Subcluster allocation for qcow2 images

KVM Forum 2020

Alberto Garcia <berto@igalia.com>

The qcow2 file format

- qcow2: native file format for storing disk images in QEMU.
- Many features: grows on demand, backing files, internal snapshots, compression, encryption...

igalia

The qcow2 file format

- qcow2: native file format for storing disk images in QEMU.
- Many features: grows on demand, backing files, internal snapshots, compression, encryption...
- **But why is it sometimes slower than a raw file?**

igalia

The qcow2 file format

- qcow2: native file format for storing disk images in QEMU.
- Many features: grows on demand, backing files, internal snapshots, compression, encryption...
- **But why is it sometimes slower than a raw file?**
- Because it is not correctly configured.

igalia

The qcow2 file format

- qcow2: native file format for storing disk images in QEMU.
- Many features: grows on demand, backing files, internal snapshots, compression, encryption...
- **But why is it sometimes slower than a raw file?**
- Because it is not correctly configured.
- Because the qcow2 driver in QEMU needs to be improved.

igalia

The qcow2 file format

- qcow2: native file format for storing disk images in QEMU.
- Many features: grows on demand, backing files, internal snapshots, compression, encryption...
- **But why is it sometimes slower than a raw file?**
- Because it is not correctly configured.
- Because the qcow2 driver in QEMU needs to be improved.
- Check my presentation at KVM Forum 2017!

igalia

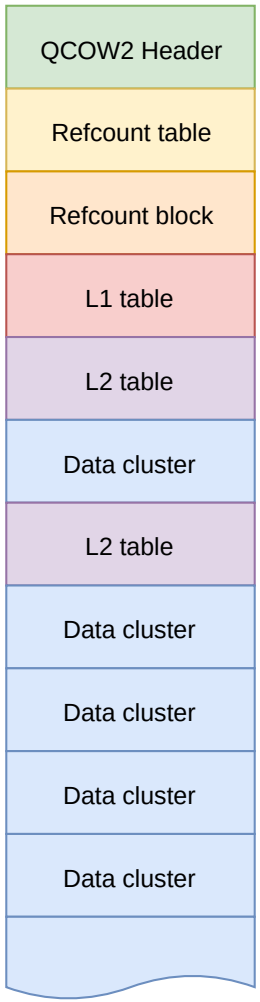
The qcow2 file format

- qcow2: native file format for storing disk images in QEMU.
- Many features: grows on demand, backing files, internal snapshots, compression, encryption...
- **But why is it sometimes slower than a raw file?**
- Because it is not correctly configured.
- Because the qcow2 driver in QEMU needs to be improved.
- Check my presentation at KVM Forum 2017!
- Because of the very design of the qcow2 file format.
  Today we are going to focus on that.

igalia

Structure of a qcow2 file

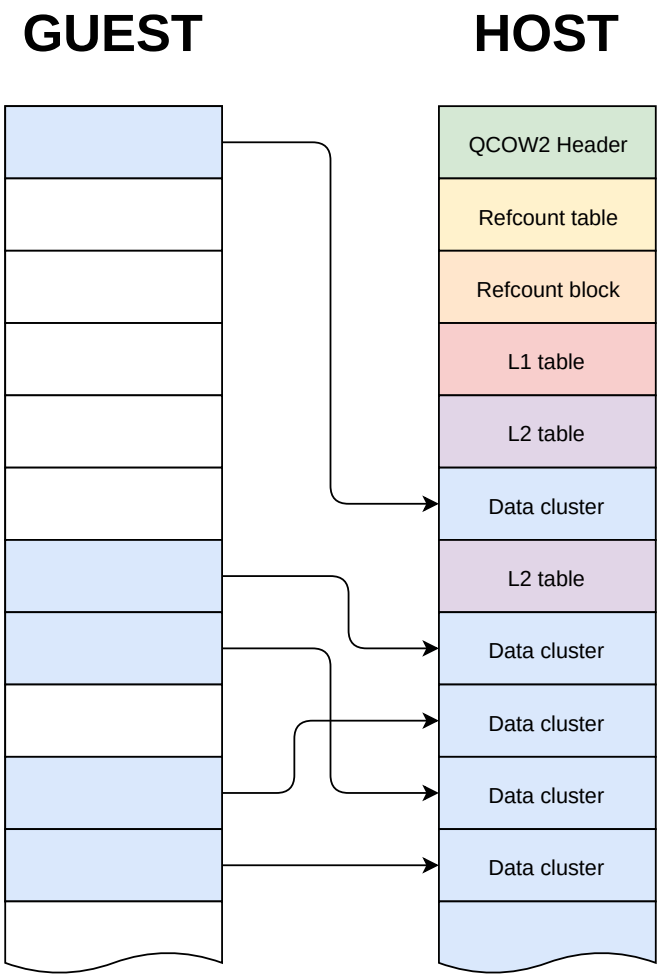
A qcow2 file is divided into clusters of equal size
(min: 512 bytes - default: 64 KB - max: 2 MB)



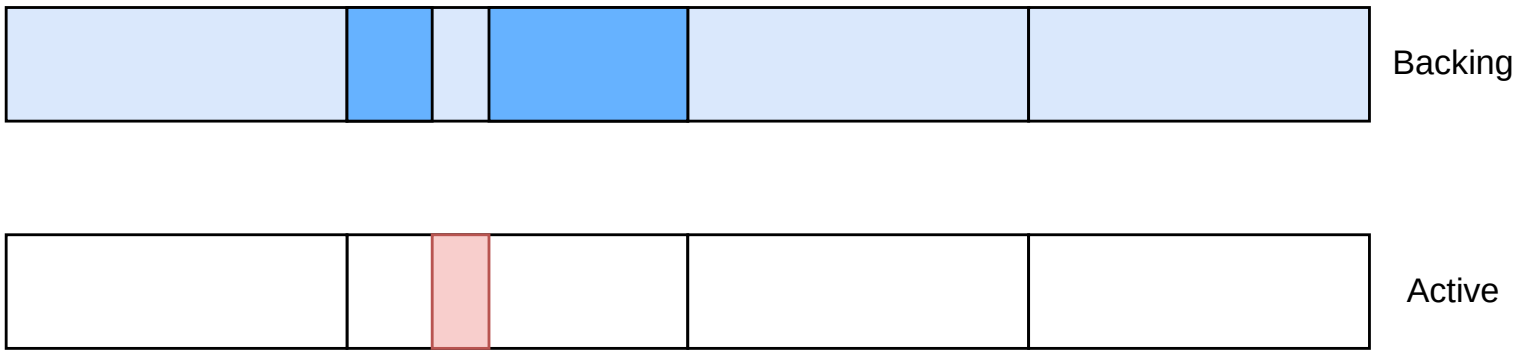
igalia

Structure of a qcow2 file

The virtual disk as seen by the VM is divided into guest clusters of the same size



Problem 1: copy-on-write means more I/O



- A data cluster is the smallest unit of allocation: writing to a new data cluster means filling it completely with data.
- If the guest write request is small, the rest must be filled with data from the backing file, or with zeroes (if there is no backing file).
- **Problem: QEMU needs to perform additional I/O to copy the rest of the data.**

Problem 1: copy-on-write means more I/O

Example: random 4KB write requests to an empty 40GB image (SSD backend)

Cluster size	With a backing file	Without a backing file*
16 KB	3600 IOPS	5859 IOPS
32 KB	2557 IOPS	5674 IOPS
64 KB	1634 IOPS	2527 IOPS
128 KB	869 IOPS	1576 IOPS
256 KB	577 IOPS	976 IOPS
512 KB	364 IOPS	510 IOPS

(*): Worst case scenario. QEMU first tries `fallocate()` which is much faster than writing zeroes

Problem 2: copy-on-write means more used space

- The larger the cluster size, the more the image grows with each allocation.
- Example: how much does an image grow after...
  - ...100 MB worth of random 4KB write requests?
  - ...creating a filesystem on an empty 1 TB image?

Cluster size	random writes	mkfs.ext4
Raw file	101 MB	1.1 GB
4 KB	158 MB	1.1 GB
64 KB	1.6 GB	1.1 GB
512 KB	11 GB	1.3 GB
2 MB	29 GB	2.1 GB

- The actual size difference in real-world scenarios depends a lot on the usage.

Decreasing the cluster size

- In summary: increasing the cluster size…
  - …results in less performance due to the additional I/O needed for copy-on-write.
  - …produces larger images and duplicate data.

Decreasing the cluster size

- In summary: increasing the cluster size…
  - …results in less performance due to the additional I/O needed for copy-on-write.
  - …produces larger images and duplicate data.
- Then let's just decrease the cluster size, right?

Decreasing the cluster size

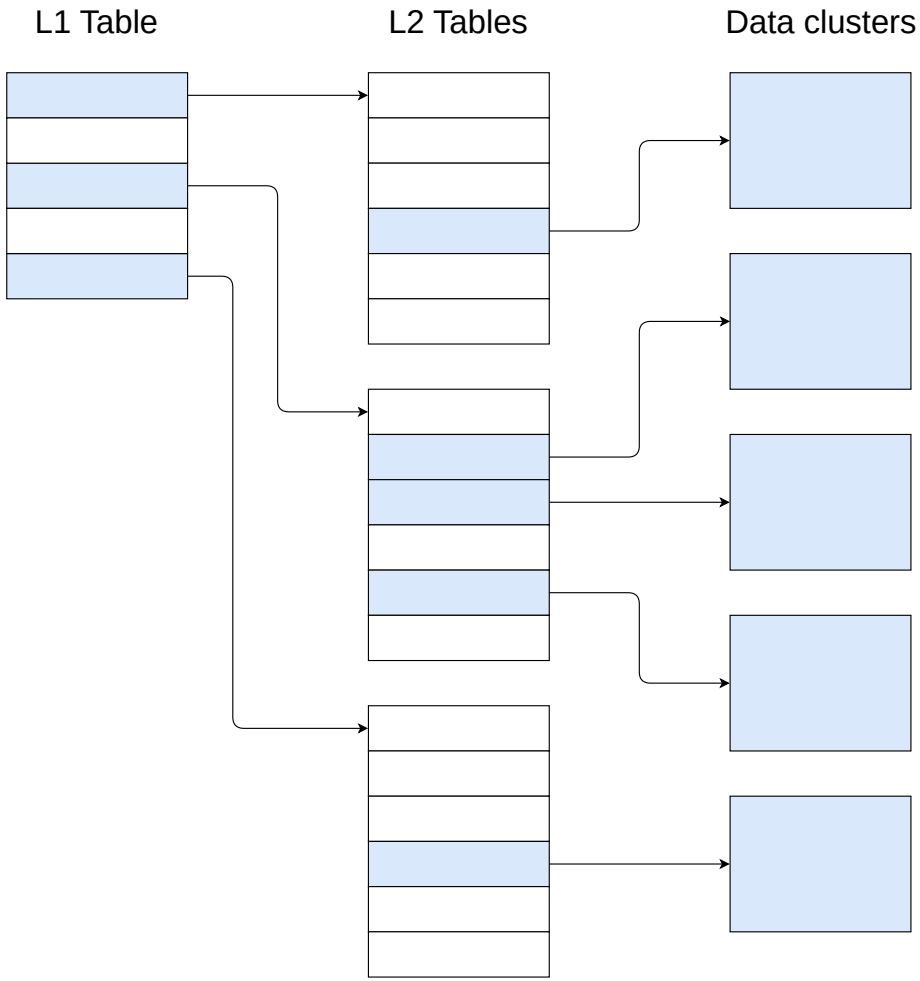
- In summary: increasing the cluster size…
  - …results in less performance due to the additional I/O needed for copy-on-write.
  - …produces larger images and duplicate data.
- Then let's just decrease the cluster size, right?
- **Not so easy: smaller clusters means more metadata**

Problem 3: Smaller clusters means more metadata

- Apart from the guest data itself, qcow2 images store some important metadata:
  - Cluster mapping (L1 and L2 tables).
  - Reference counts.
- If we have smaller clusters we'll end up having more of them, and this means additional metadata.

L1 and L2 tables

The L1 and L2 tables map guest addresses as seen by the VM into host addresses in the qcow2 file

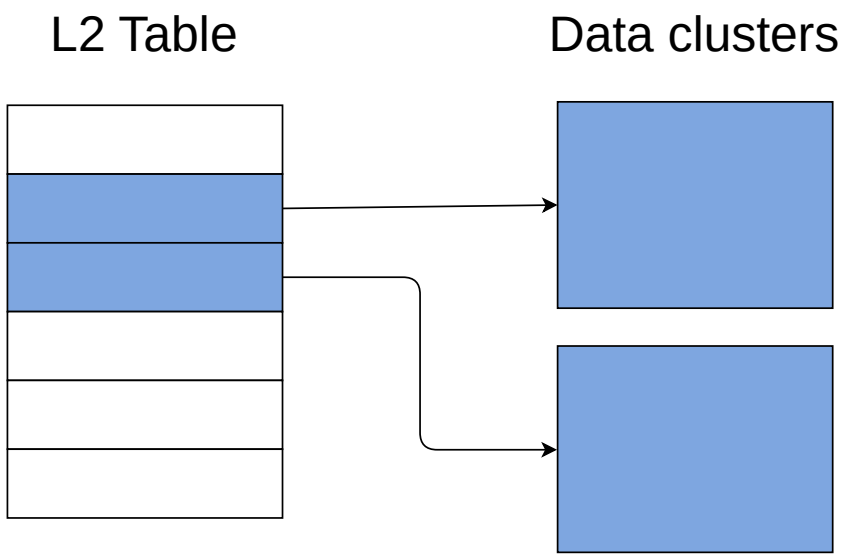


The L1 table

- There is only one L1 table per image (per snapshot, actually).
- The L1 table has a variable size but it's usually small.
  - Example: 16KB of data for a 1TB image (using the default settings).
- It is stored contiguous in the image file.
- QEMU keeps it in memory all the time.
- 64-bit entries: each contains a pointer to an L2 table.

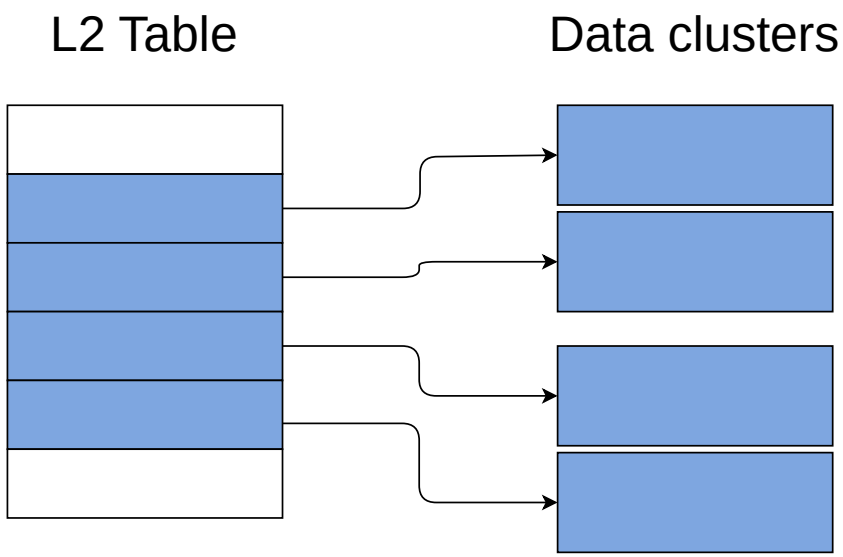
L2 tables

- There are multiple L2 tables and they are allocated on demand as the image grows.
- Each table is exactly one cluster in size.
- 64-bit entries: each contains a pointer to a data cluster.
- If we reducing the cluster size by half we need twice as many L2 entries.
- Graphically:



L2 tables

- There are multiple L2 tables and they are allocated on demand as the image grows.
- Each table is exactly one cluster in size.
- 64-bit entries: each contains a pointer to a data cluster.
- If we reducing the cluster size by half we need twice as many L2 entries.
- Graphically:



L2 metadata size

This is the maximum amount of L2 metadata needed for an image with a virtual size of 1 TB.

Cluster size	Max. L2 metadata
8 K B	1 GB
16 KB	512 MB
32 KB	256 MB
64 KB	128 MB
128 KB	64 MB
256 KB	32 MB
512 KB	16 MB
1 MB	8 MB
2 MB	4 MB

Accessing L2 metadata

- Each time we need to access a data cluster (read or write) we need to go to its L2 table to get its location.
- This is one additional I/O operation per request: severe impact in performance.
- We can mitigate that by keeping the L2 tables in RAM. QEMU has an L2 cache for that purpose.
  - Example: random 4K reads on a 40GB image:

L2 cache size	Average IOPS
1 MB	8068
2 MB	10606
5 MB	41187

- Again, reducing the cluster size by half implies:
  - Twice as much L2 metadata.
  - Twice as much RAM for the L2 cache.

Reference counts

- Each cluster in a qcow2 image has a reference count (all types, not just data clusters).
- They are stored in a two-level structure called reference table and reference blocks. Like L2 tables, the size of a reference block is also one cluster.
- Allocating clusters has the additional overhead of updating their reference counts.
- With a smaller clusters we need to allocate more of them.

igalia

The overhead of having to allocate clusters

Overall, smaller clusters are faster to fill with data, but if they get too small the overhead of the allocation process exceeds the benefits.

Cluster size	Write IOPS
512 KB	364 IOPS
256 KB	577 IOPS
128 KB	869 IOPS
64 KB	1634 IOPS
32 KB	2557 IOPS
16 KB	3600 IOPS
8 KB	758 IOPS
4 KB	97 IOPS
2 KB	77 IOPS
1 KB	62 IOPS

The situation so far

- We cannot have too big clusters because they waste more space and increase the amount of I/O needed for allocating clusters.
- We cannot have too small clusters because they increase the amount of metadata, which has a negative impact in performance and/or memory usage.
- This is a direct consequence of the design of the qcow2 format.

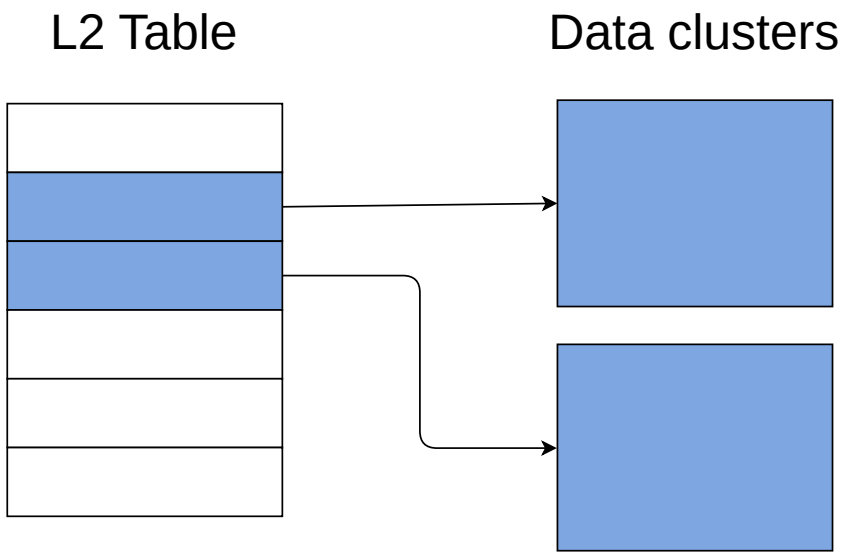
Subcluster allocation

- I'm presenting a mixed approach to mitigate this problem: **subcluster allocation**.
- In short:
  - We have big clusters in order to reduce the amount of metadata in the image.
  - Each one of the clusters is divided into 32 subclusters that can be allocated separately. This means faster allocations and reduced disk usage.

igalia

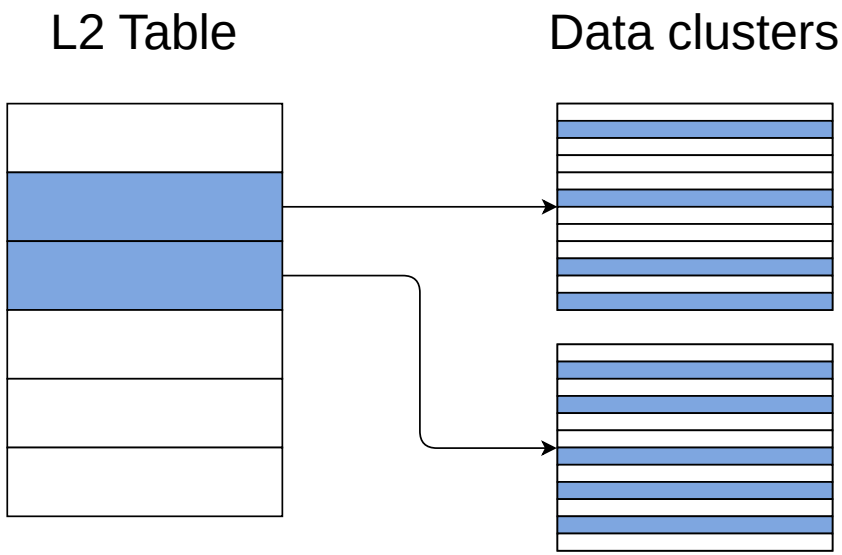
Subcluster allocation: what it looks like

A standard L2 table with entries and their data clusters



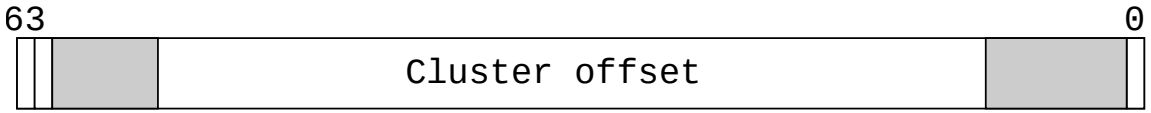
Subcluster allocation: what it looks like

An extended L2 table with subcluster allocation



L2 tables in detail

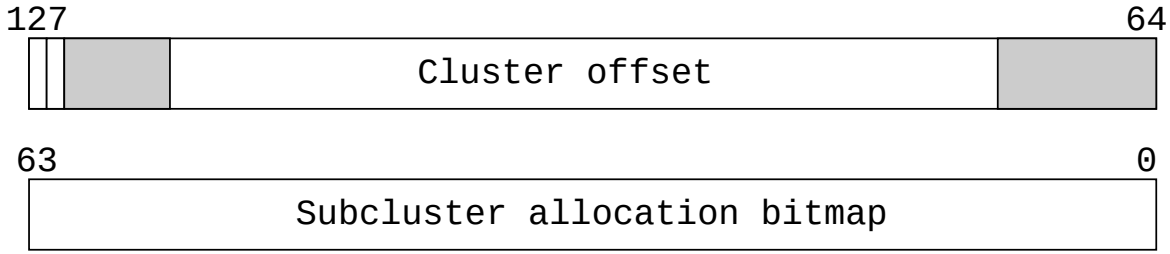
- Each L2 table contains a number of entries that look like this:



- Each cluster has one of these states:
  - Unallocated.
  - Allocated (normal or compressed).
  - All zeroes.
- Now we also need to store information for each subcluster.

Extended L2 entries

- We are adding **extended L2 entries**, which contain a 64-bit bitmap indicating the status of each subcluster.



- Each individual subcluster can be allocated, unallocated or "all zeroes".
- Compressed clusters don't have subclusters and work the same as before.

Two use cases for subcluster allocation

- Case 1: Having very large clusters in order to minimize the amount of metadata while reducing the amount of duplicated data and I/O.
- Case 2: Having smaller clusters to minimize the amount of copy-on-write and get the maximum I/O performance.

igalia

Results 1: less copy-on-write means faster I/O

- Having less copy-on-write improves the allocation performance.
- If subcluster size = request size no copy-on-write is needed!
- Average IOPS of random 4KB writes:

With a backing file		
Cluster size	**Without subclusters**	**With subclusters**
16 KB	3600 IOPS	8124 IOPS
32 KB	2557 IOPS	11575 IOPS
64 KB	1634 IOPS	13219 IOPS
128 KB	869 IOPS	12076 IOPS
256 KB	577 IOPS	9739 IOPS
512 KB	364 IOPS	4708 IOPS
1 MB	216 IOPS	2542 IOPS
2 MB	125 IOPS	1591 IOPS

Results 1: less copy-on-write means faster I/O

- Having less copy-on-write improves the allocation performance.
- If subcluster size = request size no copy-on-write is needed!
- Average IOPS of random 4KB writes:

Without a backing file*		
Cluster size	**Without subclusters**	**With subclusters**
16 KB	5859 IOPS	8063 IOPS
32 KB	5674 IOPS	11107 IOPS
64 KB	2527 IOPS	12731 IOPS
128 KB	1576 IOPS	11808 IOPS
256 KB	976 IOPS	9195 IOPS
512 KB	510 IOPS	7079 IOPS
1 MB	448 IOPS	3306 IOPS
2 MB	262 IOPS	2269 IOPS

(*): Worst case scenario. QEMU first tries `fallocate()` which is much faster than writing zeroes

Results 2: less copy-on-write means less used space

- Repeating the earlier test: how much does an image grow after...
  - ...100 MB worth of random 4KB write requests?
  - ...creating a filesystem on an empty 1 TB image?

Cluster size	random writes	mkfs.ext4
Raw file	101 MB	1.1 GB
64 KB	111 MB (vs 158 MB)	1.1 GB
512 KB	404 MB (vs 11 GB)	1.1 GB (vs 1.3 GB)
2 MB	1.6 GB (vs 29 GB)	1.1 GB (vs 2.1 GB)

igalia

Results 3: larger clusters mean less metadata

- Extended L2 entries are twice as large but each one of them references 32 subclusters.
- As a result we have **16 times less** metadata for the same unit of allocation.
- This table compares the amount of L2 metadata for a 1TB image.

Standard L2 entries	
Cluster size	**Max. L2 size**
4 KB	2 GB
8 KB	1 GB
16 KB	512 MB
32 KB	256 MB
64 KB	128 MB

Extended L2 entries	
Subcluster size	**Max. L2 size**
4 KB	128 MB
8 KB	64 MB
16 KB	32 MB
32 KB	16 MB
64 KB	8 MB

Caveats

- This feature is useful during allocation. Writing to already allocated areas won't be faster.
- Don't use it with compressed images.
  - Extended L2 entries are twice as big but offer no benefits for compressed clusters.
- If your image does not have a backing file maybe you won't see any speed-up!
  - Copy-on-write of empty clusters is already fast if the filesystem supports it.
  - However you still get the other advantages of using subclusters.
- You won't be able to read the image with older versions of QEMU (and don't expect backports!).

igalia

Implementation status

- Not available in any QEMU release yet.
- Expected in QEMU 5.2 around December.
- The implementation is complete, it is already in the repository and it is ready to be tested.
- Simply build a recent QEMU from git and create a qcow2 image with `-o extended_l2=on`.
  - Note: the default cluster size is still 64 KB. You probably want to create an image with `cluster_size=128k` or more!
- Feedback, bug reports, etc., are very much appreciated!
  - `qemu-block@nongnu.org`

igalia

Acknowledgments

This work was sponsored by

igalia