The road for thin-provisioning

Paolo Bonzini
Red Hat, Inc.
November 7th, 2012

Overview

- Why is thin provisioning important?
- Current status
- Where should we go?
- Implementation overview
- Summary

Logical block provisioning

- A disk is made of many logical blocks
- The user tells the disks how it's using them
  - The user can use its allocated resources better
  - The disk gains in speed and durability
- Automatic or manual
  - mount -o discard
  - fstrim

Logical block provisioning

- Obviously extends to virtualization
  - The user is the guest administrator
  - The disk is the storage backend
- Same tools
  - Automatic management: “mount -o discard”
  - Manual management: fstrim
  - Also via guest agent

Why is it useful?

- Guest admin only pays for actually used space
- Host admin reaps all the other benefits
  - Saved disk space
  - Improved wear-leveling for SSD
  - Shorter maintainance operations

Strategies

- In the storage
  - SCSI passthrough
- In the kernel
  - Raw images or logical volumes
- In QEMU
  - QCOW2 and other image formats

SCSI logical block provisioning

VPD page 0xb2

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

UNMAP

Bit 7 Byte	6 5	4	3	2	1 0
1					ANCHOR

WRITE SAME

Bit 7 6 5 4 Byte	3 2	1 0
1 ANCHOR	UNMAP	

GET LBA STATUS

SCSI passthrough

- Entire LUNs passed to a guest via a virtual SCSI adapter
  - virtio-scsi, megasas, ...
- QEMU acts as a bridge to the LUN
  - SCSI commands passed 1:1
  - iSCSI via kernel or userspace initiator (libiscsi)
  - Other transports (SAS, FC,...) only via kernel

SCSI passthrough

- Logical block provisioning is enabled on the storage
- Mostly available on high-end disks

SCSI passthrough

- One advantage: available now :)

but...

- Needs libiscsi or CAP_SYS_RAWIO
- All maintenance is done outside QEMU
  - LUN configuration
  - Live block operations (snapshotting, migration,...)

Raw images or volumes

- Images stored on a file, partition or LV
- SCSI command set emulated by QEMU
- Limited feature set
  - No live snapshots
  - No image templates
- Easy access to underlying file system or device features

SCSI logical block provisioning

VPD page 0xb2

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

UNMAP

Bit 7 Byte	6 5	4	3	2	1 0
1					ANCHOR

WRITE SAME

Bit 7 6 5 4 Byte	3 2	1 0
1 ANCHOR	UNMAP	

GET LBA STATUS

SCSI logical block provisioning

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

$\left.\begin{array}{l}\text{Thin-provisioned}\\ \text{Resource-provisioned}\end{array}\right\}$ Logical block management enabled

- Three types of blocks
  - Deallocated
  - Anchored
  - Mapped

$\left.\begin{array}{l}\text{Anchored}\\ \text{Mapped}\end{array}\right\}$ On disk space allocated

$\left.\begin{array}{l}\text{Deallocated}\\ \text{Anchored}\end{array}\right\}$ Block not in use

SCSI logical block provisioning

UNMAP WRITE SAME	all -> any (typically: deallocated)
UNMAP WRITE SAME + anchor	all -> anchored, mapped
WRITE	all -> mapped

SCSI commands vs. system calls

	File	Block
UNMAP WRITE SAME	fallocate(FALLOC_FL_ PUNCH_HOLE)	ioctl(BLKDISCARD)
UNMAP WRITE SAME + anchor	xfsctl(XFS_IOC_ ZERO_RANGE)	?
GET LBA STATUS	lseek(SEEK_HOLE/ SEEK_DATA) ioctl(FIEMAP)	?

SCSI commands vs. system calls

	File	Block
UNMAP WRITE SAME	fallocate(FALLOC_FL_ PUNCH_HOLE)	ioctl(BLKDISCARD) fallocate
UNMAP WRITE SAME + anchor	fallocate(FALLOC_FL_ ZERO_RANGE)	ioctl(BLKANCHOR)
GET LBA STATUS	lseek(SEEK_HOLE/ SEEK_DATA) ioctl(FIEMAP)	lseek(SEEK_HOLE/ SEEK_DATA)

SCSI commands vs. QEMU block layer

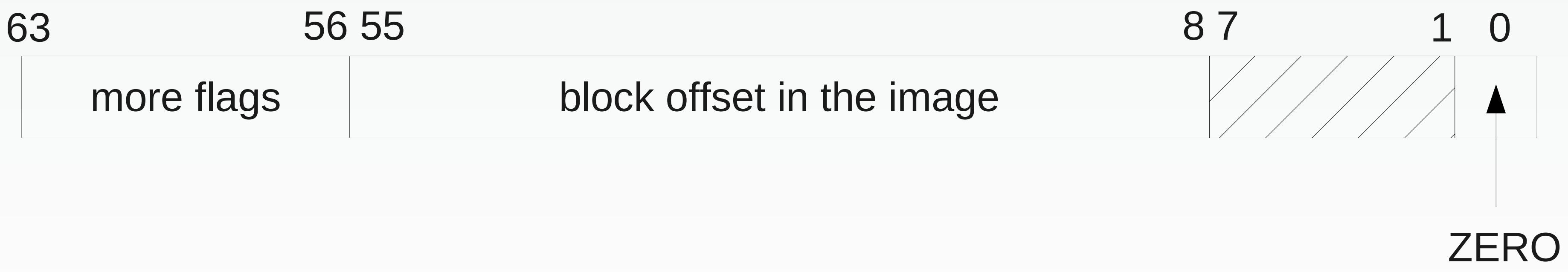
UNMAP WRITE SAME	bdrv_discard
UNMAP WRITE SAME + anchor	Not supported
GET LBA STATUS	bdrv_is_allocated (Allocated, search backing file)

SCSI commands vs. QEMU block layer

	-drive prov=thin	-drive prov=full
UNMAP WRITE SAME	bdrv_discard	bdrv_anchor
UNMAP WRITE SAME + anchor	bdrv_anchor	
GET LBA STATUS	bdrv_is_allocated (Deallocated, anchored, mapped, search backing file)	

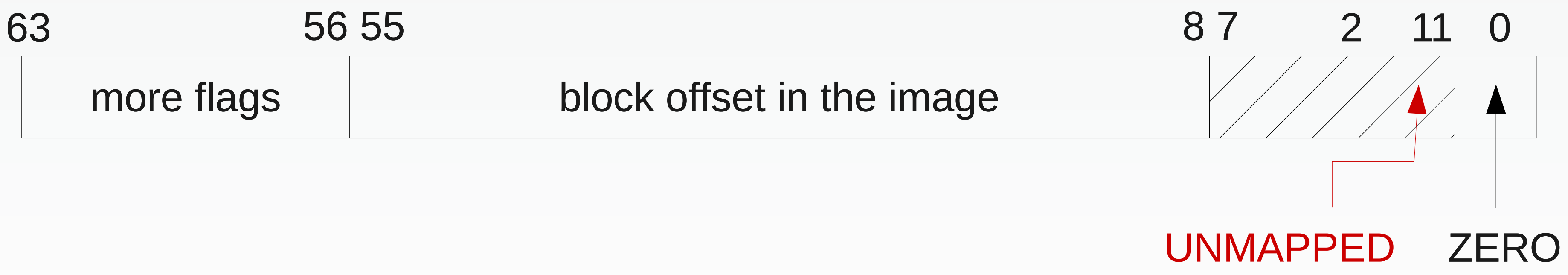
QCOW2 metadata

- “L2 table” holds pointers to data + some flags
- One entry per allocated cluster in L2 table
- Zero offset = search in backing file
- ZERO flag implies offset must be zero



QCOW2 metadata

- Add a new flag; if set, reads search backing file
  - ... even if offset is non-zero
  - ZERO clusters can have nonzero offset too
- UNMAPPED + zero offset = deallocated
- UNMAPPED + nonzero offset = anchored



Discard/anchor on QCOW2 files

Discard

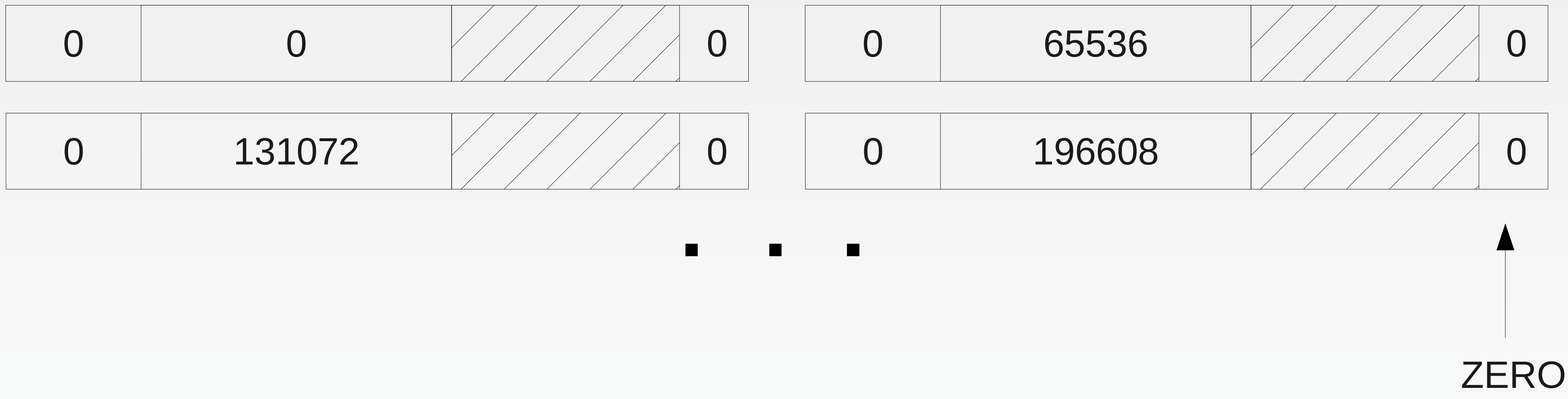
- Set unmapped bit
- Zero offset
- Add cluster to free list
- Discard the data in cluster

Anchor

- Set unmapped bit
- Anchor the data in the cluster

QCOW2 metadata preallocation

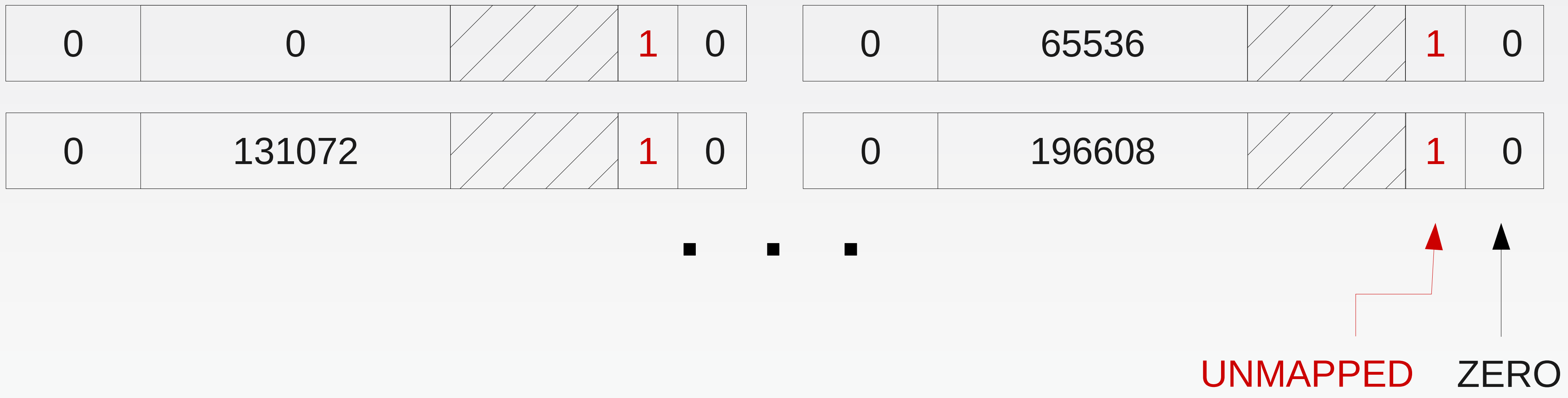
- Metadata is pre-initialized, clusters point to discarded regions (holes)



- Only possible on bottom-level images!

QCOW2 metadata preallocation

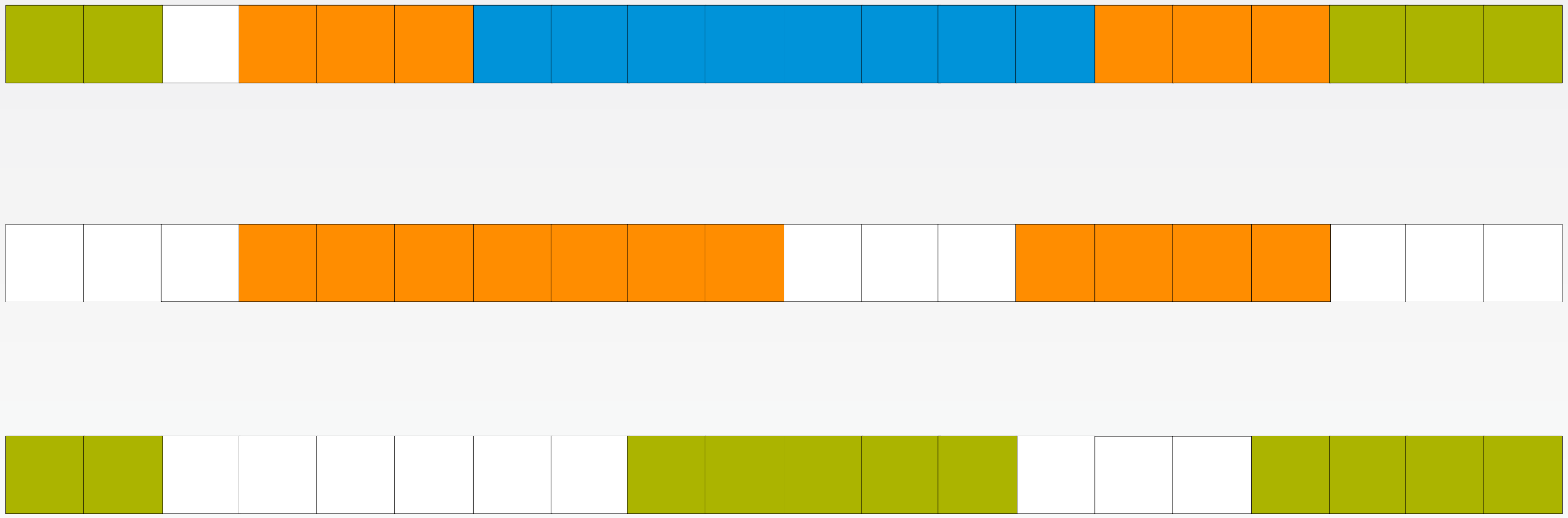
- Metadata is pre-initialized, clusters point to unmapped regions



- Now works also with a backing file!

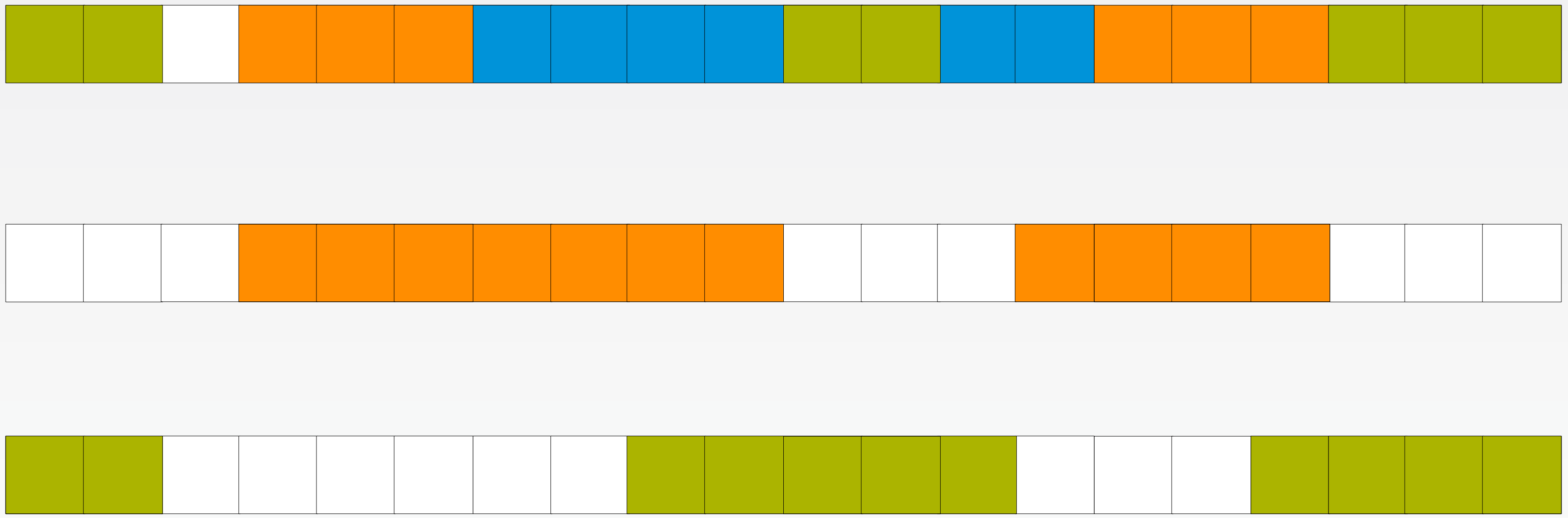
Streaming from backing files

- Data copied from backing file for faster access



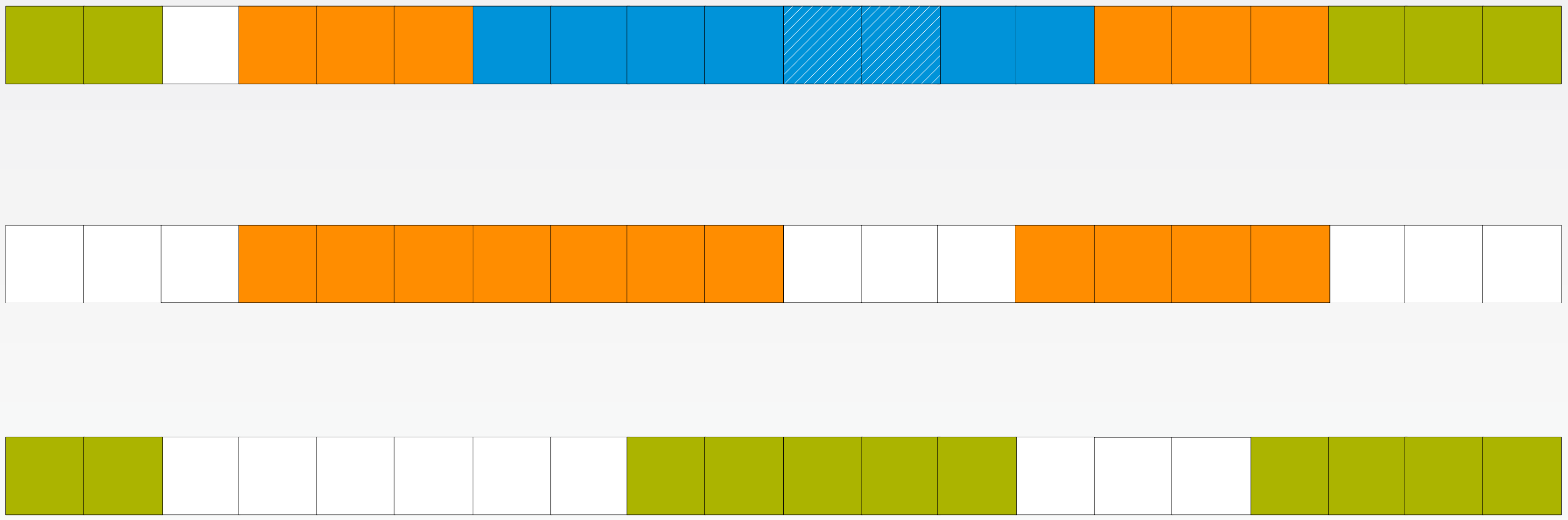
Streaming from backing files

- QEMU 1.3 lets backing file data through after discard



Streaming from backing files

- “Real” discard ignores unmapped blocks



Summary

- Benefits of logical block management:
  - Saved disk space
  - Disk durability (SSD)
  - Metadata preallocation for improved performance
  - Shorter maintainance operations
- Storage configurations supported:
  - SCSI passthrough
  - Raw images or QCOW2
  - Files, partitions, logical volumes

Todo list

Kernel:

- Simplify passthrough of UNMAP & WRITE SAME
- BLKANCHOR ioctl
- Improve lseek for block devices (GET LBA STATUS)

QEMU:

- -drive prov=...
- Discard/anchor for files and host devices
- QCOW2 discard & anchor forwarding
- QCOW2 unmapped bit support
- Optimized streaming

Questions?