages are get touched by other threads also. Sooner or later
this BC will hit it's limit and reclaiming this set of pages would affect
all the other threads.

Also such accounting reveals you NOTHING about real memory usage.
E.g. 100Mb charged for one BC can mean "this BC ate 100Mb of
memory" as well as "this BC uses one page really, but all the others
are just used by other threads" and anything between these two
corner cases.

Well. We've digressed from our main thread - discussing (dis)advantages
of current BC implemenation.
>
>>>
>>>
>>>> 2. New thread touches a new page (e.g. maps a new file) which is charged
>>>> to new BC
>>>> (and this means that this BC's must stay in memory till page is
>>>> uncharged);
>>>> 3. Thread exits after serving the request, but since it's mm is shared
>>>> with parent
>>>> all the touched pages stay resident and, thus, the new BC is still
>>>> pinned in memory.
>>>> Steps 1-3 are done multiple times for new pages (new files).
>>>> Remember that we're discussing the case when pages are not recharged.
>>>>

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Balbir Singh on Tue, 12 Sep 2006 12:01:43 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
<snip>

>>>>> E.g. I have a node with 1Gb of ram and 10 containers with 100Mb
>>>>> guarantee each.
>>>>> I want to start one more. What shall I do not to break guarantees?
>>>> Don't start the new container or change the guarantees of the existing
>>>> ones
>>>> to accommodate this one :) The QoS design (done by the administrator)
>>>> should
>>>> take care of such use-cases. It would be perfectly ok to have a
>>>> container
>>>> that does not care about guarantees to set their guarantee to 0 and set
>>>> their limit to the desired value. As Chandra has been stating we
>>>> need two
>>>> parameters (guarantee, limit), either can be optional, but not both.
>>> If I set up 9 groups to have 100Mb limit then I have 100Mb assured (on
>>> 1Gb node)
>>> for the 10th one exactly. And I do not have to set up any guarantee as

>>> it won't affect
>>> anything. So what a guarantee parameter is needed for?
>> This use case works well for providing guarantee to one container.
>> What if
>> I want guarantees of 100Mb and 200Mb for two containers? How do I setup
>> the system using limits?
> You may set any value from 100 up to 800 Mb for the first one and
> 200-900Mb for
> the second. In case of no other groups first will receive its 100Mb for
> sure and
> so does the second. If there are other groups - their guarantees should
> be concerned.

If I add another group with a guarantee of 100MB, then its limit will
be anywhere between 100MB-800MB ?

I do not understand the guarantees being concerned part.

>> Even I restrict everyone else to 700Mb. With this I cannot be sure that
>> the remaining 300Mb will be distributed as 100Mb and 200Mb.
> There's no "everyone else" here - we're talking about a "static" case.
> When new group arrives we need to recalculate guarantees as you said.

I was speaking in general where we have 'n' groups, so thats why I had
"everyone else".

> And here's my next question - what to do if the new guarantee would become
> lower that current amount of unreclaimable memory in BC?
>

I am not quite sure I understand this question. Let me try with an example.
Lets say I have 1GB of memory and I have guarantees of 100 and 200MB for
two groups. Lets say the total unreclaimable memory is 100MB. If I add a new
group with guarantee of 50MB, which is lower than the total amount of
total unreclaimable memory. The addition of the new group should be work
fine since we have 1GB - 100 - 100 - 200MB of memory available.

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Srivatsa Vaddagiri on Tue, 12 Sep 2006 12:04:07 GMT

On Tue, Sep 12, 2006 at 03:06:35PM +0400, Pavel Emelianov wrote:
> Hmmm... Beancounters can provide this after trivial changes.

All that is needed is some interface to set a thread's BC id (which you
seem to have already - sys_set_bcid)

> We may schedule them in current set of "pending" features
> (http://wiki.openvz.org/UBC_discussion)
>
> But this can create a kind of DoS within an application:
> A thread continuously touches new and new pages to it's BC and
> these pages are get touched by other threads also. Sooner or later

Any good reason why threads will touch each other's working set?
Sure nothing prevents them from touching, but I would expect each thread
(serving a separate domain) to work on its own set of private pages?

> this BC will hit it's limit and reclaiming this set of pages would affect
> all the other threads.

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Rohit Seth on Tue, 12 Sep 2006 17:22:32 GMT

On Tue, 2006-09-12 at 16:14 +0530, Srivatsa Vaddagiri wrote:
> On Mon, Sep 11, 2006 at 12:10:31PM -0700, Rohit Seth wrote:
> > It seems that a single notion of limit should suffice, and that limit
> > should more be treated as something beyond which that resource
> > consumption in the container will be throttled/not_allowed.
>
> The big question is : are containers/RG allowed to use *upto* their
> limit always? In other words, will you typically setup limits such that
> sum of all limits = max resource capacity?
>

If a user is really interested in ensuring that all scheduled jobs (or
containers) get what they have asked for (guarantees) then making the
sum of all container limits equal to total system limit is the right
thing to do.

> If it is setup like that, then what you are considering as limit is
> actually guar no?
>
Right. And if we do it like this then it is up to sysadmin to configure
the thing right without adding additional logic in kernel.

-rohit

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Srivatsa Vaddagiri on Tue, 12 Sep 2006 17:40:58 GMT
View Forum Message <> Reply to Message

On Tue, Sep 12, 2006 at 10:22:32AM -0700, Rohit Seth wrote:
> On Tue, 2006-09-12 at 16:14 +0530, Srivatsa Vaddagiri wrote:
> > On Mon, Sep 11, 2006 at 12:10:31PM -0700, Rohit Seth wrote:
> > > It seems that a single notion of limit should suffice, and that limit
> > > should more be treated as something beyond which that resource
> > > consumption in the container will be throttled/not_allowed.
> >
> > The big question is : are containers/RG allowed to use *upto* their
> > limit always? In other words, will you typically setup limits such that
> > sum of all limits = max resource capacity?
> >
>
> If a user is really interested in ensuring that all scheduled jobs (or
> containers) get what they have asked for (guarantees) then making the
> sum of all container limits equal to total system limit is the right
> thing to do.
>
> > If it is setup like that, then what you are considering as limit is
> > actually guar no?
> >
> Right. And if we do it like this then it is up to sysadmin to configure
> the thing right without adding additional logic in kernel.

Perhaps calling it as "limit" in confusing then (otoh it may go down well
with Linus!). I perhaps agree we need to go with one for now (in the
interest of making some progress), but we probably will come back to
this at a later point. For ex, I chanced upon this document:

 www.vmware.com/pdf/vmware_drs_wp.pdf

which explains how supporting a hard limit (in contrast to guar as we
have been discussing) can be usefull sometimes.

--

Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Tue, 12 Sep 2006 23:54:13 GMT
View Forum Message <> Reply to Message

On Mon, 2006-09-11 at 16:58 -0700, Rohit Seth wrote:
> On Mon, 2006-09-11 at 12:42 -0700, Chandra Seetharaman wrote:
> > On Mon, 2006-09-11 at 12:10 -0700, Rohit Seth wrote:
> > > On Mon, 2006-09-11 at 11:25 -0700, Chandra Seetharaman wrote:
> >
> > > > There could be a default container which doesn't have any guarantee or
> > > > limit.
> > >
> > > First, I think it is critical that we allow processes to run outside of
> > > any container (unless we know for sure that the penalty of running a
> > > process inside a container is very very minimal).
> >
> > When I meant a default container I meant a default "resource group". In
> > case of container that would be the default environment. I do not see
> > any additional overhead associated with it, it is only associated with
> > how resource are allocated/accounted.
> >
>
> There should be some cost when you do atomic inc/dec accounting and
> locks for add/remove resources from any container (including default
> resource group). No?

yes, it would be there, but is not heavy, IMO.

>
> > > And anything running outside a container should be limited by default
> > > Linux settings.
> >
> > note that the resource available to the default RG will be (total system
> > resource - allocated to RGs).
>
> I think it will be preferable to not change the existing behavior for
> applications that are running outside any container (in your case
> default resource group).

hmm, when you provide QoS for a set of apps, you will affect (the resource availability of) other apps. I don't see any way around it. Any ideas ?

>
> > >
> > > > When you create containers and assign guarantees to each of them
> > > > make sure that you leave some amount of resource unassigned.
> > >   ^^^^^ This will force the "default" container
> > > with limits (indirectly). IMO, the whole guarantee feature gets defeated
> >
> > You _will_ have limits for the default RG even if we don't have
> > guarantees.
> >
> > > the moment you bring in this fuzziness.
> >
> > Not really.
> > - Each RG will have a guarantee and limit of each resource.
> > - default RG will have (system resource - sum of guarantees)
> > - Every RG will be guaranteed some amount of resource to provide QoS
> > - Every RG will be limited at "limit" to prevent DoS attacks.
> > - Whoever doesn't care either of those set them to don't care values.
> >
>
> For the cases that put this don't care, do you depend on existing
> reclaim algorithm (for memory) in kernel?

Yes.
>
> > >
> > > > That
> > > > unassigned resources can be used by the default container or can be used
> > > > by containers that want more than their guarantee (and less than their
> > > > limit). This is how CKRM/RG handles this issue.
> > > >
> > > >
> > >
> > > It seems that a single notion of limit should suffice, and that limit
> > > should more be treated as something beyond which that resource
> > > consumption in the container will be throttled/not_allowed.
> >
> > As I stated in an earlier email "Limit only" approach can prevent a
> > system from DoS attacks (and also fits the container model nicely),
> > whereas to provide QoS one would need guarantee.
> >
> > Without guarantee, a RG that the admin cares about can starve if
> > all/most of the other RGs consume upto their limits.
> >
> > >
>
> If the limits are set appropriately so that containers total memory
> consumption does not exceed the system memory then there shouldn't be

> any QoS issue (to whatever extent it is applicable for specific
> scenario).

Then you will not be work-conserving (IOW over-committing), which is one
of the main advantage of this type of feature.

>
> -rohit
>
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Tue, 12 Sep 2006 23:58:08 GMT
View Forum Message <> Reply to Message

On Tue, 2006-09-12 at 14:48 +0400, Pavel Emelianov wrote:
<snip>
> > I do not think it is that simple since
> > - there is typically more than one class I want to set guarantee to
> > - I will not able to use both limit and guarantee
> > - Implementation will not be work-conserving.
> >
> > Also, How would you configure the following in your model ?
> >
> > 5 classes: Class A(10, 40), Class B(20, 100), Class C (30, 100), Class D
> > (5, 100), Class E(15, 50); (class_name(guarantee, limit))
> >
> What's the total memory amount on the node? Without it it's hard to make
> any
> guarantee.

I wrote the example treating them as %, so 100 would be the total amount

of memory.

> > "Limit only" approach works for DoS prevention. But for providing QoS
> > you would need guarantee.
> >
> You may not provide guarantee on physycal resource for a particular group
> without limiting its usage by other groups. That's my major idea.

I agree with that, but the other way around (i.e provide guarantee for
everyone by imposing limits on everyone) is what I am saying is not
possible.
>
>
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4)
(added user memory)
Posted by Chandra Seetharaman on Wed, 13 Sep 2006 00:02:12 GMT
View Forum Message <> Reply to Message

On Tue, 2006-09-12 at 10:22 -0700, Rohit Seth wrote:
> On Tue, 2006-09-12 at 16:14 +0530, Srivatsa Vaddagiri wrote:
> > On Mon, Sep 11, 2006 at 12:10:31PM -0700, Rohit Seth wrote:
> > > It seems that a single notion of limit should suffice, and that limit
> > > should more be treated as something beyond which that resource
> > > consumption in the container will be throttled/not_allowed.
> >
> > The big question is : are containers/RG allowed to use *upto* their
> > limit always? In other words, will you typically setup limits such that
> > sum of all limits = max resource capacity?
> >
>
> If a user is really interested in ensuring that all scheduled jobs (or
> containers) get what they have asked for (guarantees) then making the
> sum of all container limits equal to total system limit is the right

> thing to do.
>
> > If it is setup like that, then what you are considering as limit is
> > actually guar no?
> >
> Right. And if we do it like this then it is up to sysadmin to configure
> the thing right without adding additional logic in kernel.

It won't be a complete solution, as the user won't be able to
 - set both guarantee and limit for a resource group
 - use limit on some and guarantee on some
 - optimize the usage of available resources
>
> -rohit
>
>
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

 ---------- ----------
   Chandra Seetharaman | Be careful what you choose....
           - sekharan@us.ibm.com | .....you may get it.
 ---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Rohit Seth on Wed, 13 Sep 2006 00:39:08 GMT
View Forum Message <> Reply to Message

On Tue, 2006-09-12 at 16:54 -0700, Chandra Seetharaman wrote:
> On Mon, 2006-09-11 at 16:58 -0700, Rohit Seth wrote:
> > On Mon, 2006-09-11 at 12:42 -0700, Chandra Seetharaman wrote:
> > > On Mon, 2006-09-11 at 12:10 -0700, Rohit Seth wrote:
> > > > On Mon, 2006-09-11 at 11:25 -0700, Chandra Seetharaman wrote:
> > >
> > > > > There could be a default container which doesn't have any guarantee or
> > > > > limit.
> > > >
> > > > First, I think it is critical that we allow processes to run outside of

> > > > any container (unless we know for sure that the penalty of running a
> > > > process inside a container is very very minimal).
> > >
> > > When I meant a default container I meant a default "resource group". In
> > > case of container that would be the default environment. I do not see
> > > any additional overhead associated with it, it is only associated with
> > > how resource are allocated/accounted.
> > >
> >
> > There should be some cost when you do atomic inc/dec accounting and
> > locks for add/remove resources from any container (including default
> > resource group). No?
>
> yes, it would be there, but is not heavy, IMO.

I think anything greater than 1% could be a concern for people who are
not very interested in containers but would be forced to live with them.

> >
> > > > And anything running outside a container should be limited by default
> > > > Linux settings.
> > >
> > > note that the resource available to the default RG will be (total system
> > > resource - allocated to RGs).
> >
> > I think it will be preferable to not change the existing behavior for
> > applications that are running outside any container (in your case
> > default resource group).
>
> hmm, when you provide QoS for a set of apps, you will affect (the
> resource availability of) other apps. I don't see any way around it. Any
> ideas ?

When I say, existing behavior, I mean not getting impacted by some
artificial limits that are imposed by container subsystem. IOW, if a
sysadmin is okay to have certain apps running outside of container then
he is basically forgoing any QoS for any container on that system.

>
> >
> > > > When you create containers and assign guarantees to each of them
> > > > make sure that you leave some amount of resource unassigned.
> > > >   ^^^^^ This will force the "default" container
> > > > with limits (indirectly). IMO, the whole guarantee feature gets defeated
> > >
> > > You _will_ have limits for the default RG even if we don't have

> > > guarantees.
> > >
> > > > the moment you bring in this fuzziness.
> > >
> > > Not really.
> > > - Each RG will have a guarantee and limit of each resource.
> > > - default RG will have (system resource - sum of guarantees)
> > > - Every RG will be guaranteed some amount of resource to provide QoS
> > > - Every RG will be limited at "limit" to prevent DoS attacks.
> > > - Whoever doesn't care either of those set them to don't care values.
> > >
> >
> > For the cases that put this don't care, do you depend on existing
> > reclaim algorithm (for memory) in kernel?
>
> Yes.

So one container with these don't care condition(s) can turn the whole
guarantee thing bad. Because existing kernel reclaimer does not know
about memory commitments to other containers. Right?

> >
> > > > That
> > > > > unassigned resources can be used by the default container or can be used
> > > > > by containers that want more than their guarantee (and less than their
> > > > > limit). This is how CKRM/RG handles this issue.
> > > > >
> > > > >
> > > >
> > > > It seems that a single notion of limit should suffice, and that limit
> > > > should more be treated as something beyond which that resource
> > > > consumption in the container will be throttled/not_allowed.
> > >
> > > As I stated in an earlier email "Limit only" approach can prevent a
> > > system from DoS attacks (and also fits the container model nicely),
> > > whereas to provide QoS one would need guarantee.
> > >
> > > Without guarantee, a RG that the admin cares about can starve if
> > > all/most of the other RGs consume upto their limits.
> > >
> > >
> >
> > If the limits are set appropriately so that containers total memory
> > consumption does not exceed the system memory then there shouldn't be
> > any QoS issue (to whatever extent it is applicable for specific
> > scenario).
>

> Then you will not be work-conserving (IOW over-committing), which is one
> of the main advantage of this type of feature.
>

If for the systems where QoS is important, not over-committing will be
fine (at least to start with).

-rohit

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4)
(added user memory)
Posted by Rohit Seth on Wed, 13 Sep 2006 00:43:23 GMT
View Forum Message <> Reply to Message

On Tue, 2006-09-12 at 17:02 -0700, Chandra Seetharaman wrote:
> On Tue, 2006-09-12 at 10:22 -0700, Rohit Seth wrote:
> > On Tue, 2006-09-12 at 16:14 +0530, Srivatsa Vaddagiri wrote:
> > > On Mon, Sep 11, 2006 at 12:10:31PM -0700, Rohit Seth wrote:
> > > > It seems that a single notion of limit should suffice, and that limit
> > > > should more be treated as something beyond which that resource
> > > > consumption in the container will be throttled/not_allowed.
> > >
> > > The big question is : are containers/RG allowed to use *upto* their
> > > limit always? In other words, will you typically setup limits such that
> > > sum of all limits = max resource capacity?
> > >
> >
> > If a user is really interested in ensuring that all scheduled jobs (or
> > containers) get what they have asked for (guarantees) then making the
> > sum of all container limits equal to total system limit is the right
> > thing to do.
> >
> > > If it is setup like that, then what you are considering as limit is
> > > actually guar no?
> > >
> > Right. And if we do it like this then it is up to sysadmin to configure
> > the thing right without adding additional logic in kernel.
>
> It won't be a complete solution, as the user won't be able to
> - set both guarantee and limit for a resource group
> - use limit on some and guarantee on some
> - optimize the usage of available resources

I think, if we have some of the dynamic resource limit adjustments
possible then some of the above functionality could be achieved. And I
think that could be a good start point.

-rohit

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Wed, 13 Sep 2006 01:10:18 GMT
View Forum Message <> Reply to Message

On Tue, 2006-09-12 at 17:39 -0700, Rohit Seth wrote:
<snip>
> > yes, it would be there, but is not heavy, IMO.
>
> I think anything greater than 1% could be a concern for people who are
> not very interested in containers but would be forced to live with them.

If they are not interested in resource management and/or containers, i do not think they need to pay.

>
> > >
> > > > >
> > > > > And anything running outside a container should be limited by default
> > > > > Linux settings.
> > > > >
> > > > note that the resource available to the default RG will be (total system
> > > > resource - allocated to RGs).
> > > >
> > > I think it will be preferable to not change the existing behavior for
> > > applications that are running outside any container (in your case
> > > default resource group).
> >
> > hmm, when you provide QoS for a set of apps, you will affect (the
> > resource availability of) other apps. I don't see any way around it. Any
> > ideas ?
>
> When I say, existing behavior, I mean not getting impacted by some
> artificial limits that are imposed by container subsystem. IOW, if a

That is what I understood and replied above.

> sysadmin is okay to have certain apps running outside of container then
> he is basically forgoing any QoS for any container on that system.

Not at all. If the container they are interested in is guaranteed, I do not see how apps running outside a container would affect them.

<snip>
> > > > Not really.
> > > > - Each RG will have a guarantee and limit of each resource.
> > > > - default RG will have (system resource - sum of guarantees)

> > > > - Every RG will be guaranteed some amount of resource to provide QoS
> > > > - Every RG will be limited at "limit" to prevent DoS attacks.
> > > > - Whoever doesn't care either of those set them to don't care values.
> > > >
> > >
> > > For the cases that put this don't care, do you depend on existing
> > > reclaim algorithm (for memory) in kernel?
> >
> > Yes.
>
> So one container with these don't care condition(s) can turn the whole
> guarantee thing bad. Because existing kernel reclaimer does not know
> about memory commitments to other containers. Right?

No, the reclaimer would free up pages associated with the don't care RGs
(as the user don't care about the resource made available to them).

<snip>
> > > If the limits are set appropriately so that containers total memory
> > > consumption does not exceed the system memory then there shouldn't be
> > > any QoS issue (to whatever extent it is applicable for specific
> > > scenario).
> >
> > Then you will not be work-conserving (IOW over-committing), which is one
> > of the main advantage of this type of feature.
> >
>
> If for the systems where QoS is important, not over-committing will be
> fine (at least to start with).

The problem is that you can't do it with just limit.

--

 -- ----------
   Chandra Seetharaman | Be careful what you choose....
            - sekharan@us.ibm.com | ....you may get it.
 -- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters
(v4) (added user memory)
Posted by Chandra Seetharaman on Wed, 13 Sep 2006 01:13:10 GMT
View Forum Message <> Reply to Message

On Tue, 2006-09-12 at 17:43 -0700, Rohit Seth wrote:
<snip>

> > It won't be a complete solution, as the user won't be able to
> > - set both guarantee and limit for a resource group
> > - use limit on some and guarantee on some
> > - optimize the usage of available resources
>
> I think, if we have some of the dynamic resource limit adjustments
> possible then some of the above functionality could be achieved. And I
> think that could be a good start point.

Yes, dynamic resource adjustments should be available. But, you can't
expect the sysadmin to sit around and keep tweaking the limits so as to
achieve the QoS he wants. (Even if you have an application sitting and
doing it, as I pointed in other email it may not be possible for
different scenarios).
>
> -rohit
>
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

 ---------- ----------
   Chandra Seetharaman | Be careful what you choose....
            - sekharan@us.ibm.com | .....you may get it.
 ---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Rohit Seth on Wed, 13 Sep 2006 01:25:51 GMT
View Forum Message <> Reply to Message

On Tue, 2006-09-12 at 18:10 -0700, Chandra Seetharaman wrote:
> On Tue, 2006-09-12 at 17:39 -0700, Rohit Seth wrote:
> <snip>
> > > yes, it would be there, but is not heavy, IMO.
> >
> > I think anything greater than 1% could be a concern for people who are
> > not very interested in containers but would be forced to live with them.
>

> If they are not interested in resource management and/or containers, i
> do not think they need to pay.
> >

Think of a single kernel from a vendor that has container support built
in.

> > > >
> > > > > >
> > > > > > And anything running outside a container should be limited by default
> > > > > > Linux settings.
> > > > > >
> > > > > note that the resource available to the default RG will be (total system
> > > > > resource - allocated to RGs).
> > > > >
> > > > I think it will be preferable to not change the existing behavior for
> > > > applications that are running outside any container (in your case
> > > > default resource group).
> > >
> > hmm, when you provide QoS for a set of apps, you will affect (the
> > resource availability of) other apps. I don't see any way around it. Any
> > ideas ?
> >
> When I say, existing behavior, I mean not getting impacted by some
> artificial limits that are imposed by container subsystem. IOW, if a
>
> That is what I understood and replied above.
> > sysadmin is okay to have certain apps running outside of container then
> > he is basically forgoing any QoS for any container on that system.
>
> Not at all. If the container they are interested in is guaranteed, I do
> not see how apps running outside a container would affect them.
>

Because the kernel (outside the container subsystem) doesn't know of
these guarantees...unless you modify the page allocator to have another
variant of overcommit memory.

> <snip>
> > > > > Not really.
> > > > > - Each RG will have a guarantee and limit of each resource.
> > > > > - default RG will have (system resource - sum of guarantees)
> > > > > - Every RG will be guaranteed some amount of resource to provide QoS
> > > > > - Every RG will be limited at "limit" to prevent DoS attacks.
> > > > > - Whoever doesn't care either of those set them to don't care values.
> > > > > >
> > > > >
> > > > For the cases that put this don't care, do you depend on existing

> > > > reclaim algorithm (for memory) in kernel?
> > >
> > > Yes.
> >
> > So one container with these don't care condition(s) can turn the whole
> > guarantee thing bad. Because existing kernel reclaimer does not know
> > about memory commitments to other containers. Right?
>
> No, the reclaimer would free up pages associated with the don't care RGs
> (as the user don't care about the resource made available to them).
>

And how will the kernel reclaimer know which RGs are don't care?

-rohit

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Srivatsa Vaddagiri on Wed, 13 Sep 2006 04:41:24 GMT
View Forum Message <> Reply to Message

On Tue, Sep 12, 2006 at 06:25:51PM -0700, Rohit Seth wrote:
> And how will the kernel reclaimer know which RGs are don't care?

The UBC code can provide this information certainly. For ex: in my CPU controller patch, I do keep track of such don't care groups in the controller itself:

 http://lkml.org/lkml/2006/8/20/120

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Wed, 13 Sep 2006 08:06:41 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:
> On Tue, 2006-09-12 at 14:48 +0400, Pavel Emelianov wrote:
> <snip>
>
>>> I do not think it is that simple since

>>> - there is typically more than one class I want to set guarantee to
>>> - I will not able to use both limit and guarantee
>>> - Implementation will not be work-conserving.
>>>
>>> Also, How would you configure the following in your model ?
>>>
>>> 5 classes: Class A(10, 40), Class B(20, 100), Class C (30, 100), Class D
>>> (5, 100), Class E(15, 50); (class_name(guarantee, limit))
>>>
>>>
>> What's the total memory amount on the node? Without it it's hard to make
>> any
>> guarantee.
>>
>
> I wrote the example treating them as %, so 100 would be the total amount
> of memory.
>
OK. Then limiting must be done this way (unreclaimable limit/total limit)
A (15/40)
B (25/100)
C (35/100)
D (10/100)
E (20/50)
In this case each group will receive it's guarantee for sure.

E.g. even if A, B, E and D will eat all it's unreclaimable memory then
we'll have
100 - 15 - 25 - 20 - 10 = 30% of memory left (maybe after reclaiming) which
is perfectly enough for C's guarantee.
>
>>> "Limit only" approach works for DoS prevention. But for providing QoS
>>> you would need guarantee.
>>>
>>>
>> You may not provide guarantee on physycal resource for a particular group
>> without limiting its usage by other groups. That's my major idea.
>>
>
> I agree with that, but the other way around (i.e provide guarantee for
> everyone by imposing limits on everyone) is what I am saying is not
> possible.
Then how do you make sure that memory WILL be available when the group needs
it without limiting the others in a proper way?

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user

memory)
Posted by Srivatsa Vaddagiri on Wed, 13 Sep 2006 12:15:25 GMT
View Forum Message <> Reply to Message

On Wed, Sep 13, 2006 at 12:06:41PM +0400, Pavel Emelianov wrote:
> OK. Then limiting must be done this way (unreclaimable limit/total limit)
> A (15/40)
> B (25/100)
> C (35/100)

s/35/30?

Also the different b/n total and unreclaimable limits goes towards
limiting reclaimable memory i suppose? And 1st limit seems to be a
hard-limit while the 2nd one is soft?

> D (10/100)
> E (20/50)
> In this case each group will receive it's guarantee for sure.
>
> E.g. even if A, B, E and D will eat all it's unreclaimable memory then
> we'll have
> 100 - 15 - 25 - 20 - 10 = 30% of memory left (maybe after reclaiming) which
> is perfectly enough for C's guarantee.

I agree by carefully choosing these limits, we can provide some sort of
QoS, which is a good step to begin with.

--
Regards,
vatsa

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Wed, 13 Sep 2006 13:35:30 GMT
View Forum Message <> Reply to Message

Srivatsa Vaddagiri wrote:
> On Wed, Sep 13, 2006 at 12:06:41PM +0400, Pavel Emelianov wrote:
>> OK. Then limiting must be done this way (unreclaimable limit/total limit)
>> A (15/40)
>> B (25/100)
>> C (35/100)
>
> s/35/30?
Hmmm... No, it must be 35. It IS higher than guarantee you proposed,

but that's OK to have a limit higher than guarantee, isn't it?
>
> Also the different b/n total and unreclaimable limits goes towards
> limiting reclaimable memory i suppose? And 1st limit seems to be a
> hard-limit while the 2nd one is soft?
The first limit (let's call it soft one) is limit for unreclaimable
memory, the second (hard limit) - for booth reclaimable and not.

The ploicy is
1. if BC tries to *mmap()* unreclaimable region (e.g. w/o backed
   file as moving page to swap is not a pure "reclamation") then
   check the soft limit and prohibit mapping in case it is hit;
2. if BC tries to *touch* a page - then check for the hard limit
   and start reclaiming this BC's pages if the limit is hit.

That's how guarantees can be met. Current BC code does perform the
first check and gives you all the levers for the second one - just
the patch(es) with reclamation mechanism is required.
>
>> D (10/100)
>> E (20/50)
>> In this case each group will receive it's guarantee for sure.
>>
>> E.g. even if A, B, E and D will eat all it's unreclaimable memory then
>> we'll have
>> 100 - 15 - 25 - 20 - 10 = 30% of memory left (maybe after reclaiming) which
>> is perfectly enough for C's guarantee.
>
> I agree by carefully choosing these limits, we can provide some sort of
> QoS, which is a good step to begin with.
Sure. As I've said - soft limiting is already done with BC patches, the
hard one is not prohibited by BC (BCs even prepare a good pad for it).
When reclaiming is done we'll have a hard limit described above.

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Wed, 13 Sep 2006 13:39:22 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:
> Pavel Emelianov wrote:
> <snip>
>
>>>>>> E.g. I have a node with 1Gb of ram and 10 containers with 100Mb
>>>>>> guarantee each.
>>>>>> I want to start one more. What shall I do not to break guarantees?
>>>>> Don't start the new container or change the guarantees of the

>>>>> existing
>>>>> ones
>>>>> to accommodate this one :) The QoS design (done by the administrator)
>>>>> should
>>>>> take care of such use-cases. It would be perfectly ok to have a
>>>>> container
>>>>> that does not care about guarantees to set their guarantee to 0
>>>>> and set
>>>>> their limit to the desired value. As Chandra has been stating we
>>>>> need two
>>>>> parameters (guarantee, limit), either can be optional, but not both.
>>>> If I set up 9 groups to have 100Mb limit then I have 100Mb assured (on
>>>> 1Gb node)
>>>> for the 10th one exactly. And I do not have to set up any guarantee as
>>>> it won't affect
>>>> anything. So what a guarantee parameter is needed for?
>>> This use case works well for providing guarantee to one container.
>>> What if
>>> I want guarantees of 100Mb and 200Mb for two containers? How do I setup
>>> the system using limits?
>> You may set any value from 100 up to 800 Mb for the first one and
>> 200-900Mb for
>> the second. In case of no other groups first will receive its 100Mb for
>> sure and
>> so does the second. If there are other groups - their guarantees should
>> be concerned.
>
> If I add another group with a guarantee of 100MB, then its limit will
> be anywhere between 100MB-800MB ?
I've described this in details in my letter to sekharan@.
>
> I do not understand the guarantees being concerned part.
>
>>> Even I restrict everyone else to 700Mb. With this I cannot be sure that
>>> the remaining 300Mb will be distributed as 100Mb and 200Mb.
>> There's no "everyone else" here - we're talking about a "static" case.
>> When new group arrives we need to recalculate guarantees as you said.
>
> I was speaking in general where we have 'n' groups, so thats why I had
> "everyone else".
Well, when we talk about guarantee this implies that the number
of group doesn't chage - when it does limits/guarantees generally
must be recalculated to satisfy new group set.

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)

On Tue, 2006-09-12 at 18:25 -0700, Rohit Seth wrote:
> On Tue, 2006-09-12 at 18:10 -0700, Chandra Seetharaman wrote:
> > On Tue, 2006-09-12 at 17:39 -0700, Rohit Seth wrote:
> > <snip>
> > > > yes, it would be there, but is not heavy, IMO.
> > >
> > > I think anything greater than 1% could be a concern for people who are
> > > not very interested in containers but would be forced to live with them.
> >
> > If they are not interested in resource management and/or containers, i
> > do not think they need to pay.
> > >
>
> Think of a single kernel from a vendor that has container support built
> in.

Ok. Understood.

Here are results of some of the benchmarks we have run in the past
(April 2005) with CKRM which showed no/negligible performance impact in
that scenario.
 http://marc.theaimsgroup.com/?l=ckrm-tech&m=111325064322 305&w=2
 http://marc.theaimsgroup.com/?l=ckrm-tech&m=111385973226 267&w=2
 http://marc.theaimsgroup.com/?l=ckrm-tech&m=111291409731 929&w=2
>
<snip>

> > Not at all. If the container they are interested in is guaranteed, I do
> > not see how apps running outside a container would affect them.
> >
>
> Because the kernel (outside the container subsystem) doesn't know of

The core resource subsystem (VM subsystem for memory) would know about
the guarantees and don't cares, and it would handle it appropriately.

> these guarantees...unless you modify the page allocator to have another
> variant of overcommit memory.
>

<snip>
>
> > No, the reclaimer would free up pages associated with the don't care RGs
> > (as the user don't care about the resource made available to them).
> >
>

> And how will the kernel reclaimer know which RGs are don't care?

By looking into the beancounter associated with the container/RG

--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Wed, 13 Sep 2006 22:31:04 GMT
View Forum Message <> Reply to Message

On Wed, 2006-09-13 at 12:06 +0400, Pavel Emelianov wrote:
> Chandra Seetharaman wrote:
> > On Tue, 2006-09-12 at 14:48 +0400, Pavel Emelianov wrote:
> > <snip>
> >
> >>> I do not think it is that simple since
> >>> - there is typically more than one class I want to set guarantee to
> >>> - I will not able to use both limit and guarantee
> >>> - Implementation will not be work-conserving.
> >>>
> >>> Also, How would you configure the following in your model ?
> >>>
> >>> 5 classes: Class A(10, 40), Class B(20, 100), Class C (30, 100), Class D
> >>> (5, 100), Class E(15, 50); (class_name(guarantee, limit))
> >>>
> >>>
> >> What's the total memory amount on the node? Without it it's hard to make
> >> any
> >> guarantee.
> >>
> >
> > I wrote the example treating them as %, so 100 would be the total amount
> > of memory.
> >
> OK. Then limiting must be done this way (unreclaimable limit/total limit)
> A (15/40)
> B (25/100)
> C (35/100)
> D (10/100)
> E (20/50)

> In this case each group will receive it's guarantee for sure.
>
> E.g. even if A, B, E and D will eat all it's unreclaimable memory then
> we'll have
> 100 - 15 - 25 - 20 - 10 = 30% of memory left (maybe after reclaiming) which
> is perfectly enough for C's guarantee.

How did you arrive at the +5 number ?

What if I have 40 containers each with 2% guarantee ? what do we do
then ? and many other different combinations (what I gave was not the
only scenario).

> >
> >>> "Limit only" approach works for DoS prevention. But for providing QoS
> >>> you would need guarantee.
> >>>
> >>>
> >> You may not provide guarantee on physycal resource for a particular group
> >> without limiting its usage by other groups. That's my major idea.
> >>
> >
> > I agree with that, but the other way around (i.e provide guarantee for
> > everyone by imposing limits on everyone) is what I am saying is not
> > possible.
> Then how do you make sure that memory WILL be available when the group needs
> it without limiting the others in a proper way?

You could limit others only if you _know_ somebody is not getting what
they are supposed to get (based on guarantee).

>
>
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC:
resource beancounters (v4) (added user memory)
Posted by Rohit Seth on Thu, 14 Sep 2006 01:22:28 GMT
View Forum Message <> Reply to Message

On Wed, 2006-09-13 at 15:20 -0700, Chandra Seetharaman wrote:
> On Tue, 2006-09-12 at 18:25 -0700, Rohit Seth wrote:
> > On Tue, 2006-09-12 at 18:10 -0700, Chandra Seetharaman wrote:
> > > On Tue, 2006-09-12 at 17:39 -0700, Rohit Seth wrote:
> > > <snip>
> > > > > yes, it would be there, but is not heavy, IMO.
> > > >
> > > > I think anything greater than 1% could be a concern for people who are
> > > > not very interested in containers but would be forced to live with them.
> > >
> > > If they are not interested in resource management and/or containers, i
> > > do not think they need to pay.
> > >
> >
> > Think of a single kernel from a vendor that has container support built
> > in.
>
> Ok. Understood.
>
> Here are results of some of the benchmarks we have run in the past
> (April 2005) with CKRM which showed no/negligible performance impact in
> that scenario.
> http://marc.theaimsgroup.com/?l=ckrm-tech&m=111325064322 305&w=2
> http://marc.theaimsgroup.com/?l=ckrm-tech&m=111385973226 267&w=2
> http://marc.theaimsgroup.com/?l=ckrm-tech&m=111291409731 929&w=2
> >

These are good results. But I still think the cost will increase over a
period of time as more logic gets added. Any data on microbenchmarks
like lmbench.

> <snip>
>
> > > Not at all. If the container they are interested in is guaranteed, I do
> > > not see how apps running outside a container would affect them.
> > >
> >
> > Because the kernel (outside the container subsystem) doesn't know of
>
> The core resource subsystem (VM subsystem for memory) would know about
> the guarantees and don't cares, and it would handle it appropriately.
>

...meaning hooks in the generic kernel reclaim algorithm. Getting
something like that in mainline will be at best tricky.

-rohit

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Thu, 14 Sep 2006 07:53:18 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:
> On Wed, 2006-09-13 at 12:06 +0400, Pavel Emelianov wrote:
>
>> Chandra Seetharaman wrote:
>>
>>> On Tue, 2006-09-12 at 14:48 +0400, Pavel Emelianov wrote:
>>> <snip>
>>>
>>>
>>>> I do not think it is that simple since
>>>> - there is typically more than one class I want to set guarantee to
>>>> - I will not able to use both limit and guarantee
>>>> - Implementation will not be work-conserving.
>>>>
>>>> Also, How would you configure the following in your model ?
>>>>
>>>> 5 classes: Class A(10, 40), Class B(20, 100), Class C (30, 100), Class D
>>>> (5, 100), Class E(15, 50); (class_name(guarantee, limit))
>>>>
>>>>
>>>>
>>> What's the total memory amount on the node? Without it it's hard to make
>>> any
>>> guarantee.
>>>
>>>
>> I wrote the example treating them as %, so 100 would be the total amount
>> of memory.
>>
>>
> OK. Then limiting must be done this way (unreclaimable limit/total limit)
> A (15/40)
> B (25/100)
> C (35/100)
> D (10/100)
> E (20/50)

>> In this case each group will receive it's guarantee for sure.
>>
>> E.g. even if A, B, E and D will eat all it's unreclaimable memory then
>> we'll have
>> 100 - 15 - 25 - 20 - 10 = 30% of memory left (maybe after reclaiming) which
>> is perfectly enough for C's guarantee.
>>
>
> How did you arrive at the +5 number ?
>
I've solved a linear equations set :)
> What if I have 40 containers each with 2% guarantee ? what do we do
> then ? and many other different combinations (what I gave was not the
> _only_ scenario).
>
Then you need to solve a set of 40 equations. This sounds weird, but
don't afraid - sets like these are solved lightly.
>
>>>
>>>
>>>>> "Limit only" approach works for DoS prevention. But for providing QoS
>>>>> you would need guarantee.
>>>>>
>>>>>
>>>>>
>>>> You may not provide guarantee on physycal resource for a particular group
>>>> without limiting its usage by other groups. That's my major idea.
>>>>
>>>>
>>> I agree with that, but the other way around (i.e provide guarantee for
>>> everyone by imposing limits on everyone) is what I am saying is not
>>> possible.
>>>
>> Then how do you make sure that memory WILL be available when the group needs
>> it without limiting the others in a proper way?
>>
>
> You could limit others only if you _know_ somebody is not getting what
> they are supposed to get (based on guarantee).
>
I don't understand your idea. Limit does _not_ imply anything - it's
just a limit.
You may limit anything to anyone w/o bothering the consequences.
Guarantee implies that the resource you guarantee will be available and
this "will be" is something not that easy.

So I repeat my question - how can you be sure that these X megabytes you
guarantee to some group won't be used by others so that you won't be able

to reclaim them?

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Balbir Singh on Thu, 14 Sep 2006 08:06:54 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:

> I don't understand your idea. Limit does _not_ imply anything - it's
> just a limit.
> You may limit anything to anyone w/o bothering the consequences.
> Guarantee implies that the resource you guarantee will be available and
> this "will be" is something not that easy.
>
> So I repeat my question - how can you be sure that these X megabytes you
> guarantee to some group won't be used by others so that you won't be able
> to reclaim them?
>
>

May be we can treat a guarantee as a soft guarantee. A soft
guarantee would imply that when a group needs its guaranteed resources, the
system makes its best effort to make it available.

In soft guarantees, resources not actively used by a group can be shared with
other groups.

Hard guarantees would probably require reserving the resource in advance and
sharing of the resources not used, with other groups, might not be possible.

Comments?

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Thu, 14 Sep 2006 13:02:48 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:

> Pavel Emelianov wrote:
>
>> I don't understand your idea. Limit does _not_ imply anything - it's
>> just a limit.
>> You may limit anything to anyone w/o bothering the consequences.
>> Guarantee implies that the resource you guarantee will be available and
>> this "will be" is something not that easy.
>>
>> So I repeat my question - how can you be sure that these X megabytes you
>> guarantee to some group won't be used by others so that you won't be
>> able
>> to reclaim them?
>>
>>
>
> May be we can treat a guarantee as a soft guarantee. A soft
> guarantee would imply that when a group needs its guaranteed
> resources, the
> system makes its best effort to make it available.
>
> In soft guarantees, resources not actively used by a group can be
> shared with
> other groups.
>
> Hard guarantees would probably require reserving the resource in
> advance and
> sharing of the resources not used, with other groups, might not be
> possible.
>
> Comments?
>
Reserving in advance means that sometimes you won't be able to start a
new group without taking back some of reserved pages. This is ... strange.

I think that a satisfactory solution now would be:
 - limit unreclaimable memory during mmap() against soft limit to prevent
   potential rejects during page faults;
 - reclaim memory in case of hitting hard limit;
 - guarantees are done via setting soft and hard limits as I've shown
before.

The question still open is wether or not to account fractions.
I propose to skip fractions for a while and try to charge the page to
it's first user.

So final BC design is:
1. three resources:
      - kernel memory

- user unreclaimable memory
- user reclaimable memory

2. unreclaimable memory is charged "in advance", reclaimable
   is charged "on demand" with reclamation if needed
3. each object (kernel one or user page) is charged to the
   first user
4. each resource controller declares it's own
   - meaning of "limit" parameter (percent/size/bandwidth/etc)
   - behaviour on changing limit (e.g. reclamation)
   - behaviour on hitting the limit (e.g. reclamation)
5. BC can be assigned to any task by pid (not just current)
   without recharging currently charged resources.

Subject: Re: [ckrm-tech] [PATCH] BC:
resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Thu, 14 Sep 2006 23:13:23 GMT
View Forum Message <> Reply to Message

On Wed, 2006-09-13 at 18:22 -0700, Rohit Seth wrote:
<snip>
> >
> > Here are results of some of the benchmarks we have run in the past
> > (April 2005) with CKRM which showed no/negligible performance impact in
> > that scenario.
> > http://marc.theaimsgroup.com/?l=ckrm-tech&m=111325064322 305&w=2
> > http://marc.theaimsgroup.com/?l=ckrm-tech&m=111385973226 267&w=2
> > http://marc.theaimsgroup.com/?l=ckrm-tech&m=111291409731 929&w=2
> > >
>
>
> These are good results. But I still think the cost will increase over a
> period of time as more logic gets added. Any data on microbenchmarks

IMO, overhead may not increase for a _non-user_ of the feature.

> like lmbench.

I think we have run those, but I could not find the results in the
mailing list.
>
> > <snip>
> >
> > > > Not at all. If the container they are interested in is guaranteed, I do
> > > > not see how apps running outside a container would affect them.
> > > >
> > >
> > > Because the kernel (outside the container subsystem) doesn't know of

> >
> > The core resource subsystem (VM subsystem for memory) would know about
> > the guarantees and don't cares, and it would handle it appropriately.
> >
>
> ...meaning hooks in the generic kernel reclaim algorithm. Getting
> something like that in mainline will be at best tricky.

Yes, it does mean doing something in the reclamation path.

>
>
> -rohit
>
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Thu, 14 Sep 2006 23:42:44 GMT
View Forum Message <> Reply to Message

On Thu, 2006-09-14 at 11:53 +0400, Pavel Emelianov wrote:

<snip>

> > What if I have 40 containers each with 2% guarantee ? what do we do
> > then ? and many other different combinations (what I gave was not the
> > _only_ scenario).
> >
> Then you need to solve a set of 40 equations. This sounds weird, but
> don't afraid - sets like these are solved lightly.

extrapolate that to a varying # of permutations and real time changes in
the system workload. Won't it be complex ?

Wouldn't it be a lot simpler if we have the guarantee support instead ?
Why you do not like guarantee ? :)

<snip>

> >> Then how do you make sure that memory WILL be available when the group needs
> >> it without limiting the others in a proper way?

> >>
> >
> > You could limit others only if you _know_ somebody is not getting what
> > they are supposed to get (based on guarantee).
> >
> I don't understand your idea. Limit does _not_ imply anything - it's
> just a limit.

I didn't mean "limit" as defined in BC. I meant it in the generic sense.
IOW, if we have to provide guarantees then it would limit other RGs from
getting that (amount of guaranteed) resource.

> You may limit anything to anyone w/o bothering the consequences.
> Guarantee implies that the resource you guarantee will be available and
> this "will be" is something not that easy.
>
> So I repeat my question - how can you be sure that these X megabytes you
> guarantee to some group won't be used by others so that you won't be able
> to reclaim them?

It depends on how the memory controller is implemented. It could be
implemented in different ways:
 - reclamation path will _not_ free pages belonging to a RG that is
   below its guarantee.
 - allocation from a "over guarantee" RG can succeed iff there is
   memory after satisfying all guarantees (or will free pages from the
   requesting RG before it will succeed).
 - ...

BTW, my point is to have guarantees for _all_ resources not just memory.

>
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Fri, 15 Sep 2006 00:02:21 GMT
View Forum Message <> Reply to Message

On Thu, 2006-09-14 at 17:02 +0400, Pavel Emelianov wrote:

<snip>
> >
> Reserving in advance means that sometimes you won't be able to start a
> new group without taking back some of reserved pages. This is ... strange.

I do not see it strange. At the time of creation, user sees the failure
(that there isn't enough resource to provide the required/requested
guarantee) and can act accordingly.

BTW, VMware does it this way.

>
> I think that a satisfactory solution now would be:
> - limit unreclaimable memory during mmap() against soft limit to prevent
> potential rejects during page faults;

we can have guarantee and still handle it this way.
> - reclaim memory in case of hitting hard limit;
> - guarantees are done via setting soft and hard limits as I've shown
> before.

complexity is high in doing that.
>
> The question still open is wether or not to account fractions.
> I propose to skip fractions for a while and try to charge the page to
> it's first user.

sounds fine

>
> So final BC design is:
> 1. three resources:
> - kernel memory
> - user unreclaimable memory
> - user reclaimable memory

should be able to get other controllers also under this framework.

> 2. unreclaimable memory is charged "in advance", reclaimable
> is charged "on demand" with reclamation if needed
> 3. each object (kernel one or user page) is charged to the
> first user

> 4. each resource controller declares it's own
> - meaning of "limit" parameter (percent/size/bandwidth/etc)
> - behaviour on changing limit (e.g. reclamation)
> - behaviour on hitting the limit (e.g. reclamation)
> 5. BC can be assigned to any task by pid (not just current)
> without recharging currently charged resources.

Please see the emails i sent earlier in this context:
http://marc.theaimsgroup.com/?l=ckrm-tech&m=115593001810 616&w=2

We would need at least:
- BC should be created/deleted explicitly by the user
- cleaner interface for controller writers

--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Fri, 15 Sep 2006 07:15:49 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:
> On Thu, 2006-09-14 at 11:53 +0400, Pavel Emelianov wrote:
>
> <snip>
>
>
>>> What if I have 40 containers each with 2% guarantee ? what do we do
>>> then ? and many other different combinations (what I gave was not the
>>> _only_ scenario).
>>>
>>>
>> Then you need to solve a set of 40 equations. This sounds weird, but
>> don't afraid - sets like these are solved lightly.
>>
>
> extrapolate that to a varying # of permutations and real time changes in
> the system workload. Won't it be complex ?
>
I have a C program that computes limits to obtain desired guarantees
in a single 'for (i = 0; i < n; n++)' loop for any given set of guarantees.
With all error handling, beautifull output, nice formatting etc it weights

only 60 lines.
> Wouldn't it be a lot simpler if we have the guarantee support instead ?
> Why you do not like guarantee ? :)
>
I do not 'do not like guarantee'. I'm just sure that there are two ways
for providing guarantee (for unreclaimable resorces):
1. reserving resource for group in advance
2. limit resource for others
Reserving is worse as it is essentially limiting (you cut off 100Mb from
1Gb RAM thus limiting the other groups by 900Mb RAM), but this limiting
is too strict - you _have_ to reserve less than RAM size. Limiting in
run-time is more flexible (you may create an overcommited BC if you
want to) and leads to the same result - guarantee.
> <snip>
>
[snip]

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Fri, 15 Sep 2006 07:21:39 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:
> On Thu, 2006-09-14 at 17:02 +0400, Pavel Emelianov wrote:
>
> <snip>
>
>> Reserving in advance means that sometimes you won't be able to start a
>> new group without taking back some of reserved pages. This is ... strange.
>>
>
> I do not see it strange. At the time of creation, user sees the failure
> (that there isn't enough resource to provide the required/requested
> guarantee) and can act accordingly.
>
> BTW, VMware does it this way.
>
And VPS density in VMware is MUCH lower than in
OpenVZ with beancounters :)
>
>> I think that a satisfactory solution now would be:
>> - limit unreclaimable memory during mmap() against soft limit to prevent
>>   potential rejects during page faults;
>>
>
> we can have guarantee and still handle it this way.
>

>> - reclaim memory in case of hitting hard limit;
>> - guarantees are done via setting soft and hard limits as I've shown
>> before.
>>
>
> complexity is high in doing that.
>
Nope. I've already said in another letter that a program of 60 lines
does this in a single loop.
>> The question still open is wether or not to account fractions.
>> I propose to skip fractions for a while and try to charge the page to
>> it's first user.
>>
>
> sounds fine
>
>
>> So final BC design is:
>> 1. three resources:
>>   - kernel memory
>>   - user unreclaimable memory
>>   - user reclaimable memory
>>
>
> should be able to get other controllers also under this framework.
>
OK. But note, that it's easy to add new resource to current BC code.
The most difficult thing is placing 'charge/uncharge' calls over the kernel.
>
>> 2. unreclaimable memory is charged "in advance", reclaimable
>>   is charged "on demand" with reclamation if needed
>> 3. each object (kernel one or user page) is charged to the
>>   first user
>> 4. each resource controller declares it's own
>>   - meaning of "limit" parameter (percent/size/bandwidth/etc)
>>   - behaviour on changing limit (e.g. reclamation)
>>   - behaviour on hitting the limit (e.g. reclamation)
>> 5. BC can be assigned to any task by pid (not just current)
>>   without recharging currently charged resources.
>>
>
> Please see the emails i sent earlier in this context:
> http://marc.theaimsgroup.com/?l=ckrm-tech&m=115593001810 616&w=2
>
> We would need at least:
> - BC should be created/deleted explicitly by the user
> - cleaner interface for controller writers
>

OK.
Next week we'll try to send a new set of patches.

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by dev on Fri, 15 Sep 2006 08:46:14 GMT
View Forum Message <> Reply to Message

> <snip>
>
>>Reserving in advance means that sometimes you won't be able to start a
>>new group without taking back some of reserved pages. This is ... strange.
>
>
> I do not see it strange. At the time of creation, user sees the failure
> (that there isn't enough resource to provide the required/requested
> guarantee) and can act accordingly.
>
> BTW, VMware does it this way.
This is not true at least for ESX server.
It overcommits memory and does dirty tricks like balooning to free memory then.

[...]

> We would need at least:
> - BC should be created/deleted explicitly by the user
> - cleaner interface for controller writers
why do you bother for the last too much?
The number of controlers is quite limited...

Kirill

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by dev on Fri, 15 Sep 2006 08:51:24 GMT
View Forum Message <> Reply to Message

Chandra,

>>>>What if I have 40 containers each with 2% guarantee ? what do we do
>>>>then ? and many other different combinations (what I gave was not the
>>>>_only_ scenario).
>>>>
>>>>
>>>

>>>Then you need to solve a set of 40 equations. This sounds weird, but
>>>don't afraid - sets like these are solved lightly.
>>>
>>
>>extrapolate that to a varying # of permutations and real time changes in
>>the system workload. Won't it be complex ?
>>
>
> I have a C program that computes limits to obtain desired guarantees
> in a single 'for (i = 0; i < n; n++)' loop for any given set of guarantees.
> With all error handling, beautifull output, nice formatting etc it weights
> only 60 lines.
>
>>Wouldn't it be a lot simpler if we have the guarantee support instead ?
the calculation above doesn't seem hard :)

>>Why you do not like guarantee ? :)

> I do not 'do not like guarantee'. I'm just sure that there are two ways
> for providing guarantee (for unreclaimable resorces):
> 1. reserving resource for group in advance
> 2. limit resource for others
> Reserving is worse as it is essentially limiting (you cut off 100Mb from
> 1Gb RAM thus limiting the other groups by 900Mb RAM), but this limiting
> is too strict - you _have_ to reserve less than RAM size. Limiting in
> run-time is more flexible (you may create an overcommited BC if you
> want to) and leads to the same result - guarantee.
I think this deserves putting on Wiki.
It is very good clear point.

Chanrda, do you propose some 3rd way (we are unaware of) of implementing guarantees?

Thanks,
Kirill

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by dev on Fri, 15 Sep 2006 08:53:54 GMT
View Forum Message <> Reply to Message

> CKRM/RG handles it this way:
>
> Amount of a resource a child RG gets is the ratio of its share value to
> the parent's total # of shares. Children's resource allocation can be
> changed just by changing the parent's total # of shares.
>
> If you case about initial situation would be:

> Total memory in the system 100MB
> parent's total # of shares: 100 (1 share == 1MB)
> 10 children with # of shares: 10 (i.e each children has 10MB)
>
> When I want to add another child, just change parent's total # of shares
> to be say 125:
> Total memory in the system 100MB
> parent's total # of shares: 125 (1 share == 0.8MB)
> 10 children with # of shares: 10 (i.e each children has 8MB)
> Now you are left with 25 shares (or 20MB) that you can assign to new
> child(ren) as you please.

setting memory in "shares" doesn't look user friendly at all...

Kirill

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Fri, 15 Sep 2006 11:15:45 GMT
View Forum Message <> Reply to Message

Kirill Korotaev wrote:

[snip]
>> I have a C program that computes limits to obtain desired guarantees
>> in a single 'for (i = 0; i < n; n++)' loop for any given set of guarantees.
>> With all error handling, beautifull output, nice formatting etc it weights
>> only 60 lines.

Look at http://wiki.openvz.org/Containers/Guarantees_for_resources
I've described there how a guarantee can be get with limiting in details.

[snip]

>> I do not 'do not like guarantee'. I'm just sure that there are two ways
>> for providing guarantee (for unreclaimable resorces):
>> 1. reserving resource for group in advance
>> 2. limit resource for others
>> Reserving is worse as it is essentially limiting (you cut off 100Mb from
>> 1Gb RAM thus limiting the other groups by 900Mb RAM), but this limiting
>> is too strict - you _have_ to reserve less than RAM size. Limiting in
>> run-time is more flexible (you may create an overcommited BC if you
>> want to) and leads to the same result - guarantee.
> I think this deserves putting on Wiki.
> It is very good clear point.

This is also on the page I gave link at.

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Balbir Singh on Mon, 18 Sep 2006 08:25:55 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Kirill Korotaev wrote:
>
> [snip]
>>> I have a C program that computes limits to obtain desired guarantees
>>> in a single 'for (i = 0; i < n; n++)' loop for any given set of guarantees.
>>> With all error handling, beautifull output, nice formatting etc it weights
>>> only 60 lines.
>
> Look at http://wiki.openvz.org/Containers/Guarantees_for_resources
> I've described there how a guarantee can be get with limiting in details.
>
> [snip]
>
>>> I do not 'do not like guarantee'. I'm just sure that there are two ways
>>> for providing guarantee (for unreclaimable resorces):
>>> 1. reserving resource for group in advance
>>> 2. limit resource for others
>>> Reserving is worse as it is essentially limiting (you cut off 100Mb from
>>> 1Gb RAM thus limiting the other groups by 900Mb RAM), but this limiting
>>> is too strict - you _have_ to reserve less than RAM size. Limiting in
>>> run-time is more flexible (you may create an overcommited BC if you
>>> want to) and leads to the same result - guarantee.
>> I think this deserves putting on Wiki.
>> It is very good clear point.
>
> This is also on the page I gave link at.
>

This approach has the following disadvantages

1. Lets consider initialization - When we create 'n' groups initially, we need to spend O(n^2) time to assign guarantees.
2. Every time a limit or a guarantee changes, we need to recalculate guarantees and ensure that the change will not break any guarantees
3. The same thing as stated above, when a resource group is created or deleted

This can lead to some instability; a change in one group propagates to all other groups.

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Mon, 18 Sep 2006 08:56:14 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:

[snip]

> This approach has the following disadvantages
> 1. Lets consider initialization - When we create 'n' groups
> initially, we need
>   to spend O(n^2) time to assign guarantees.

1. Not guarantees - limits. If you do not need guarantees - assign
   overcommited limits. Most of OpenVZ users do so and nobody claims.
2. If you start n groups at once then limits are calculated in O(n)
   time, not O(n^2).

> 2. Every time a limit or a guarantee changes, we need to recalculate
> guarantees
>   and ensure that the change will not break any guarantees

The same.

> 3. The same thing as stated above, when a resource group is created
> or deleted
>
> This can lead to some instability; a change in one group propagates to
> all other groups.

Let me cite a part of your answer on my letter from 11.09.2006:
"...
  xemul> I have a node with 1Gb of ram and 10 containers with 100Mb
  xemul> guarantee each. I want to start one more.
  xemul> What shall I do not to break guarantees?

 Don't start the new container or change the guarantees of the
 existing ones to accommodate this one ... It would be perfectly
 ok to have a container that does not care about guarantees to
 set their guarantee to 0 and set their limit to the desired value
..."

The same for the limiting - either do not start new container, or
recalculate limits to meet new requirements. You may not take care of
guarantees as weel and create an overcommited configuration.

And one more thing. We've asked it many times and I ask it again -
please, show us the other way for providing guarantee rather than

limiting or reserving.

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Balbir Singh on Mon, 18 Sep 2006 11:20:51 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Balbir Singh wrote:
>
> [snip]
>
>> This approach has the following disadvantages
>> 1. Lets consider initialization - When we create 'n' groups
>> initially, we need
>>   to spend O(n^2) time to assign guarantees.
>
> 1. Not guarantees - limits. If you do not need guarantees - assign
>   overcommited limits. Most of OpenVZ users do so and nobody claims.
> 2. If you start n groups at once then limits are calculated in O(n)
>   time, not O(n^2).

Yes.. if you start them at once, but if they are incrementally
added and started it is O(n^2)

>
>> 2. Every time a limit or a guarantee changes, we need to recalculate
>> guarantees
>>   and ensure that the change will not break any guarantees
>
> The same.
>
>> 3. The same thing as stated above, when a resource group is created
>> or deleted
>>
>> This can lead to some instability; a change in one group propagates to
>> all other groups.
>
> Let me cite a part of your answer on my letter from 11.09.2006:
> "...
>   xemul> I have a node with 1Gb of ram and 10 containers with 100Mb
>   xemul> guarantee each. I want to start one more.
>   xemul> What shall I do not to break guarantees?
>
> Don't start the new container or change the guarantees of the
> existing ones to accommodate this one ... It would be perfectly
> ok to have a container that does not care about guarantees to

> set their guarantee to 0 and set their limit to the desired value
> ..."
>
> The same for the limiting - either do not start new container, or
> recalculate limits to meet new requirements. You may not take care of
> guarantees as weel and create an overcommited configuration.
>
> And one more thing. We've asked it many times and I ask it again -
> please, show us the other way for providing guarantee rather than
> limiting or reserving.

There are some other options, I am sure Chandra will probably have
more.

1. Reclaim resources from other containers. This can be done well for
   user-pages, if we ensure that each container does not mlock more
   than its guaranteed share of memory.
2. Provide best effort guarantees for non-reclaimable memory
3. oom-kill a container or a task within a resource group that has
   exceeded its guarantee and some other container is unable to meet its
   guarantee

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Balbir Singh on Mon, 18 Sep 2006 11:27:07 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> Kirill Korotaev wrote:
>
> [snip]
>>> I have a C program that computes limits to obtain desired guarantees
>>> in a single 'for (i = 0; i < n; n++)' loop for any given set of guarantees.
>>> With all error handling, beautifull output, nice formatting etc it weights
>>> only 60 lines.
>
> Look at http://wiki.openvz.org/Containers/Guarantees_for_resources
> I've described there how a guarantee can be get with limiting in details.
>
> [snip]
>

>>> I do not 'do not like guarantee'. I'm just sure that there are two ways
>>> for providing guarantee (for unreclaimable resorces):
>>> 1. reserving resource for group in advance
>>> 2. limit resource for others
>>> Reserving is worse as it is essentially limiting (you cut off 100Mb from
>>> 1Gb RAM thus limiting the other groups by 900Mb RAM), but this limiting
>>> is too strict - you _have_ to reserve less than RAM size. Limiting in
>>> run-time is more flexible (you may create an overcommited BC if you
>>> want to) and leads to the same result - guarantee.
>> I think this deserves putting on Wiki.
>> It is very good clear point.
>
> This is also on the page I gave link at.

The program (calculate_limits()) listed on the website does not work for
the following case

N=2;
R=100;
g[2] = {30, 30};

The output is -10 and -10 for the limits

For

N=3;
R=100;
g[3] = {30, 30, 10};

I get -70, -70 and -110 as the limits

Am I interpreting the parameters correctly? Or the program is broken?

--

Balbir Singh,
Linux Technology Center,
IBM Software Labs

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Mon, 18 Sep 2006 11:32:32 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:

> Pavel Emelianov wrote:
>> Balbir Singh wrote:
>>
>> [snip]
>>
>>> This approach has the following disadvantages
>>> 1. Lets consider initialization - When we create 'n' groups
>>> initially, we need
>>>   to spend O(n^2) time to assign guarantees.
>>
>> 1. Not guarantees - limits. If you do not need guarantees - assign
>>   overcommited limits. Most of OpenVZ users do so and nobody claims.
>> 2. If you start n groups at once then limits are calculated in O(n)
>>   time, not O(n^2).
>
> Yes.. if you start them at once, but if they are incrementally
> added and started it is O(n^2)

See my comment below.

>
>>
>>> 2. Every time a limit or a guarantee changes, we need to recalculate
>>> guarantees
>>>   and ensure that the change will not break any guarantees
>>
>> The same.
>>
>>> 3. The same thing as stated above, when a resource group is created
>>> or deleted
>>>
>>> This can lead to some instability; a change in one group propagates to
>>> all other groups.
>>
>> Let me cite a part of your answer on my letter from 11.09.2006:
>> "...
>>   xemul> I have a node with 1Gb of ram and 10 containers with 100Mb
>>   xemul> guarantee each. I want to start one more.
>>   xemul> What shall I do not to break guarantees?
>>
>> Don't start the new container or change the guarantees of the
>> existing ones to accommodate this one ... It would be perfectly
>> ok to have a container that does not care about guarantees to
>> set their guarantee to 0 and set their limit to the desired value
>> ..."
>>
>> The same for the limiting - either do not start new container, or
>> recalculate limits to meet new requirements. You may not take care of

>> guarantees as weel and create an overcommited configuration.

As I do not see any reply on this I consider "O(n^2) disadvantage" to
be irrelevant.

>>
>> And one more thing. We've asked it many times and I ask it again -
>> please, show us the other way for providing guarantee rather than
>> limiting or reserving.
>
> There are some other options, I am sure Chandra will probably have
> more.
>
> 1. Reclaim resources from other containers. This can be done well for
>   user-pages, if we ensure that each container does not mlock more
>   than its guaranteed share of memory.

We've already agreed to consider unreclaimable resources only.
If we provide reclaimable memory *only* then we can provide any
guarantee with a single page available for user-space.
Unreclaimable resource is the most interesting one.

> 2. Provide best effort guarantees for non-reclaimable memory

That's the question - how?

> 3. oom-kill a container or a task within a resource group that has
>   exceeded its guarantee and some other container is unable to meet its
>   guarantee

Oom-killer must start only when there are no other ways to find memory.
This must be a "last argument", not the regular solution.

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Mon, 18 Sep 2006 12:37:59 GMT
View Forum Message <> Reply to Message

Balbir Singh wrote:

[snip]
>
> The program (calculate_limits()) listed on the website does not work for
> the following case
>
> N=2;
> R=100;

> g[2] = {30, 30};
>
>
> The output is -10 and -10 for the limits
>
> For
>
> N=3;
> R=100;
> g[3] = {30, 30, 10};
>
> I get -70, -70 and -110 as the limits
>
> Am I interpreting the parameters correctly? Or the program is broken?
>

Program on site is broken. Thanks for noticing:

```
$ gcc guar.c -o guar
$ ./guar 30 30
guar lim
  30  70 ( 70/1)
  30  70 ( 70/1)
$ ./guar 30 30 10
guar lim
  30  45 ( 90/2)
  30  45 ( 90/2)
  10  25 ( 50/2)
```

To stop future "errors" remember that this is a simplified program that
considers guarantees to be <= 100%, sum of guarantees to be <= 100% etc.

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Mon, 18 Sep 2006 23:48:42 GMT
View Forum Message <> Reply to Message

On Fri, 2006-09-15 at 11:15 +0400, Pavel Emelianov wrote:
> Chandra Seetharaman wrote:
> > On Thu, 2006-09-14 at 11:53 +0400, Pavel Emelianov wrote:
> >
> > <snip>
> >
> >
> >>> What if I have 40 containers each with 2% guarantee ? what do we do
> >>> then ? and many other different combinations (what I gave was not the

> >>> _only_ scenario).
> >>>
> >>>
> >> Then you need to solve a set of 40 equations. This sounds weird, but
> >> don't afraid - sets like these are solved lightly.
> >>
> >
> > extrapolate that to a varying # of permutations and real time changes in
> > the system workload. Won't it be complex ?
> >
> I have a C program that computes limits to obtain desired guarantees
> in a single 'for (i = 0; i < n; n++)' loop for any given set of guarantees.
> With all error handling, beautifull output, nice formatting etc it weights
> only 60 lines.
> > Wouldn't it be a lot simpler if we have the guarantee support instead ?
> > Why you do not like guarantee ? :)
> >
> I do not 'do not like guarantee'. I'm just sure that there are two ways
> for providing guarantee (for unreclaimable resorces):
> 1. reserving resource for group in advance
> 2. limit resource for others
> Reserving is worse as it is essentially limiting (you cut off 100Mb from
> 1Gb RAM thus limiting the other groups by 900Mb RAM), but this limiting
> is too strict - you _have_ to reserve less than RAM size. Limiting in
> run-time is more flexible (you may create an overcommited BC if you
> want to) and leads to the same result - guarantee.

I do not agree with, "it will limit the efficient usage of resource,
hence lets not provide the feature".

We should provide the feature to the user and the user decide how they
want the resources to be used.

If they decide to use guarantees, they do know what is the cost.

> > <snip>
> >
> [snip]
>
> ---------- ----------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Mon, 18 Sep 2006 23:51:35 GMT
View Forum Message <> Reply to Message

On Fri, 2006-09-15 at 12:49 +0400, Kirill Korotaev wrote:
> > <snip>
> >
> >>Reserving in advance means that sometimes you won't be able to start a
> >>new group without taking back some of reserved pages. This is ... strange.
> >
> >
> > I do not see it strange. At the time of creation, user sees the failure
> > (that there isn't enough resource to provide the required/requested
> > guarantee) and can act accordingly.
> >
> > BTW, VMware does it this way.
> This is not true at least for ESX server.

Hmm, from what I have seen, in ESX server, creation of a VM will fail,
if the specified guarantees cannot meet at the time of creation.

> It overcommits memory and does dirty tricks like balooning to free memory then.

This is how they handle over commit, which is not what I was talking
about.

<snip>

--

Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Mon, 18 Sep 2006 23:57:44 GMT

On Fri, 2006-09-15 at 12:57 +0400, Kirill Korotaev wrote:
> > CKRM/RG handles it this way:
> >
> > Amount of a resource a child RG gets is the ratio of its share value to
> > the parent's total # of shares. Children's resource allocation can be
> > changed just by changing the parent's total # of shares.
> >
> > If you case about initial situation would be:
> > Total memory in the system 100MB
> > parent's total # of shares: 100 (1 share == 1MB)
> > 10 children with # of shares: 10 (i.e each children has 10MB)
> >
> > When I want to add another child, just change parent's total # of shares
> > to be say 125:
> > Total memory in the system 100MB
> > parent's total # of shares: 125 (1 share == 0.8MB)
> > 10 children with # of shares: 10 (i.e each children has 8MB)
> > Now you are left with 25 shares (or 20MB) that you can assign to new
> > child(ren) as you please.
>
> setting memory in "shares" doesn't look user friendly at all...

in RG, the user can set the root level shares to be the "total # of
pages", and then the shares will simply reflect the number of pages.

>
> Kirill
>
>
>
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user

memory)
Posted by Chandra Seetharaman on Tue, 19 Sep 2006 00:05:59 GMT
View Forum Message <> Reply to Message

On Mon, 2006-09-18 at 12:56 +0400, Pavel Emelianov wrote:

<snip>

> The same for the limiting - either do not start new container, or
> recalculate limits to meet new requirements. You may not take care of
> guarantees as weel and create an overcommited configuration.
>
> And one more thing. We've asked it many times and I ask it again -
> please, show us the other way for providing guarantee rather than
> limiting or reserving.

Why do we want the capability to be snipped at the infrastructure level.
Let the controller writers decide how they want to provide the
capability and the users to decide if they want to use the feature at a
price.

>
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Chandra Seetharaman on Tue, 19 Sep 2006 00:08:20 GMT
View Forum Message <> Reply to Message

On Mon, 2006-09-18 at 16:37 +0400, Pavel Emelianov wrote:
> Balbir Singh wrote:
>
> [snip]
> >
> > The program (calculate_limits()) listed on the website does not work for

> > the following case
> >
> > N=2;
> > R=100;
> > g[2] = {30, 30};
> >
> >
> > The output is -10 and -10 for the limits
> >
> > For
> >
> > N=3;
> > R=100;
> > g[3] = {30, 30, 10};
> >
> > I get -70, -70 and -110 as the limits
> >
> > Am I interpreting the parameters correctly? Or the program is broken?
> >
>
> Program on site is broken. Thanks for noticing:
>
> $ gcc guar.c -o guar
> $ ./guar 30 30
> guar lim
>   30 70 (70/1)
>   30 70 (70/1)
> $ ./guar 30 30 10
> guar lim
>   30 45 (90/2)
>   30 45 (90/2)
>   10 25 (50/2)

I am confused. Are you changing the parameters on how the user want the
groups to be controlled.

They want the resource usage to be between 30 and 70, but you change it
to be 30-45.

>
>
> To stop future "errors" remember that this is a simplified program that
> considers guarantees to be <= 100%, sum of guarantees to be <= 100% etc.
>
> -- -------------
> Using Tomcat but need to do more? Need to support web services, security?
> Get stuff done quickly with pre-integrated technology to make your job easier
> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo

> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
> _____
> ckrm-tech mailing list
> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
--

---------- ----------
Chandra Seetharaman | Be careful what you choose....
- sekharan@us.ibm.com | .....you may get it.
---------- ----------

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Tue, 19 Sep 2006 08:04:50 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:
> On Mon, 2006-09-18 at 12:56 +0400, Pavel Emelianov wrote:
>
> <snip>
>
>
>> The same for the limiting - either do not start new container, or
>> recalculate limits to meet new requirements. You may not take care of
>> guarantees as weel and create an overcommited configuration.
>>
>> And one more thing. We've asked it many times and I ask it again -
>> please, show us the other way for providing guarantee rather than
>> limiting or reserving.
>>
>
> Why do we want the capability to be snipped at the infrastructure level.
> Let the controller writers decide how they want to provide the
> capability and the users to decide if they want to use the feature at a
> price.
>

That's what we proposed in the very beginning - to review an infrastructure
with minimal functionality (limiting) and develop new features after the
"core"
is accepted.

I'm glad that we've finaly made a bargain :)

>
>> ---------- ----------
>> Using Tomcat but need to do more? Need to support web services, security?

>> Get stuff done quickly with pre-integrated technology to make your job easier
>> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
>> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
>> _____
>> ckrm-tech mailing list
>> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
>>

Subject: Re: [ckrm-tech] [PATCH] BC: resource beancounters (v4) (added user memory)
Posted by Pavel Emelianov on Tue, 19 Sep 2006 08:06:39 GMT
View Forum Message <> Reply to Message

Chandra Seetharaman wrote:
> On Mon, 2006-09-18 at 16:37 +0400, Pavel Emelianov wrote:
>
>> Balbir Singh wrote:
>>
>> [snip]
>>
>>> The program (calculate_limits()) listed on the website does not work for
>>> the following case
>>>
>>> N=2;
>>> R=100;
>>> g[2] = {30, 30};
>>>
>>>
>>> The output is -10 and -10 for the limits
>>>
>>> For
>>>
>>> N=3;
>>> R=100;
>>> g[3] = {30, 30, 10};
>>>
>>> I get -70, -70 and -110 as the limits
>>>
>>> Am I interpreting the parameters correctly? Or the program is broken?
>>>
>>>
>> Program on site is broken. Thanks for noticing:
>>
>> $ gcc guar.c -o guar
>> $ ./guar 30 30
>> guar lim
>> 30 70 (70/1)

>>   30 70 (70/1)
>> $ ./guar 30 30 10
>> guar lim
>>   30 45 (90/2)
>>   30 45 (90/2)
>>   10 25 (50/2)
>>
>
> I am confused. Are you changing the parameters on how the user want the
> groups to be controlled.
>

Nope. I just calculate some auxiliary values to acheive the goal.

> They want the resource usage to be between 30 and 70, but you change it
> to be 30-45.
>

User wants group to consume _at_least_ 30%. I do provide it, but do not
prevent it from consuming more.

>
>> To stop future "errors" remember that this is a simplified program that
>> considers guarantees to be <= 100%, sum of guarantees to be <= 100% etc.
>>
>> ---------- ----------
>> Using Tomcat but need to do more? Need to support web services, security?
>> Get stuff done quickly with pre-integrated technology to make your job easier
>> Download IBM WebSphere Application Server v.1.0.1 based on Apache Geronimo
>> http://sel.as-us.falkag.net/sel?cmd=lnk&kid=120709&b id=263057&dat=121642
>> _____
>> ckrm-tech mailing list
>> https://lists.sourceforge.net/lists/listinfo/ckrm-tech
>>