BTRFS and free space - emergency response

Feb 11, 2019 in **FILESYSTEMS** • **LINUX** • **SYSADMIN**
9 min read

I run BTRFS on my root filesystem (on Linux), mostly for the quick snapshot and restore functionality. Yesterday I ran into a common problem: **my drive was suddenly full**. I went from 4GB of free space on my system drive to 0 in an instant, causing all sorts of chaos on my system.

This problem happens to lots of people because BTRFS doesn't have a linear relationship to "free space available". There are a few concepts that get in the way:

- **Compression**: BTRFS supports compressing data as it writes. This obviously changes the amount of data that can be stored. - 50MB of text may take only 5MB "room" on the drive.
- **Metadata**: BTRFS stores your data separately from metadata. Both data and metadata occupy "space".
- **Chunk allocation**: BTRFS allocates space for your data in chunks.
- **Multiple devices**: BTRFS supports multiple devices working together, RAID-style. That means there's extra information to store for every file. For example, RAID-1 stores two copies of every file, so a 50MB file takes 100MB of space.
- **Snapshots**: BTRFS can store snapshots of your device, which really store more like a diff from the current state. How much data is in the diff depends on your current state... so the snapshot itself doesn't have a consistent size.
- **Nested volumes**: BTRFS lets you divide the filesystem into "subvolumes" - each of which can (someday) have its own RAID configuration.

It's easy to look at the drive and tell how many MiB of space has not been used yet. But it's very hard to accurately say how much of your data you can write in that space. For this reason the amount of "free space" reported on BRFS volumes by system utilities like `df` can jump a lot - like my disappearing 4GiB. Worse, the free space reported by general tools is misleading. BTRFS can run out of space while `df` still thinks you have lots available.

Let's walk through how BTRFS stores data, to understand the problem a bit better. Then we can solve it with some of BTRFS' own tools.

How much free space do I have?

Rather than using general tools like `df` to answer this question, it's better to get more detail using the `btrfs` CLI tool.

BTRFS starts out with a big pool of raw storage, and allocates as it goes. You can get a listing of all the devices in a block device like this:

```
$ sudo btrfs fi show
Label: 'OS'  uuid: c0d21ade-5570-41a3-b0cf-a5ce219e7a8e
  Total devices 1 FS bytes used 31.74GiB
  devid    1 size 48.83GiB used 47.80GiB path /dev/nvme0n1p2
```

In this case, I only have one physical device involved. You can see that it gives me a total number of bytes allocated, compared to the total size. In another filesystem this might be the number reported to `df`. Not so with BTRFS! Let's dig deeper.

```
$ btrfs fi df /

Data, single: total=45.75GiB, used=30.56GiB
System, single: total=32.00MiB, used=16.00KiB
Metadata, single: total=2.02GiB, used=1.17GiB
GlobalReserve, single: total=89.31MiB, used=0.00B
```

The "total" values here are the breakdown of what the first command counts as "used". `btrfs fi df` shows us of the allocated space, how much is actually storing data, and how much is just empty allocation. In this case: on my 48GiB device, 47GiB is allocated. Of the allocation, 31GiB is actually storing data. Side note: if you're in a multi-drive situation this command will take into account RAID metadata.

Here's an easier view:

```
$ sudo btrfs fi usage /

Overall:
    Device size:      48.83GiB
    Device allocated:     47.80GiB
    Device unallocated:      1.03GiB
    Device missing:        0.00B
    Used:       31.74GiB
    Free (estimated):     16.22GiB  (min: 16.22GiB)
    Data ratio:           1.00
    Metadata ratio:         1.00
    Global reserve:     89.31MiB  (used: 0.00B)

Data,single: Size:45.75GiB, Used:30.56GiB
   /dev/nvme0n1p2   45.75GiB

Metadata,single: Size:2.02GiB, Used:1.18GiB
```

```
   /dev/nvme0n1p2    2.02GiB

System,single: Size:32.00MiB, Used:16.00KiB
   /dev/nvme0n1p2   32.00MiB

Unallocated:
   /dev/nvme0n1p2    1.03GiB
```

This shows the breakdown of space allocated and used across all the devices in this block device. “Overall” is for the whole block device, and that “Free (estimated)” number is what gets reported to `df`.

This is a problem: **most of my normal tools tell me I have 15GB free space. But if I write 1GiB more data, BTRFS will run out of space anyways.** This issue is a pain in the ass and hard to diagnose. It’s even harder to fix, since most of the solutions require having some extra space on the device.

Converting unused allocation to free space

So, why does BTRFS allocate so much space to store such a small amount of data? Here I am storing 31GiB of data in 47GiB of allocation, the used/total ratio is 0.66! This is very inefficient. It’s an unfortunate consequence of being a copy-on-write filesystem - BTRFS starts every write in a freshly allocated chunk. But the chunksize is static, and files come in all sizes. So lots of the time, a chunk is incompletely filled. That’s the “allocated but not used” space we’re complaining about.

Fortunately there’s a way to address this problem: BTRFS has a tool to “rebalance” your filesystem. It was originally designed for balancing the data stored across multiple drives (hence the name). It is also useful in single drive configurations though, to rebalance how data is stored within the allocation.

By default, `balance` will rewrite *all* the data on the disk. This is probably unnecessary. Chunks will be unevenly filled, but we saw above that the average should be about 66% used. So we’ll filter based on data (`-d`) usage, and only rebalance chunks that are less than 66% used. That will leave any partially filled chunks which are more-filled than average.

```
# Run it in the background, cause it takes a long time.
$ sudo btrfs balance start -dusage=66 / &
# Check status
$ sudo btrfs balance status -v /
Balance on '/' is running
1 out of about 27 chunks balanced (5 considered),  96% left
Dumping filters: flags 0x1, state 0x1,
# Or be lazy, and have bash report status every 60 seconds.
$ while :; do sudo btrfs balance status -v / ; sleep 60; done
Balance on '/' is running
3 out of about 27 chunks balanced (12 considered),  89% left
```

```
Dumping filters: flags 0x1, state 0x1, force is off
  DATA (flags 0x2): balancing, usage=66
Balance on '/' is running
4 out of about 27 chunks balanced (13 considered),  85% left
Dumping filters: flags 0x1, state 0x1, force is off
  DATA (flags 0x2): balancing, usage=66
...
# When the balance operation finishes:
Done, had to relocate 19 out of 59 chunks
```

There's a nice big differnce once it's finished:

```
$ btrfs filesystem df /
Data, single: total=32.53GiB, used=30.83GiB
System, single: total=32.00MiB, used=16.00KiB
Metadata, single: total=2.02GiB, used=1.17GiB
GlobalReserve, single: total=84.67MiB, used=0.00B
```

That's 15GiB of space allocated for other use. My usage ratio is now 0.94. Huzzah! In some rare cases you may need to do this on the Metadata allocation (use `-musage` instead of `-dusage` above).

If you've already run out of space

If you have already run out of space, you can't run a `balance`! In that caseyou have to get sneaky. Here are your options:

1) Free up space

This is harder than it sounds. If you just delete data, it will probably leave those chunks partially filled and therefore allocated. What you really need is *unallocated* space. The easiest place to get this is by deleting snapshots. Start from the oldest one, since it will be the biggest.

Once you have a little bit of wiggle room, rebalance a small segment, like Metadata. Then proceed with rebalancing data as described above.

2) Add some space

Don't forget, a BTRFS volume can span multiple devices! I had to exercise this option recently. Add a device - a flash drive will do, but choose the fastest thing you can - and add it to the BTRFS volume.

```
# Add your extra drive (/dev/sda).
$ sudo btrfs device add -f /dev/sda /
# Now run the smallest balance operation you can.
$ sudo btrfs balance start -dusage=1 /
Done, had to relocate 1 out of 59 chunks
# Remove the device, and run a proper balance.
$ sudo btrfs device remove /dev/sda /
$ sudo btrfs balance start -dusage=66 /
Done, had to relocate 18 out of 59 chunks
```

Balance operations usually take a long time - more than an hour is not unusual. It will take even longer with slow flash media involved. For that reason, I use a very low balance filter (`-dusage=`) in this example. We only need to free up a teensy bit of space to run balance again without the flash disk in the mix.

And this last option is how I saved my computer last night. I hope this helps someone out of a similar predicament someday.

UPDATE

Now that I've had to do this a few times, it's *way* better to rebalance a full filesystem by adding a ramdisk to it. Not only is it faster than a flash device, it's also more reliable in most cases... and certainly for my kind of use case (a developer laptop) the important preconditions apply: lots of RAM, reliable power source. Here's the recipe:

```
# Create a ramdisk. Make sure /dev/ram0 isn't in use already before doing this!
$ sudo mknod -m 660 /dev/ram0 b 1 0
$ sudo chown root:disk /dev/ram0
# Mount the ramdisk with a concrete size. Otherwise it grows to whatever is needed.
$ sudo mkdir /mnt/ramdisk
$ sudo mount -t ramfs -o size=4G,maxsize=4G /dev/ram0 /mnt/ramdisk
# Create a file on the ramdisk to use as a loopback device.
$ sudo dd if=/dev/zero of /mnt/ramdisk/extend.img bs=4M count=1000
$ sudo losetup -fP /mnt/ramdisk/extend.img
# figure out which loopback device ID is yours
$ sudo losetup -a |grep extend.img
/dev/loop10: [5243078]:8563965 (/mnt/ramdisk/extend.img)
# Add the loopback device to the btrfs filesystem
$ sudo btrfs device add /dev/loop10 /
# Decide on your balance ratio and balance as usual.
$ sudo btrfs fi usage / |head -n 6
Overall:
    Device size:         400.91GiB
    Device allocated:        396.36GiB
    Device unallocated:        4.55GiB
    Device missing:          0.00B
    Used:            348.91GiB
```

```
$ echo 'scale=2;348/396' |bc
.87

$ sudo btrfs balance start -dusage=87 /
Done, had to relocate 46 out of 400 chunks
# Remove the device and destroy it.
$ sudo btrfs device delete /dev/loop0 /
$ sudo losetup -d /dev/loop10
$ sudo umount /mnt/ramdisk
$ sudo rm -rf /dev/ram0
```

ALSO ON **OHTHEHUGEMANATEE**

Perfect Linux Mint on My Macbook Pro 9,2

3 years ago • 1 comment

A few months ago I posted about my frustrations getting El Capitan working with …

An Open Letter to Wired Magazine

5 years ago • 11 comments

Dear Wired – I read your post about the changes in ad blocker policy on your …

Kubernetes for Stateful Applications: …

2 years ago • 1 comment

I recently got to proctor an Openhack event on modern containerization. It ended …

I'm Joining Because Th

3 years ago • 7

I'm excited to I've signed wi a Principal So