Subject: Re: [PATCH 2/3] i/o bandwidth controller infrastructure
Posted by Andrea Righi on Thu, 26 Jun 2008 22:36:46 GMT

Andrew,

thanks for your time and the detailed revision. I'll try to fix
everything you reported and better document the code according to your
suggestions. I'll re-submit a new patchset version ASAP.

A few comments below.

Andrew Morton wrote:
[snip]

```
>> +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.
```

and BTW I was actually wondering if the output could be changed with something less human readable and better parseable, I means just print only raw numbers. And describe the semantic in the documentation.

```
>> +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.
```

However, holding a reference wouldn't allow to unplug the device, i.e.
a USB disk. As reported in Documentation/controllers/io-throttle.txt:

WARNING: per-block device limiting rules always refer to the dev_t
device number. If a block device is unplugged (i.e. a USB device) the
limiting rules defined for that device persist and they are still valid
if a new device is plugged in the system and it uses the same major and
minor numbers.

This would be a feature in my case, but I don't know if it would be a
bug in general.

> 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().

Yes, and that seems to be more reasonable, since we display major,minor
numbers in the output.

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

Yes, all the partitions are treated as separate devices with
(potentially) different limiting rules, but I don't understand why it
would be broken... dev_t has both minor and major numbers, so it would
be possible to select single partitions as well.

```
>> +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.

Yes.

>
>> + *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.

I agree.

>> +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.
```

n->iorate can only change via userspace->kernel interface, that just
replaces the node in the list using the rcu way. AFAIK this shouldn't be
racy, but better to do it using the local variable to avoid future bugs.

```
>> + 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?

The code uses jiffies_to_usecs() and usecs_to_jiffies(), that should be
ok, isn't it?

-Andrea