Subject: Re: [PATCH 2/3] i/o bandwidth controller infrastructure
Posted by akpm on Thu, 26 Jun 2008 00:29:00 GMT
View Forum Message <> Reply to Message

On Fri, 20 Jun 2008 12:05:34 +0200
Andrea Righi <righi.andrea@gmail.com> wrote:

> This is the core io-throttle kernel infrastructure. It creates the basic
> interfaces to cgroups and implements the I/O measurement and throttling
> functions.
>
> Signed-off-by: Andrea Righi <righi.andrea@gmail.com>
> ---
> block/Makefile | 2 +
> block/blk-io-throttle.c | 393 ++
> include/linux/blk-io-throttle.h | 12 ++
> include/linux/cgroup_subsys.h | 6 +
> init/Kconfig | 10 +
> 5 files changed, 423 insertions(+), 0 deletions(-)
> create mode 100644 block/blk-io-throttle.c
> create mode 100644 include/linux/blk-io-throttle.h
>
> diff --git a/block/Makefile b/block/Makefile
> index 5a43c7d..8dec69b 100644
> --- a/block/Makefile
> +++ b/block/Makefile
> @@ -14,3 +14,5 @@ obj-$(CONFIG_IOSCHED_CFQ) += cfq-iosched.o
>
> obj-$(CONFIG_BLK_DEV_IO_TRACE) += blktrace.o
> obj-$(CONFIG_BLOCK_COMPAT) += compat_ioctl.o
> +
> +obj-$(CONFIG_CGROUP_IO_THROTTLE) += blk-io-throttle.o
> diff --git a/block/blk-io-throttle.c b/block/blk-io-throttle.c
> new file mode 100644
> index 0000000..4ec02bb
> --- /dev/null
> +++ b/block/blk-io-throttle.c
> @@ -0,0 +1,393 @@
> +/*
> + * blk-io-throttle.c
> + *
> + * This program is free software; you can redistribute it and/or
> + * modify it under the terms of the GNU General Public
> + * License as published by the Free Software Foundation; either
> + * version 2 of the License, or (at your option) any later version.
> + *
> + * This program is distributed in the hope that it will be useful,
> + * but WITHOUT ANY WARRANTY; without even the implied warranty of

> + * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
> + * General Public License for more details.
> + *
> + * You should have received a copy of the GNU General Public
> + * License along with this program; if not, write to the
> + * Free Software Foundation, Inc., 59 Temple Place - Suite 330,
> + * Boston, MA 021110-1307, USA.
> + *
> + * Copyright (C) 2008 Andrea Righi <righi.andrea@gmail.com>
> + */
> +
> +#include <linux/init.h>
> +#include <linux/module.h>
> +#include <linux/cgroup.h>
> +#include <linux/slab.h>
> +#include <linux/gfp.h>
> +#include <linux/err.h>
> +#include <linux/sched.h>
> +#include <linux/fs.h>
> +#include <linux/jiffies.h>
> +#include <linux/hardirq.h>
> +#include <linux/list.h>
> +#include <linux/spinlock.h>
> +#include <linux/uaccess.h>
> +#include <linux/vmalloc.h>
> +#include <linux/blk-io-throttle.h>
> +
> +#define ONE_SEC 1000000L /* # of microseconds in a second */

Remove this, use USEC_PER_SEC throughout.

> +#define KBS(x) ((x) * ONE_SEC >> 10)

Convert to lower-case-named C function, please.

> +
> +struct iothrottle_node {
> + struct list_head node;
> + dev_t dev;
> + unsigned long iorate;
> + unsigned long timestamp;
> + atomic_long_t stat;
> +};

Please document each field in structures. This is usually more useful
and important than documenting the code whcih manipulates those fields.

It is important that the units of fields such as iorate and timestamp

and stamp be documented.

> +struct iothrottle {
> + struct cgroup_subsys_state css;
> + /* protects the list below, not the single elements */
> + spinlock_t lock;
> + struct list_head list;
> +};

Looking elsewhere in the code it appears that some RCU-based locking is performed. That should be documented somewhere. Fully. At the definition site of the data whcih is RCU-protceted would be a good site.

> +static inline struct iothrottle *cgroup_to_iothrottle(struct cgroup *cont)
> +{
> + return container_of(cgroup_subsys_state(cont, iothrottle_subsys_id),
> + struct iothrottle, css);
> +}
> +
> +static inline struct iothrottle *task_to_iothrottle(struct task_struct *task)
> +{
> + return container_of(task_subsys_state(task, iothrottle_subsys_id),
> + struct iothrottle, css);
> +}
> +
> +static inline struct iothrottle_node *iothrottle_search_node(
> + const struct iothrottle *iot,
> + dev_t dev)
> +{
> + struct iothrottle_node *n;
> +
> + list_for_each_entry_rcu(n, &iot->list, node)
> + if (n->dev == dev)
> + return n;
> + return NULL;
> +}

This will be too large for inlining.

This function presumably has caller-provided locking requirements? They should be documented here.

> +static inline void iothrottle_insert_node(struct iothrottle *iot,
> + struct iothrottle_node *n)
> +{
> + list_add_rcu(&n->node, &iot->list);

> +}
> +
> +static inline struct iothrottle_node *iothrottle_replace_node(
> +   struct iothrottle *iot,
> +   struct iothrottle_node *old,
> +   struct iothrottle_node *new)
> +{
> + list_replace_rcu(&old->node, &new->node);
> + return old;
> +}

Dittoes.

> +static inline struct iothrottle_node *iothrottle_delete_node(
> +   struct iothrottle *iot,
> +   dev_t dev)
> +{
> + struct iothrottle_node *n;
> +
> + list_for_each_entry(n, &iot->list, node)
> + if (n->dev == dev) {
> +   list_del_rcu(&n->node);
> +   return n;
> + }
> + return NULL;
> +}

Too large for inlining.

Was list_for_each_entry_rcu() needed?

Does this function have any caller-provided locking requirements?

> +/*
> + * Note: called from kernel/cgroup.c with cgroup_lock() held.
> + */
> +static struct cgroup_subsys_state *iothrottle_create(
> +   struct cgroup_subsys *ss, struct cgroup *cont)

static struct cgroup_subsys_state *iothrottle_create(struct cgroup_subsys *ss,
      struct cgroup *cont)

would be more typical code layout (here and elsewhere)

static struct cgroup_subsys_state *
iothrottle_create(struct cgroup_subsys *ss, struct cgroup *cont)

is another way.

```
> +{
> + struct iothrottle *iot;
> +
> + iot = kmalloc(sizeof(*iot), GFP_KERNEL);
> + if (unlikely(!iot))
> +  return ERR_PTR(-ENOMEM);
> +
> + INIT_LIST_HEAD(&iot->list);
> + spin_lock_init(&iot->lock);
> +
> + return &iot->css;
> +}
> +
> +/*
> + * Note: called from kernel/cgroup.c with cgroup_lock() held.
> + */
> +static void iothrottle_destroy(struct cgroup_subsys *ss, struct cgroup *cont)
> +{
> + struct iothrottle_node *n, *p;
> + struct iothrottle *iot = cgroup_to_iothrottle(cont);
> +
> + /*
> +  * don't worry about locking here, at this point there must be not any
> +  * reference to the list.
> +  */
> + list_for_each_entry_safe(n, p, &iot->list, node)
> +  kfree(n);
> + kfree(iot);
> +}
> +
> +static ssize_t iothrottle_read(struct cgroup *cont,
> +         struct cftype *cft,
> +         struct file *file,
> +         char __user *userbuf,
> +         size_t nbytes,
> +         loff_t *ppos)
> +{
> + struct iothrottle *iot;
> + char *buffer;
> + int s = 0;
> + struct iothrottle_node *n;
> + ssize_t ret;
> +
> + buffer = kmalloc(nbytes + 1, GFP_KERNEL);
> + if (!buffer)
> +  return -ENOMEM;
```

```
> +
> + cgroup_lock();
> + if (cgroup_is_removed(cont)) {
> +  ret = -ENODEV;
> +  goto out;
> + }
> +
> + iot = cgroup_to_iothrottle(cont);
> + rcu_read_lock();
> + list_for_each_entry_rcu(n, &iot->list, node) {
> +  unsigned long delta, rate;
> +
> +  BUG_ON(!n->dev);
> +  delta = jiffies_to_usecs((long)jiffies - (long)n->timestamp);
> +  rate = delta ? KBS(atomic_long_read(&n->stat) / delta) : 0;
> +  s += scnprintf(buffer + s, nbytes - s,
> +         "=== device (%u,%u) ===\n"
> +         "  bandwidth limit: %lu KiB/sec\n"
> +         "current i/o usage: %lu KiB/sec\n",
> +         MAJOR(n->dev), MINOR(n->dev),
> +         n->iorate, rate);
> + }
> + rcu_read_unlock();
> + ret = simple_read_from_buffer(userbuf, nbytes, ppos, buffer, s);
> +out:
> + cgroup_unlock();
> + kfree(buffer);
> + return ret;
> +}
```

This is a kernel->userspace interface. It is part of the kernel ABI.
We will need to support in a back-compatible fashion for ever. Hence
it is important. The entire proposed kernel<->userspace interface
should be completely described in the changelog or the documentation so
that we can understand and review what you are proposing.

```
> +static inline dev_t devname2dev_t(const char *buf)
> +{
> + struct block_device *bdev;
> + dev_t ret;
> +
> + bdev = lookup_bdev(buf);
> + if (IS_ERR(bdev))
> +  return 0;
> +
> + BUG_ON(!bdev->bd_inode);
> + ret = bdev->bd_inode->i_rdev;
> + bdput(bdev);
```

> +
> + return ret;
> +}

Too large to inline. I get tired of telling people this. Please just
remove all the inlining from all the patches. Then go back and
selectively inline any functions which really do need to be inlined
(overall reduction in generated .text is a good heuristic).

How can this function not be racy? We're returning a dev_t which
refers to a device upon which we have no reference. A better design
might be to rework the who9le thing to operate on a `struct
block_device *' upon whcih this code holds a reference, rather than
using bare dev_t.

I _guess_ it's OK doing an in-kernel filesystem lookup here. But did
you consider just passing the dev_t in from userspace? It's just a
stat().

Does all this code treat /dev/sda1 as a separate device from /dev/sda2?
 If so, that would be broken.

> +static inline int iothrottle_parse_args(char *buf, size_t nbytes,
> +   dev_t *dev, unsigned long *val)
> +{
> + char *p;
> +
> + p = memchr(buf, ':', nbytes);
> + if (!p)
> + return -EINVAL;
> + *p++ = '\0';
> +
> + /* i/o bandiwth is expressed in KiB/s */

typo.

This comment is incorrect, isn't it? Or at least misleading. The
bandwidth can be expressed in an exotically broad number of different
ways.

> + *val = ALIGN(memparse(p, &p), 1024) >> 10;
> + if (*p)
> + return -EINVAL;
> +
> + *dev = devname2dev_t(buf);
> + if (!*dev)
> + return -ENOTBLK;
> +

> + return 0;
> +}

uninline...

I think the whole memparse() thing is over the top:

+- BANDWIDTH is the maximum I/O bandwidth on DEVICE allowed by CGROUP (we can
+ use a suffix k, K, m, M, g or G to indicate bandwidth values in KB/s, MB/s
+ or GB/s),

For starters, we don't _display_ the bacndwidth back to the user in the
units with which it was written, so what's the point?

Secondly, we hope and expect that humans won't be diorectly echoing raw
data into kernel pseudo files. We shouild expect and plan for (or even
write) front-end management applications. And such applications won't
need these ornate designed-for-human interfaces.

IOW: I'd suggest this interface be changed to accept a plain old 64-bit
bytes-per-second and leave it at that.

> +static ssize_t iothrottle_write(struct cgroup *cont,
> +   struct cftype *cft,
> +   struct file *file,
> +   const char __user *userbuf,
> +   size_t nbytes, loff_t *ppos)
> +{
> + struct iothrottle *iot;
> + struct iothrottle_node *n, *tmpn = NULL;
> + char *buffer, *tmpp;

Please avoid variables called tmp or things derived from it. Surely we
can think of some more communucative identifier?

> + dev_t dev;
> + unsigned long val;
> + int ret;
> +
> + if (!nbytes)
> + return -EINVAL;
> +
> + /* Upper limit on largest io-throttle rule string user might write. */
> + if (nbytes > 1024)
> + return -E2BIG;
> +
> + buffer = kmalloc(nbytes + 1, GFP_KERNEL);
> + if (!buffer)

```
> +  return -ENOMEM;
> +
> + if (copy_from_user(buffer, userbuf, nbytes)) {
> +  ret = -EFAULT;
> +  goto out1;
> + }
> +
> + buffer[nbytes] = '\0';
```

strncpy_from_user()? (I'm not sure that strncpy_from_user() does the null-termination as desired).

```
> + tmpp = strstrip(buffer);
> +
> + ret = iothrottle_parse_args(tmpp, nbytes, &dev, &val);
> + if (ret)
> +  goto out1;
> +
> + if (val) {
> +  tmpn = kmalloc(sizeof(*tmpn), GFP_KERNEL);
> +  if (!tmpn) {
> +   ret = -ENOMEM;
> +   goto out1;
> +  }
> +  atomic_long_set(&tmpn->stat, 0);
> +  tmpn->timestamp = jiffies;
> +  tmpn->iorate = val;
> +  tmpn->dev = dev;
> + }
> +
> + cgroup_lock();
> + if (cgroup_is_removed(cont)) {
> +  ret = -ENODEV;
> +  goto out2;
> + }
> +
> + iot = cgroup_to_iothrottle(cont);
> + spin_lock(&iot->lock);
> + if (!val) {
> +  /* Delete a block device limiting rule */
> +  n = iothrottle_delete_node(iot, dev);
> +  goto out3;
> + }
> + n = iothrottle_search_node(iot, dev);
> + if (n) {
> +  /* Update a block device limiting rule */
> +  iothrottle_replace_node(iot, n, tmpn);
> +  goto out3;
```

```
> + }
> + /* Add a new block device limiting rule */
> + iothrottle_insert_node(iot, tmpn);
> +out3:
> + ret = nbytes;
> + spin_unlock(&iot->lock);
> + if (n) {
> +  synchronize_rcu();
> +  kfree(n);
> + }
> +out2:
> + cgroup_unlock();
> +out1:
> + kfree(buffer);
> + return ret;
> +}
> +
> +static struct cftype files[] = {
> + {
> +  .name = "bandwidth",
> +  .read = iothrottle_read,
> +  .write = iothrottle_write,
> + },
> +};
> +
> +static int iothrottle_populate(struct cgroup_subsys *ss, struct cgroup *cont)
> +{
> + return cgroup_add_files(cont, ss, files, ARRAY_SIZE(files));
> +}
> +
> +struct cgroup_subsys iothrottle_subsys = {
> + .name = "blockio",
> + .create = iothrottle_create,
> + .destroy = iothrottle_destroy,
> + .populate = iothrottle_populate,
> + .subsys_id = iothrottle_subsys_id,
> +};
> +
> +static inline int __cant_sleep(void)
> +{
> + return in_atomic() || in_interrupt() || irqs_disabled();
> +}
```

err, no.

I don't know what this is doing or why it was added, but whatever it is
it's a hack and it all needs to go away.

Please describe what problem this is trying to solve and let's take a
look at it. That should have been covered in code comments anyway.
But because it wasn't, I am presently unable to help.

> +void cgroup_io_throttle(struct block_device *bdev, size_t bytes)
> +{
> + struct iothrottle *iot;
> + struct iothrottle_node *n;
> + unsigned long delta, t;
> + long sleep;
> +
> + if (unlikely(!bdev || !bytes))
> + return;
> +
> + iot = task_to_iothrottle(current);
> + if (unlikely(!iot))
> + return;
> +
> + BUG_ON(!bdev->bd_inode);
> +
> + rcu_read_lock();
> + n = iothrottle_search_node(iot, bdev->bd_inode->i_rdev);
> + if (!n || !n->iorate)
> + goto out;
> +
> + /* Account the i/o activity */
> + atomic_long_add(bytes, &n->stat);
> +
> + /* Evaluate if we need to throttle the current process */
> + delta = (long)jiffies - (long)n->timestamp;
> + if (!delta)
> + goto out;
> +
> + t = usecs_to_jiffies(KBS(atomic_long_read(&n->stat) / n->iorate));

Are you sure that n->iorate cannot be set to zero between the above
test and this division? Taking a copy into a local variable would fix
that small race.

> + if (!t)
> + goto out;
> +
> + sleep = t - delta;
> + if (unlikely(sleep > 0)) {
> + rcu_read_unlock();
> + if (__cant_sleep())
> +   return;
> + pr_debug("io-throttle: task %p (%s) must sleep %lu jiffies\n",

> +   current, current->comm, delta);
> + schedule_timeout_killable(sleep);
> + return;
> + }
> + /* Reset i/o statistics */
> + atomic_long_set(&n->stat, 0);
> + /*
> + * NOTE: be sure i/o statistics have been resetted before updating the
> + * timestamp, otherwise a very small time delta may possibly be read by
> + * another CPU w.r.t. accounted i/o statistics, generating unnecessary
> + * long sleeps.
> + */
> + smp_wmb();
> + n->timestamp = jiffies;
> +out:
> + rcu_read_unlock();
> +}
> +EXPORT_SYMBOL(cgroup_io_throttle);

I'm confused. This code is using jiffies but the string "HZ" doesn't
appears anywhere in the diff. Where are we converting from the
kernel-internal HZ rate into suitable-for-exposing-to-userspace units?

HZ can vary from 100 to 1000 (approx). What are the implications of
this for the accuracy of this code?

I have no comments on the overall design. I'm not sure that I understand
it yet.

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