Thin-provisioned disks with QEMU and KVM

Paolo Bonzini
Red Hat, Inc.
devconf.cz, February 2014
QEMU vs. KVM

- QEMU is where the cool stuff happens
  - Fast paravirtualized hardware (virtio)
  - Virtual machine lifecycle (including migration)
  - Storage stack
- kvm.ko lets you use QEMU for virtualization
  - Narrow-scoped, mature code
  - Also cool :)
- This talk will be about QEMU
Outline

- Thin provisioning concepts
- What was there
- Requirements
- Designing a thin-provisioning API
- Future work
Thin provisioning concepts

- A disk is made of many blocks
- The user tells the disks how it's using them
- The disk can be used more efficiently
  - Speed and durability gains for SSDs
  - Oversubscription of networked storage
  - Efficient maintenance operations
- Better name (from SCSI standard): “Logical block provisioning”
Thin provisioning and virtualization

- The advantages extend to virtualization
  - Can be applied to any storage backend
  - “Software-defined storage” before it became cool
- The user is the guest administrator
  - Only pay for actually used space
  - Guest disks can be overprovisioned
- Host admin saves disk space
Multiple storage layers

- qcow2, raw, ...
  - file, block device, gluster, iSCSI
    - ext4/XFS
    - SSD, NAS, dm-thinp
  - gluster, iSCSI
  - gluster, NFS
  - Ext4, XFS
What was there

- Lazy allocation of blocks
- Differential disks (copy-on-write)
- High-level watermark: management can query the highest sector in use
- QEMU-specific formats: qcow, qcow2
- Foreign formats: cow, vdi, vmdk, vpc, vhdx, ...
What was missing

- Could not reclaim space at the host level
  - Blocks will never be discarded once written
- No support for “raw” block devices or files
  - Mandatory performance cost from additional layers
  - Cannot unify management for physical and virtual machines
Objectives (host)

- Support a wide range of backends
- Maintenance operations should be faster thanks to hints from the guest
- Maintenance operations should have access to all the capabilities of the backend
Example: maintenance operations (1)

- Raw block devices may be nonzero when they are assigned to a VM
- Expensive scrubbing required when a virtual disk is created
  - BLKZEROOUT can offload this to the storage
  - The disk can optimize the operation (e.g. won't scrub all the way down to the platters)
Example: maintenance operations (2)

• Data copied from backing file for faster access
Objectives (guest)

- virtio-blk support not required
  - Use SCSI commands with virtio-scsi
  - Accurate implementation of SCSI commands
- Changing the host backend should not modify the guest hardware
- Guests should have access to most capabilities of the backend
SCSI logical block provisioning

- Three types of disks
  - Fully-provisioned
  - Thin-provisioned
  - Resource-provisioned

- Three types of blocks
  - Deallocated
  - Anchored
  - Mapped

Logical block management enabled

On disk space allocated

Block not in use
SCSI logical block provisioning

VPD page 0xb2

- LBPU: UNMAP command supported
- LBPWS/LBPWS10: unmap with WRITE SAME
- LBPRZ: “unmapped” blocks read zero
- ANC_SUP: ANCHOR supported

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ANCHOR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>ANCHOR</td>
<td>UNMAP</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SCSI logical block provisioning commands

- UNMAP: unmap aligned group of sectors
  - Granularity also available in VPD
- WRITE SAME: write the same block to a consecutive group of sectors
  - No alignment considerations
  - Without UNMAP: just write them
  - With UNMAP: can also unmap, as long as sectors read back to the given payload
SCSI logical block provisioning and Linux

- Can use either UNMAP, or WRITE SAME with UNMAP bit set
  - UNMAP preferred if both available
  - Can be tweaked from sysfs
- GET LBA STATUS: not directly accessible
  - lseek(fd, ..., SEEK_HOLE/DATA) could do it
- ANCHOR: not supported
- WRITE SAME without UNMAP: only supported with zero payload (BLKZEROOUT ioctl)
Guest interface limitations

- Host will not necessarily zero discarded sectors
  - LBPRZ must not change when changing storage backend
  - Choice 1: if host cannot discard-and-zero, never discard sectors and just write out zeroes
  - Choice 2: guest must default to LBPRZ=0
- Choice 1 not plausible, kills thin provisioning
- Can still detect WRITE SAME + UNMAP + zero payload and optimize it
Guest interface limitations

- Discard not always desirable
  - May want to keep files preallocated (for performance)
  - Avoid too much fragmentation
- Disable it by default
  - Enable it with QEMU “-drive ...,discard=on” or equivalent libvirt XML
  - SCSI commands will remain available but do nothing
  - Permitted by SCSI spec with LBPRZ=0
<table>
<thead>
<tr>
<th>Layers &amp; APIs</th>
<th>ATA: Trim</th>
<th>SCSI: Unmap, Write Same, Get Block Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest</td>
<td>QEMU</td>
<td>discard, get_block_status, write_zeroes, get_info</td>
</tr>
<tr>
<td>QEMU</td>
<td>Network</td>
<td>Gluster: discard, zerofill iSCSI: Unmap, Write Same, Get Block Status, LBPRZ</td>
</tr>
<tr>
<td>Network</td>
<td>Linux</td>
<td>File system: fallocate, FIEMAP, xfsctl Block device: BLKDISCARD, BLKZEROOOUT, BLKDISCARDZEROES</td>
</tr>
<tr>
<td>Linux</td>
<td>Storage</td>
<td>ATA: Trim SCSI: Unmap, Write Same, Get Block Status, LBPRZ</td>
</tr>
</tbody>
</table>
QEMU APIs: discard

- Discards a range of sectors (or an aligned subset of it)
- No-op unless the “discard=on” option was passed
- Users: ATA and SCSI emulation
QEMU APIs: get_block_status

- Return metadata for a group of sectors
  - Are the sectors read from an older snapshot?
  - Are the sectors allocated in the backend?
  - Are the sectors known to be zero?
  - Where is the data stored in the backend?
- More powerful than SCSI Get Block Status
- Users: “qemu-img convert”, “qemu-img map”, image streaming/mirroring, SCSI emulation (not yet)
QEMU APIs: write_zeroes

- Resembles SCSI Write Same with zero payload
- BDRV_O_MAY_UNMAP flag available
  - Same as SCSI “UNMAP” bit
  - Only matters if discard=on
- Users: “qemu-img convert”, SCSI emulation
QEMU APIs: get_info

- unallocated_blocks_are_zero
  - “This block is unallocated, do I know it's zero?”
  - Same as LBPRZ
  - Used by generic get_block_status code

- can_write_zeroes_with_unmap
  - “Will write_zeroes+BDRV_O_MAY_UNMAP try to unmap some blocks?”
  - Same as LPBRZ && LPBWS
Future work

- Improved tuning of file formats
- More homogeneous functionality for backends
- Better support for preallocation
Tuning of file formats

- qcow2 currently lets backing file data through after discard
Homogeneous functionality for backends

<table>
<thead>
<tr>
<th></th>
<th>ext4</th>
<th>XFS</th>
<th>Blockdev</th>
<th>iSCSI</th>
<th>Gluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discard</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Write zero</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Discard+zero</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Get status</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Fast convert</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Better support for preallocation

- Preallocation avoids performance problem of thin provisioning
- Three types of blocks
  - Deallocated
  - Anchored
  - Mapped
    - On disk space allocated
    - Block not in use
- Problem: limited support in Linux
Questions