Block device configuration done right

Markus Armbruster <armbru@redhat.com>
Kevin Wolf <kwolf@redhat.com>
KVM Forum 2013

Part I

What's wrong with it?

What's a block device in QEMU?

Consists of

- frontend (guest part, a.k.a. device model)
  virtual IDE disk, SD card, ...
- backend (host part)
  image file, logical volume, remote image, ...

redhat.

Beginnings: all I want is an image file

How hard can this be?

`-hda` *FILENAME*	`-hdb` *FILENAME*
`-hdc` *FILENAME*	`-hdd` *FILENAME*
`-cdrom` *FILENAME*	`-mtdblock` *FILENAME*
`-sd` *FILENAME*	`-pflash` *FILENAME*
`-fda` *FILENAME*	`-fdb` *FILENAME*

As many options as block devices

Surprise: one size doesn't fit all

Oops, I need to control geometry of hda!

Easy, just add another option:
`-hdachs` *C*,*H*,*S*

Second thoughts, add one with parameters:
`-drive if=ide,index=0,cyls=`*C*`,heads=`*H*`,...`

Frontend-specific configuration

Oops, I need to control virtio-blk's PCI address!

Easy, just add a parameter for it:
`-drive if=virtio,addr=`*`DEVFN`*`,...`

Second thoughts:

- `addr` valid only with `if=virtio`
- Frontends have many more parameters...
- Bake them all into block layer? *No way!*

Separate frontend & backend configuration

`-drive` configures both frontend and backend
Problem: bad at configuring frontends

Solution:

- Create a way to configure just a backend
  `-drive if=none,id=drive0,...`
- Then configure the frontend the usual way
  `-device ide-hd,drive=drive0,...`

Backend configuration

`-drive` designed for "format over protocol"

Can do:

- Select a format driver (parameter `format`)
- Configure a protocol (parameter `file`)
- Generic parameters (remaining ones)

We'll see: bad at configuring backends, too!

Example: raw over file

```
-drive if=none,format=raw,file=foo.img
```

Format raw over protocol file (both block drivers)
Format raw does effectively nothing
It's just for filling "format over protocol" mold

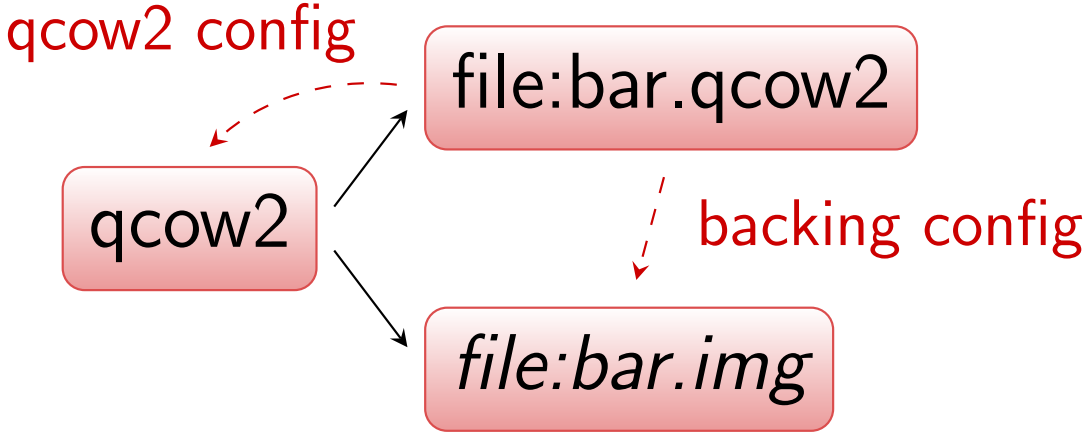
Example: raw over nbd

```
if=none,format=raw,file=nbd:cloud9:1234
```

Block layer & protocol driver parse value of file
Ad hoc syntaxes abound; anathema to QMP
Great fun: users putting colons in filenames

Example: qcow2 over file

```
if=none,format=qcow2,file=bar.qcow2
```



Can't control qcow2 parameters

Can't control backing protocol

A use case for full protocol control

Idea:

- Run QEMU with minimal privileges
- Pass it file descriptors, not filenames

Required to get best mileage out of SELinux & NFS

But: can't pass a backing fd!
(Libvirt is not amused)

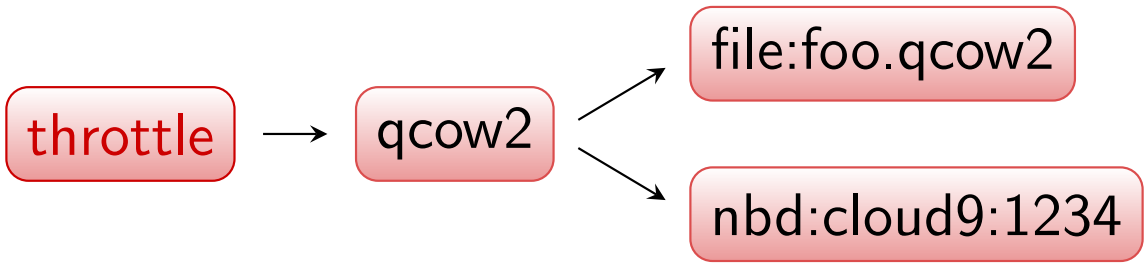
Example: raw over blkdebug over ...

```
if=none,format=raw,file=blkdebug:dbg.cfg:...
```

Protocol recursion: "..." parsed as protocol
blkdebug acts as filter

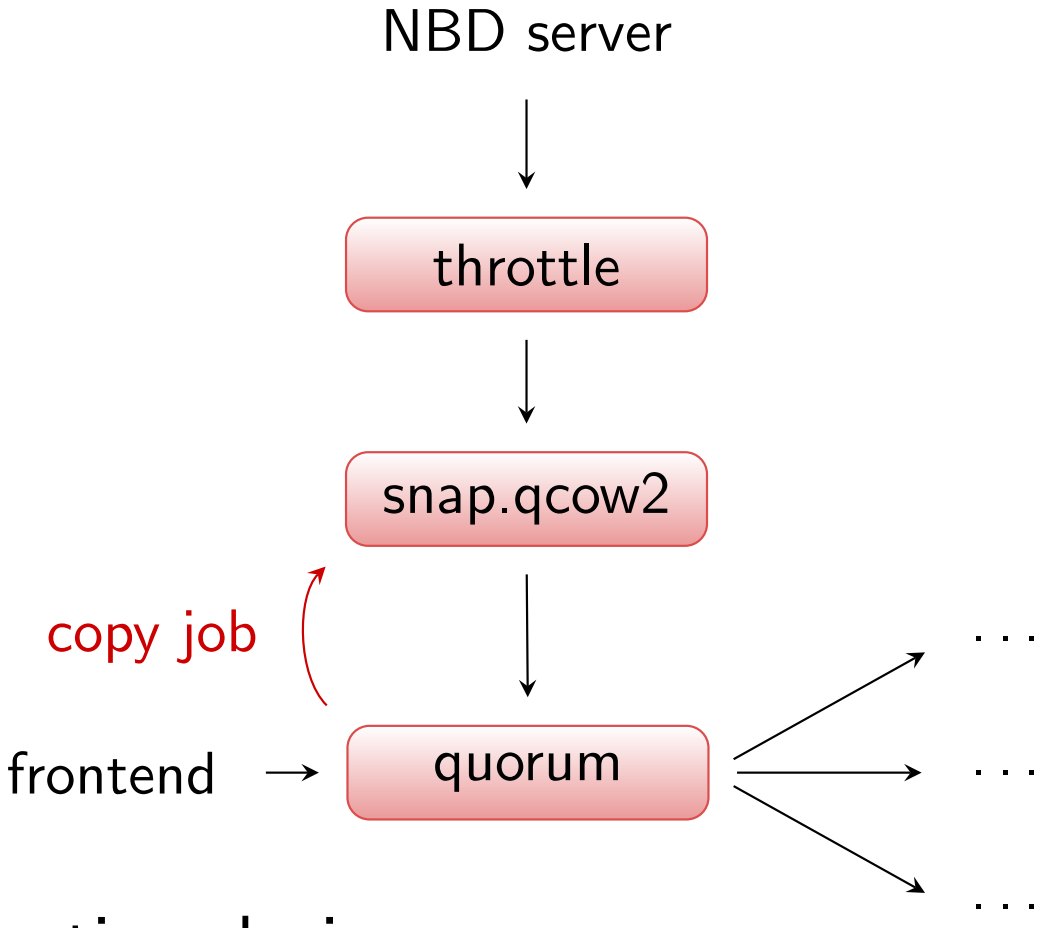
Example: I/O throttling done right

Want filter (now: baked into block layer)
Want to apply anywhere (now: only at root)



More than just “format over protocol”

Example: Image fleecing



Reconfiguration during use

This ain't trees anymore!

Putting it all together

- Got a (mostly treeish) graph to deal with
- Nodes are block driver instances, frontends, backends, block jobs, ...
- Need full control over every node
- QMP is JSON, can do graph
- Command line is going to be awkward
- Far out: fit into QOM

Part II
Implementing the new way

What do I need to implement?

"blockdev" has become a massive project to fix everything in the block layer

- Three years talk, but nothing happened
- Need to break it up into manageable pieces
- Start **somewhere**

Section 1

The command line

redhat.

Get rid of options in “filenames”

Instead allow driver-specific options in -drive

Old:

```
-drive file=nbd:localhost:10809
```

New:

```
-drive file.driver=nbd,file.host=localhost
```

- `file.port` has a default value now

redhat.

Colons and file names

Old:

`-drive file=test:a.qcow2`

- Parsed as protocol `test` with argument `a.qcow2`
- No way to escape the colon
- Similar trouble with colons in options

New:

`-drive file.filename=test:a.qcow2`

- Works fine; driver-specific options are not parsed

redhat.

Allow options for format drivers

Format drivers never see the filename

- No evil filename parsing
- Can't provide options

The new syntax can provide options:

```
-drive file=test.qcow2,lazy-refcounts=on
```

redhat.

Options for backing files

Backing file name/format taken from qcow2 header

- libvirt would rather pass file descriptors
- No control over other options:
  You always get the qemu defaults

We can allow passing options now:

```
-drive file=test.qcow2,\
backing.file.filename=/dev/fdset/1
```

- Still to be done for many options

Section 2

QMP

redhat.

What's the goal of blockdev-add?

We need a QMP command to hotplug block devices that...

- ...takes structured data instead of a string
- ...configures only the backend
- ...creates block devices without any magic attached
  - e.g. don't automatically disappear when the frontend is unplugged

redhat.

What does it look like?

Minimal example:

```
{ "execute": "blockdev-add",
  "arguments": { "options" : {
      "id":         "my_disk",
      "driver":     "qcow2",
      "file":     { "driver": "file",
                    "filename": "test.qcow2" } } } }
```

- `-drive` mapped to JSON
- But JSON schema restricts allowed options
  - Only backend options
  - Only long-term supportable options

redhat.

A more interesting example

Override the backing file with a file descriptor:

```
{ "execute": "blockdev-add",
  "arguments": {
    "options": {
      "driver":  "qcow2",
      "id":      "my_disk",
      "discard": "unmap",
      "cache":   { "direct": true,
                   "writeback": true },
      "file":    { "driver": "file",
                   "filename": "/tmp/test.qcow2" },
      "backing": { "driver": "raw",
                   "file": { "driver": "file",
                             "filename": "/dev/fdset/4" }
                 } } } }
```

Section 3

Complex graphs

redhat.

Building a tree

Remember: *QMP is JSON, and JSON loves trees*

Keep simple things simple: Trees can declare children inline

```
"backing": { "driver": "qcow2",
             "file": { "driver": "file",
                       "filename": "/dev/fdset/4" },
             "backing": ... }
```

redhat.

Building any graph

- Create the referenced block device with a separate `blockdev-add` call
- Reference it by its ID:

```
"backing": "my_backing_file"
```

- Requires changes to some assumptions:
  - Suddenly not only the root has an ID
  - Nodes start having a separate life cycle

redhat.

Section 4

Next steps

redhat.

Next steps (I)

Convert more options:

- Makes them available below top level

Convert drivers from filename parsing to options:

- Mainly network protocols are missing

redhat.

Next steps (II)

Allow building complex graphs:

- Involves getting rid of assumptions

Create a block filter infrastructure:

- Makes I/O throttling and Copy on Read configurable with `blockdev-add`

Questions?