Subject: Re: [PATCH -mm 2/3] i/o bandwidth controller infrastructure
Posted by Andrea Righi on Mon, 14 Jul 2008 21:26:51 GMT
View Forum Message <> Reply to Message

Li Zefan wrote:
>> +/* The various types of throttling algorithms */
>> +enum iothrottle_strategy {
>> + IOTHROTTLE_LEAKY_BUCKET,
>
> It's better to explicitly assigned 0 to IOTHROTTLE_LEAKY_BUCKET.

OK.

>
>> + IOTHROTTLE_TOKEN_BUCKET,
>> +};
>
>> +static int iothrottle_parse_args(char *buf, size_t nbytes, dev_t *dev,
>> + u64 *iorate,
>> + enum iothrottle_strategy *strategy,
>> + s64 *bucket_size)
>> +{
>> + char *p = buf;
>> + int count = 0;
>> + char *s[3];
>> + unsigned long strategy_val;
>> + int ret;
>> +
>> + /* split the colon-delimited input string into its elements */
>> + memset(s, 0, sizeof(s));
>> + while (count < ARRAY_SIZE(s)) {
>> + p = memchr(p, ':', buf + nbytes - p);
>> + if (!p)
>> + break;
>> + *p++ = '\0';
>> + if (p >= buf + nbytes)
>> + break;
>> + s[count++] = p;
>> + }
>
> use strsep()

OK.

>
>> +
>> + /* i/o bandwidth limit */
>> + if (!s[0])

```
>> +  return -EINVAL;
>> + ret = strict_strtoull(s[0], 10, iorate);
>> + if (ret < 0)
>> +  return ret;
>> + if (!*iorate) {
>> +  /*
>> +   * we're deleting a limiting rule, so just ignore the other
>> +   * parameters
>> +   */
>> +  *strategy = 0;
>> +  *bucket_size = 0;
>> +  goto out;
>> + }
>> + *iorate = ALIGN(*iorate, 1024);
>> +
>> + /* throttling strategy */
>> + if (!s[1])
>> +  return -EINVAL;
>> + ret = strict_strtoul(s[1], 10, &strategy_val);
>> + if (ret < 0)
>> +  return ret;
>> + *strategy = (enum iothrottle_strategy)strategy_val;
>> + switch (*strategy) {
>> + case IOTHROTTLE_LEAKY_BUCKET:
>> +  /* leaky bucket ignores bucket size */
>> +  *bucket_size = 0;
>> +  goto out;
>> + case IOTHROTTLE_TOKEN_BUCKET:
>> +  break;
>> + default:
>> +  return -EINVAL;
>> + }
>> +
>> + /* bucket size */
>> + if (!s[2])
>> +  return -EINVAL;
>> + ret = strict_strtoll(s[2], 10, bucket_size);
>> + if (ret < 0)
>> +  return ret;
>> + if (*bucket_size < 0)
>> +  return -EINVAL;
>> + *bucket_size = ALIGN(*bucket_size, 1024);
>> +out:
>> +
>> + /* block device number */
>> + *dev = devname2dev_t(buf);
>
> why not parse dev before parse bandwidth limit ?
```

Well... due to the filesystem lookup in devname2dev_t() this option is the most expensive to be parsed. For this reason I put it at the end, so if any other parameter is wrong we can skip the lookup_bdev(). Does it make sense?

>
>> + return *dev ? 0 : -EINVAL;
>> +}
>> +
>> +static int iothrottle_write(struct cgroup *cgrp, struct cftype *cft,
>> +   const char *buffer)
>> +{
>> + struct iothrottle *iot;
>> + struct iothrottle_node *n, *newn = NULL;
>> + dev_t dev;
>> + u64 iorate;
>> + enum iothrottle_strategy strategy;
>> + s64 bucket_size;
>> + char *buf;
>> + size_t nbytes = strlen(buffer);
>> + int ret = 0;
>> +
>> + buf = kmalloc(nbytes + 1, GFP_KERNEL);
>> + if (!buf)
>> + return -ENOMEM;
>> + memcpy(buf, buffer, nbytes + 1);
>> +
>
> redundant kmalloc, just use buffer, and ...

uhmm... I would apply strsep() directly to buffer in this way, that is a const char *.

>
>> + ret = iothrottle_parse_args(buf, nbytes, &dev, &iorate,
>> +   &strategy, &bucket_size);
>> + if (ret)
>> + goto out1;
>> + if (iorate) {
>> + newn = kmalloc(sizeof(*newn), GFP_KERNEL);
>> + if (!newn) {
>> +   ret = -ENOMEM;
>> +   goto out1;
>> + }
>> + newn->dev = dev;
>> + newn->iorate = iorate;

```
>> +  newn->strategy = strategy;
>> +  newn->bucket_size = bucket_size;
>> +  newn->timestamp = jiffies;
>> +  atomic_long_set(&newn->stat, 0);
>> +  atomic_long_set(&newn->token, 0);
>> + }
>> + if (!cgroup_lock_live_group(cgrp)) {
>> +  kfree(newn);
>> +  ret = -ENODEV;
>> +  goto out1;
>> + }
>> + iot = cgroup_to_iothrottle(cgrp);
>> +
>> + spin_lock(&iot->lock);
>> + if (!iorate) {
>> +  /* Delete a block device limiting rule */
>> +  n = iothrottle_delete_node(iot, dev);
>> +  goto out2;
>> + }
>> + n = iothrottle_search_node(iot, dev);
>> + if (n) {
>> +  /* Update a block device limiting rule */
>> +  iothrottle_replace_node(iot, n, newn);
>> +  goto out2;
>> + }
>> + /* Add a new block device limiting rule */
>> + iothrottle_insert_node(iot, newn);
>> +out2:
>> + spin_unlock(&iot->lock);
>> + cgroup_unlock();
>> + if (n) {
>> +  synchronize_rcu();
>> +  kfree(n);
>> + }
>> +out1:
>> + kfree(buf);
>> + return ret;
>> +}
>> +
>> +static struct cftype files[] = {
>> + {
>> +  .name = "bandwidth",
>> +  .read_seq_string = iothrottle_read,
>> +  .write_string = iothrottle_write,
>
> and you should specify .max_write_len = XXX unless XXX <= 64.
> You use 1024 in v4.
```

OK. Anyway, probably something like 256 would be enough.

Thanks again for the detailed revision Li! I'll include your fixes in
a new patchset version.

-Andrea

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