
Subject: V2: Add tgid aggregation to beancounters (was Re: [ckrm-tech] [PATCH]
BC: resource beancounters (v4)
Posted by [Balbir Singh](#) on Mon, 11 Sep 2006 21:24:19 GMT
[View Forum Message](#) <> [Reply to Message](#)

Balbir Singh wrote:

> On Wed, Sep 06, 2006 at 05:06:44PM +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.

>>

>> 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
>
> Hi, Kirill,
>
> I've got a patch to extend bean-counters to do simple aggregation.
>
> The idea is to use this to finally do task migration. Initial comments on the
> design and idea would be useful. The original idea was suggested by Dave
> Hansen during a private discussion.
>
> TODOS:
>
> 1. Add task data extraction support
> 2. Add task migration support
>
> I've gotten the patch to compile and boot on a x86-64 box.
>
```

Attempt to create per-tgid beancounters. These are especially useful for memory (since all threads share memory) and they also aid task/tgid migration.

TODO's

1. Add support for unused_pages (so that accounting is accurate and consistent with beancounter accounting principles).
2. Add system call support to extract tgid information
3. Consider refactoring the code

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

```
include/bc/beancounter.h | 62 +++++
kernel/bc/beancounter.c | 166 +++++
kernel/bc/misc.c        | 5 +
kernel/fork.c           | 4 -
4 files changed, 230 insertions(+), 7 deletions(-)
```

```
diff -puN include/bc/beancounter.h~per-tgid-resource-tracking
include/bc/beancounter.h
--- linux-2.6.18-rc5/include/bc/beancounter.h~per-tgid-resource-tracking
2006-09-08 12:03:31.000000000 +0530
+++ linux-2.6.18-rc5-balbir/include/bc/beancounter.h 2006-09-12
02:22:03.000000000 +0530
@@ -42,7 +42,10 @@ struct bc_resource_parm {
#include <linux/list.h>
```

```

#include <asm/atomic.h>

#define BC_MAXVALUE LONG_MAX
#define BC_MAXVALUE LONG_MAX
+
#define BC_TGID_HASH_BITS 6
#define BC_TGID_HASH_SIZE (1 << BC_TGID_HASH_BITS)

/*
 * This magic is used to distinguish user beancounter and pages beancounter
@@ -73,6 +76,18 @@ struct beancounter {
#endif
/* resources statistics and settings */
struct bc_resource_parm bc_parms[BC_RESOURCES];
+ struct hlist_head tgid_hash[BC_TGID_HASH_SIZE];
+};
+
+/*
+ * Per tgid resource statistics
+ */
+struct tgid_beancounter {
+ struct bc_resource_parm tbc_parms[BC_RESOURCES];
+ struct hlist_node hash;
+ pid_t tgid;
+ struct beancounter *bc;
+ atomic_t tbc_refcount;
+};

enum bc_severity { BC_BARRIER, BC_LIMIT, BC_FORCE };
@@ -101,6 +116,16 @@ static inline void bc_adjust_maxheld(str
    parm->maxheld = parm->held;
}

+static inline void tbc_adjust_maxheld(struct tgid_beancounter *tbc,
+ int resource)
+{
+ struct bc_resource_parm *parm;
+
+ parm = &tbc->tbc_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;
@@ -110,6 +135,16 @@ static inline void bc_adjust_minheld(str
  parm->minheld = parm->held;

```

```

}

+static inline void tbc_adjust_minheld(struct tgid_beancounter *tbc,
+  int resource)
+{
+ struct bc_resource_parm *parm;
+
+ parm = &tbc->tbc_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,
@@ -119,6 +154,11 @@ void bc_uncharge_locked(struct beancount
+ void bc_uncharge(struct beancounter *bc, int res, unsigned long val);

+ struct beancounter *beancounter_findcreate(bcid_t id, int mask);
+struct tgid_beancounter *tgid_beancounter_findcreate(
+  struct beancounter *bc,
+  int mask,
+  int locked);
+void tgid_beancounter_release(struct tgid_beancounter *tbc, int locked);

+ static inline struct beancounter *get_beancounter(struct beancounter *bc)
+ {
@@ -126,7 +166,15 @@ static inline struct beancounter *get_be
+   return bc;
+ }

+static inline struct tgid_beancounter *tgid_get_beancounter(
+  struct tgid_beancounter *tbc)
+{
+ atomic_inc(&tbc->tbc_refcount);
+ return tbc;
+}
+
+ void put_beancounter(struct beancounter *bc);
+void tgid_put_beancounter(struct tgid_beancounter *tbc);

+ void bc_init_early(void);
+ void bc_init_late(void);
@@ -135,6 +183,18 @@ void bc_init_proc(void);
+ extern struct beancounter init_bc;
+ extern const char *bc_rnames[];

+ #define tgid_beancounter_findcreate_locked(bc, mask) \

```

```

+ tgid_beancounter_findcreate(bc, mask, 1)
+
+ #define tgid_beancounter_findcreate_unlocked(bc, mask) \
+ tgid_beancounter_findcreate(bc, mask, 0)
+
+ #define tgid_beancounter_release_locked(bc) \
+ tgid_beancounter_release(bc, 1)
+
+ #define tgid_beancounter_release_unlocked(bc) \
+ tgid_beancounter_release(bc, 0)
+
+ #else /* CONFIG_BEANCOUNTERS */

+ #define nr_beancounters 0
diff -puN kernel/bc/beancounter.c~per-tgid-resource-tracking kernel/bc/beancounter.c
--- linux-2.6.18-rc5/kernel/bc/beancounter.c~per-tgid-resource-tracking
2006-09-08 12:03:31.000000000 +0530
+++ linux-2.6.18-rc5-balbir/kernel/bc/beancounter.c 2006-09-12
02:45:53.000000000 +0530
@@ -14,9 +14,13 @@
+ #include <bc/vmrss.h>

+ static kmem_cache_t *bc_cache;
+ static kmem_cache_t *bc_tgid_cache;
+ static struct beancounter default_beancounter;
+ static struct tgid_beancounter default_tgid_beancounter;

+ static void init_beancounter_struct(struct beancounter *bc, bcid_t id);
+ static void init_tgid_beancounter_struct(struct tgid_beancounter *tbc,
+ struct beancounter *bc);

+ struct beancounter init_bc;

@@ -34,6 +38,7 @@ const char *bc_rnames[] = {
+ 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))
+ #define bc_tgid_hash_fn(bcid) (hash_long(bcid, BC_TGID_HASH_BITS))

+ /*
+ * Per resource beancounting. Resources are tied to their bc id.
@@ -97,6 +102,103 @@ out:
+ return new_bc;
+ }

+ /*
+ * Introduce a hierarchy for beancounters.
+ * bc

```

```

+ * tbc tbc ... tbc tbc
+ * Each tgid_beancounter tracks the resource usage for the tgid.
+ * It makes it easier to move tasks across beancounters, since we know
+ * the usage of every tgid. It's quite easy to extend this detail to a
+ * per-task level, by creating task_beancounters under each tgid_beancounter.
+ */
+struct tgid_beancounter *tgid_beancounter_findcreate(struct beancounter *bc,
+    int mask, int locked)
+{
+ struct tgid_beancounter *new_tbc, *tbc = NULL;
+ unsigned long flags = 0; /* use a macro to hide if reqd */
+ struct hlist_head *slot = NULL;
+ struct hlist_node *pos = NULL;
+
+ get_beancounter(bc);
+ slot = &bc->tgid_hash[bc_tgid_hash_fn(current->tgid)];
+ new_tbc = NULL;
+
+retry:
+ if (!locked)
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ hlist_for_each_entry (tbc, pos, slot, hash)
+ if (tbc->tgid == current->tgid)
+ break;
+
+ if (pos != NULL) {
+ if (!(mask & BC_ALLOC))
+ put_beancounter(bc);
+ if (mask & BC_ALLOC)
+ tgid_get_beancounter(tbc);
+ if (!locked)
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+
+ if (new_tbc != NULL)
+ kmem_cache_free(bc_tgid_cache, new_tbc);
+ return tbc;
+ }
+
+ if (new_tbc != NULL)
+ goto out_install;
+
+ if (!locked)
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+
+ if (!(mask & BC_ALLOC))
+ goto out;
+
+ new_tbc = kmem_cache_alloc(bc_tgid_cache,

```

```

+ mask & BC_ALLOC_ATOMIC ? GFP_ATOMIC : GFP_KERNEL);
+ if (new_tbc == NULL)
+ goto out;
+
+ init_tgid_beancounter_struct(new_tbc, bc);
+ goto retry;
+
+out_install:
+ hlist_add_head(&new_tbc->hash, slot);
+ if (!locked)
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+out:
+ if (!(mask & BC_ALLOC))
+ put_beancounter(bc);
+ if (new_tbc == NULL) {
+ new_tbc = &default_tgid_beancounter;
+ }
+ return new_tbc;
+}
+
+void tgid_put_beancounter(struct tgid_beancounter *tbc)
+{
+ int i;
+ unsigned long flags = 0;
+ struct beancounter *bc = tbc->bc;
+
+ if (tbc == &default_tgid_beancounter) {
+ return;
+ }
+
+ put_beancounter(bc);
+ if (!atomic_dec_and_lock_irqsave(&tbc->tbc_refcount, &bc->bc_lock,
+ flags))
+ return;
+
+ for (i = 0; i < BC_RESOURCES; i++)
+ if (tbc->tbc_parms[i].held != 0)
+ printk("BC: %d has %lu of %s held on put\n", tbc->tgid,
+ tbc->tbc_parms[i].held, bc_rnames[i]);
+
+ hlist_del(&tbc->hash);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+
+ kmem_cache_free(bc_tgid_cachep, tbc);
+}
+
+void put_beancounter(struct beancounter *bc)
+{

```

```

int i;
@@ -110,15 +212,15 @@ void put_beancounter(struct beancounter

    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,
+       printk("BC: %d has %lu of %s held on put\n", 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);
+       printk("BC: %d has %lu of unused pages held on put\n",
+       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,
+       printk("BC: %d hash %llu of rss pages held on put\n", bc->bc_id,
            bc->rss_pages);
#endif
    hlist_del(&bc->hash);
@@ -139,12 +241,22 @@ int bc_charge_locked(struct beancounter
    enum bc_severity strict)
    {
        unsigned long new_held;
+       unsigned long tgid_new_held;
+       struct tgid_beancounter *tbc;
+
+       tbc = tgid_beancounter_findcreate_locked(bc, BC_LOOKUP);
+       if (!tbc) {
+           printk(KERN_WARNING "Missing tgid beancounter for bc %d tgid "
+           "%d\n", bc->bc_id, current->tgid);
+           return 0;
+       }

        /*
         * 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;
+       tgid_new_held = tbc->tbc_parms[resource].held + val;

        switch (strict) {
        case BC_BARRIER:
@@ -160,6 +272,7 @@ int bc_charge_locked(struct beancounter
        case BC_FORCE:
            bc->bc_parms[resource].held = new_held;
            bc_adjust_maxheld(bc, resource);

```

```

+ tbc_adjust_maxheld(tbc, resource);
return 0;

default:
@@ -167,6 +280,7 @@ int bc_charge_locked(struct beancounter
}

bc->bc_parms[resource].failcnt++;
+ tbc->tbc_parms[resource].failcnt++;
return -ENOMEM;
}
EXPORT_SYMBOL_GPL(bc_charge_locked);
@@ -189,6 +303,25 @@ 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)
{
+ struct tgid_beancounter *tbc;
+ unsigned long val2 = val;
+
+ tbc = tgid_beancounter_findcreate_locked(bc, BC_LOOKUP);
+ if (!tbc) {
+ printk(KERN_WARNING "Missing tgid beancounter for bc %d tgid "
+ "%d\n", bc->bc_id, current->tgid);
+ return;
+ }
+
+ if (unlikely(tbc->tbc_parms[resource].held < val2)) {
+ printk("BC: overuncharging bc %d %s: val %lu, holds %lu\n",
+ tbc->tgid, bc_rnames[resource], val2,
+ tbc->tbc_parms[resource].held);
+ val2 = tbc->tbc_parms[resource].held;
+ }
+ tbc->tbc_parms[resource].held -= val;
+ tbc_adjust_minheld(tbc, resource);
+
+ 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,
@@ -199,6 +332,7 @@ void bc_uncharge_locked(struct beancount
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)
@@ -227,12 +361,31 @@ EXPORT_SYMBOL_GPL(bc_uncharge);

```

```

static void init_beancounter_struct(struct beancounter *bc, bcid_t id)
{
+ int i;
+
    bc->bc_magic = BC_MAGIC;
    atomic_set(&bc->bc_refcount, 1);
    spin_lock_init(&bc->bc_lock);
    bc->bc_id = id;
+ for (i = 0; i < BC_TGID_HASH_SIZE; i++)
+ INIT_HLIST_HEAD(&bc->tgid_hash[i]);
}

+static void init_tgid_beancounter_struct(struct tgid_beancounter *tbc,
+ struct beancounter *bc)
+{
+ int k;
+
+ INIT_HLIST_NODE(&tbc->hash);
+ atomic_set(&tbc->tbc_refcount, 1);
+ tbc->bc = bc;
+ tbc->tgid = current->tgid;
+ for (k = 0; k < BC_RESOURCES; k++) {
+ tbc->tbc_parms[k].limit = BC_MAXVALUE;
+ tbc->tbc_parms[k].barrier = BC_MAXVALUE;
+ tbc->tbc_parms[k].held = 0;
+ }
+}

static void init_beancounter_nolimits(struct beancounter *bc)
{
    int k;
@@ -281,7 +434,12 @@ void __init bc_init_late(void)
    sizeof(struct beancounter), 0,
    SLAB_HWCACHE_ALIGN | SLAB_PANIC, NULL, NULL);

+ bc_tgid_cachep = kmem_cache_create("tgid_beancounters",
+ sizeof(struct tgid_beancounter), 0,
+ SLAB_HWCACHE_ALIGN | SLAB_PANIC, NULL, NULL);
+
    bc = &default_beancounter;
    init_beancounter_syslimits(bc);
    init_beancounter_struct(bc, 0);
+ init_tgid_beancounter_struct(&default_tgid_beancounter, bc);
}
diff -puN kernel/bc/misc.c~per-tgid-resource-tracking kernel/bc/misc.c
--- linux-2.6.18-rc5/kernel/bc/misc.c~per-tgid-resource-tracking 2006-09-08
14:18:55.000000000 +0530
+++ linux-2.6.18-rc5-balbir/kernel/bc/misc.c 2006-09-12 02:15:07.000000000 +0530
@@ -22,10 +22,15 @@ void bc_task_charge(struct task_struct *

```

```

    bc = old_bc->fork_bc;
    new_bc->exec_bc = get_beancounter(bc);
    new_bc->fork_bc = get_beancounter(bc);
+ tgid_beancounter_findcreate_unlocked(bc, BC_ALLOC);
}

void bc_task_uncharge(struct task_struct *tsk)
{
+ struct tgid_beancounter *tbc;
+ tbc = tgid_beancounter_findcreate_unlocked(tsk->task_bc.exec_bc,
+     BC_LOOKUP);
    put_beancounter(tsk->task_bc.exec_bc);
    put_beancounter(tsk->task_bc.fork_bc);
+ tgid_put_beancounter(tbc);
}
diff -puN kernel/fork.c~per-tgid-resource-tracking kernel/fork.c
--- linux-2.6.18-rc5/kernel/fork.c~per-tgid-resource-tracking 2006-09-11
23:53:11.000000000 +0530
+++ linux-2.6.18-rc5-balbir/kernel/fork.c 2006-09-12 02:04:49.000000000 +0530
@@ -994,8 +994,6 @@ 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);
@@ -1106,6 +1104,8 @@ static struct task_struct *copy_process(
    if (clone_flags & CLONE_THREAD)
        p->tgid = current->tgid;

+ bc_task_charge(current, p);
+
    if ((retval = security_task_alloc(p)))
        goto bad_fork_cleanup_policy;
    if ((retval = audit_alloc(p)))

```

Balbir Singh,
 Linux Technology Center,
 IBM Software Labs
