How to Handle Globally Distributed QCOW2 Chains?

Eyal Moscovici & Amit Abir
Oracle-Ravello

About Us

- Eyal Moscovici
  - With Oracle Ravello since 2015
  - Software Engineer in the Virtualization group, focusing on the Linux kernel and QEMU

- Amit Abir
  - With Oracle Ravello since 2011
  - Virtual Storage & Networking Team Leader

Agenda

- Oracle Ravello Introduction
- Storage Layer Design
- Storage Layer Implementation
- Challenges and Solutions
- Summary

Oracle Ravello - Introduction

- Founded in 2011 by Qumranet founders, acquired in 2016 by Oracle
- Oracle Ravello is a **Virtual Cloud Provider**
- Allows seamless “**Lift and Shift**”:
  - Migrate on-premise data-center workloads to the public cloud
- No need to change:
  - The VM images
  - Network configuration
  - Storage configuration

Migration to the Cloud - Challenges

- Virtual hardware
  - Different hypervisors have different virtual hardware
  - Chipsets, disk/net controllers, SMBIOS/ACPI and etc.
- Network topology and capabilities
  - Clouds only support L3 IP-based communication
  - No switches, VLANs, Mirror-ports and etc.

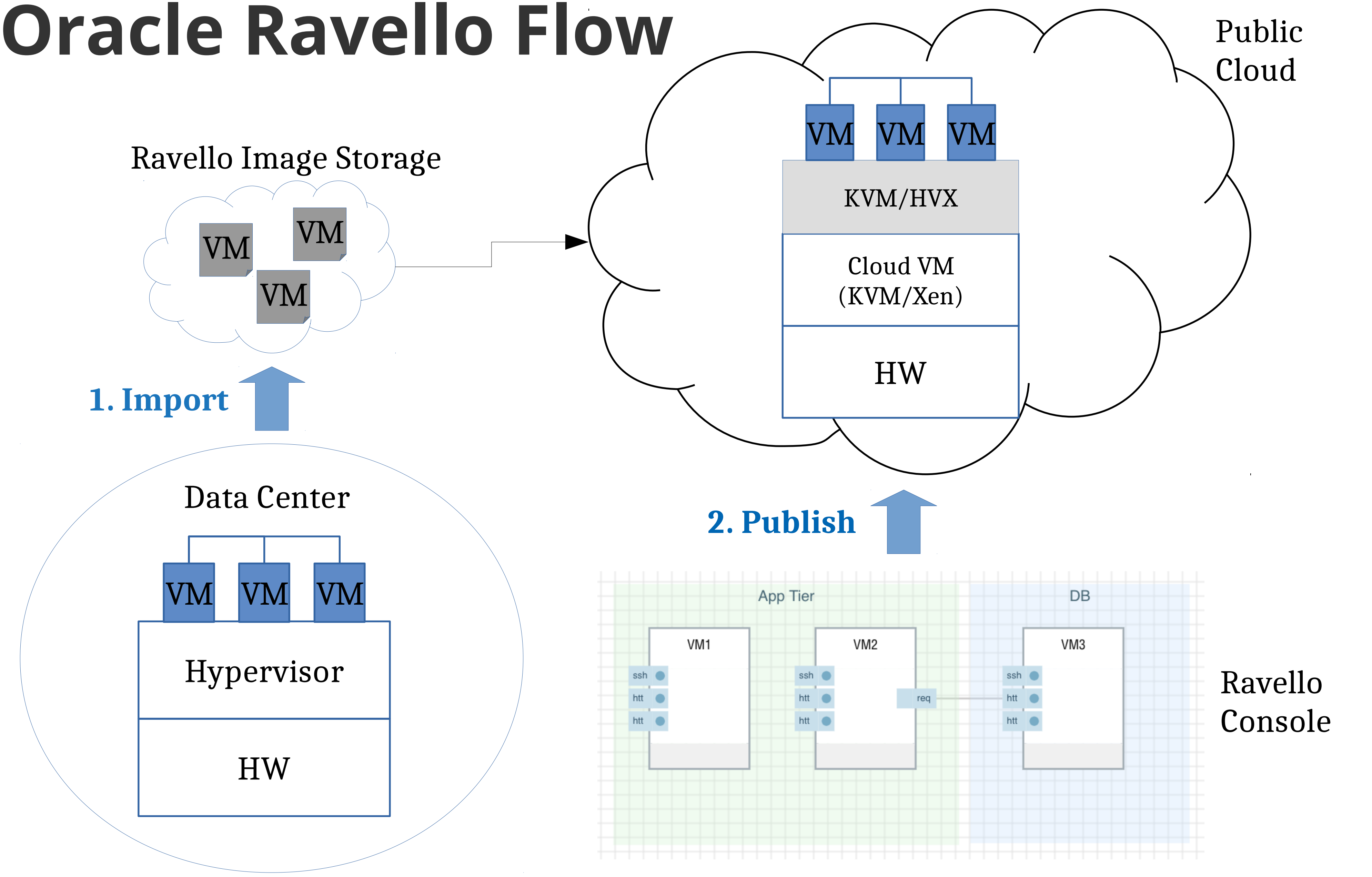
Virtual hardware support

- Solved by **Nested Virtualization:**
  - **HVX:** Our own binary translation hypervisor
  - **KVM:** When HW assist available
- Enhanced **QEMU, SeaBIOS & OVMF** supporting:
  - i440bx chipset
  - VMXNET3, PVSCSI
  - Multiple Para-virtual interfaces (including VMWare backdoor ports)
  - SMBIOS & ACPI interface
  - Boot from LSILogic & PVSCSI

Network capabilities support

- Solved by our Software Defined Network - **SDN**
- Leveraging **Linux SDN components**
  - Tun/Tap, TC Actions, Bridge, eBPF and etc.
- Fully distributed network functions
  - Leverages **OpenVSwitch**

Oracle Ravello Flow

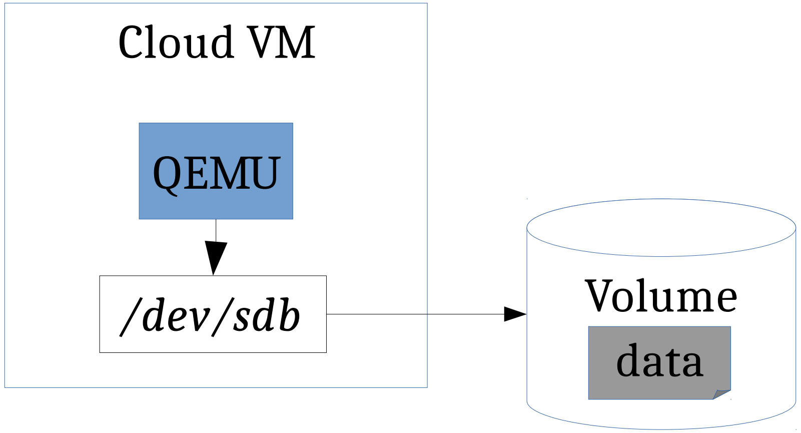


Storage Layer - Challenges

- Where to place the VM disks data?
- Should support multiple clouds and regions
- Fetch data in real time
- Clone a VM fast
- Writes to the disk should be persistent

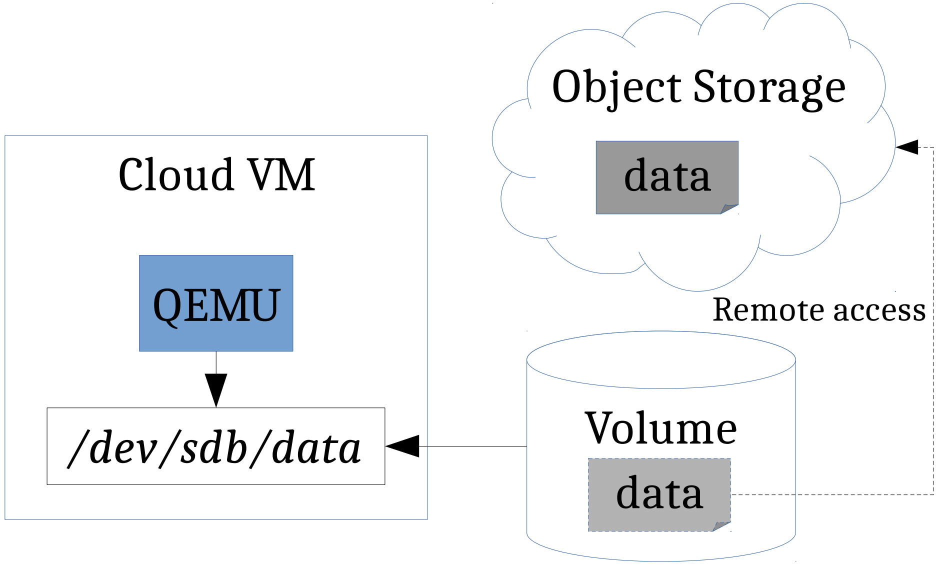
Storage Layer – Basic Solution

- Place the VMs disk images directly on cloud volumes (EBS)
- Advantages:
  - Performance
  - Zero time to first byte
- Disadvantages:
  - Cloud and region bounded
  - Long cloning time
  - Too expensive



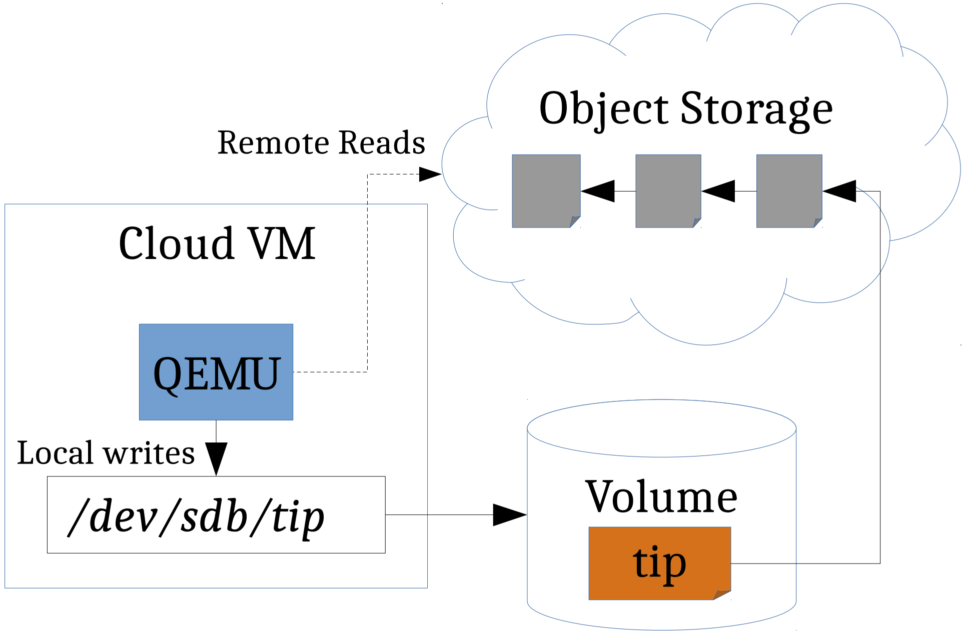
Storage Layer – Alternative Solution

- Place a raw file in the cloud object storage
- Advantages:
  - Globally available
  - Fast cloning
  - Inexpensive
- Disadvantages:
  - Long boot time
  - Long snapshot time
  - Same sectors stored many times



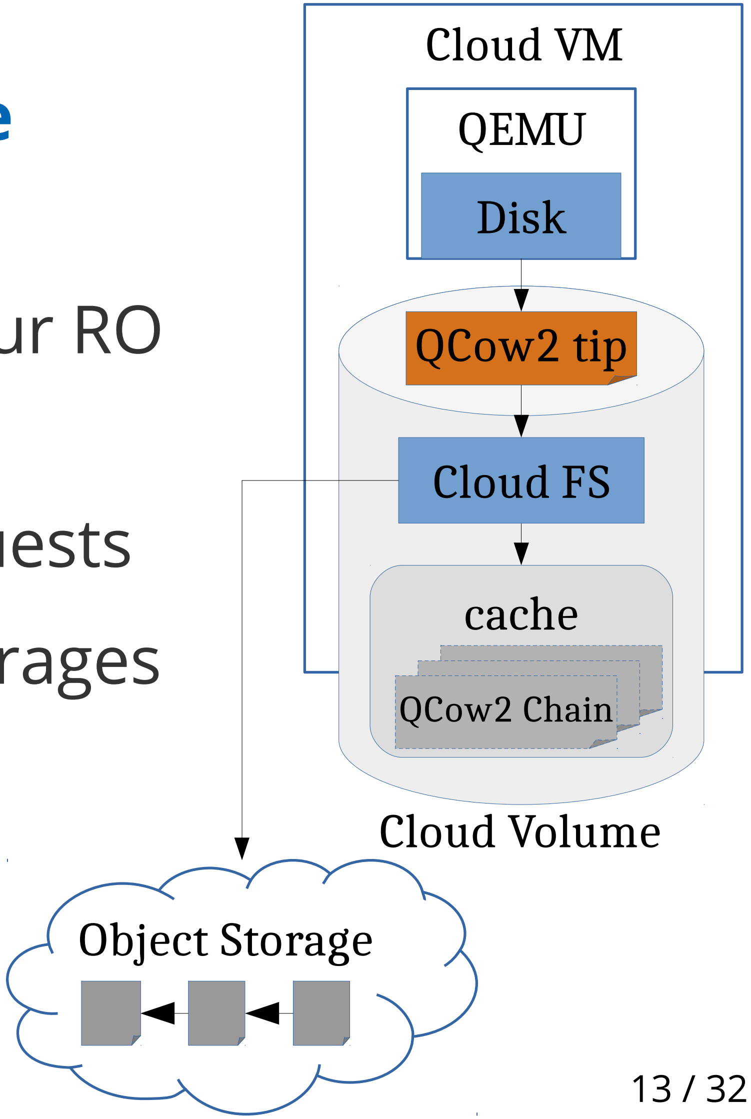
Storage Layer – Our Solution

- Place the image in the object storage and upload deltas to create a chain
- Advantages:
  - Boot starts immediately
  - Store only new data
  - Globally available
  - Fast cloning
  - Inexpensive
- Disadvantages:
  - Performance penalty

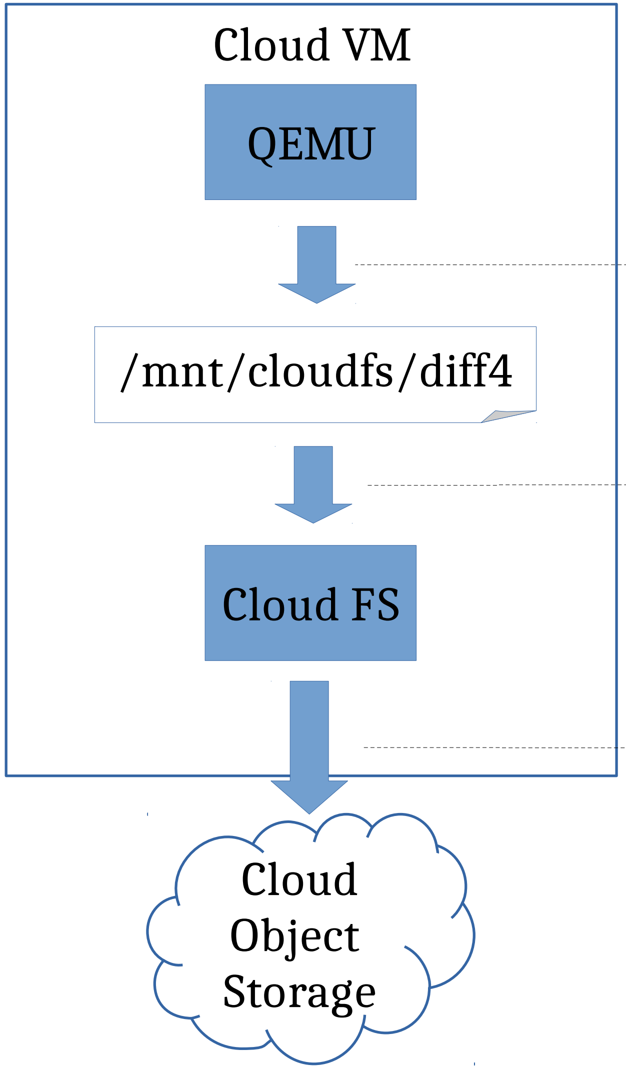


Storage Layer Architecture

- VM disk is backed by a **QCow2 image chain**
- Reads are performed by **Cloud FS**: Our RO storage layer file system
  - Translates disk reads to HTTP requests
  - Supports multiple cloud object storages
  - Caches read data locally
  - Fuse based



CloudFS - Read Flow



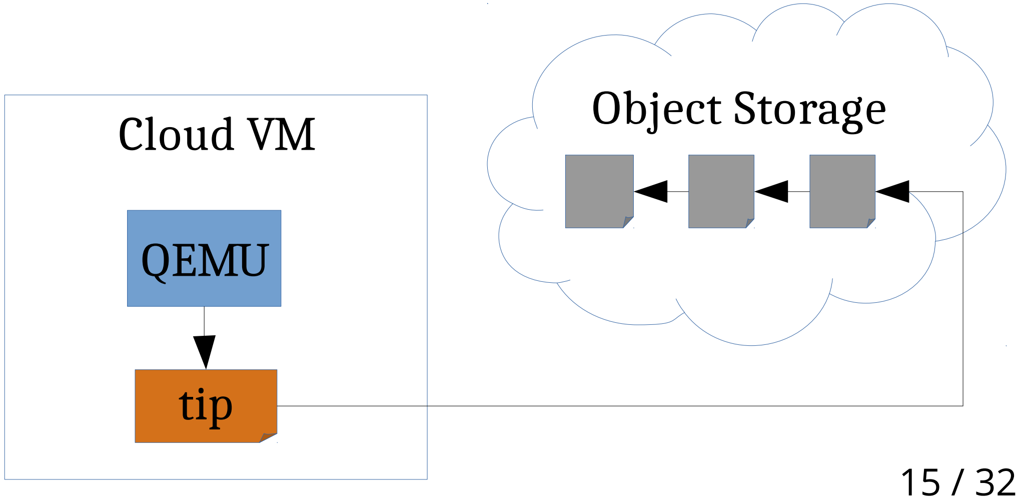
```
read("/mnt/cloudfs/diff4", offset=1024, size=512, ...)
```

```
fuse_op_read("/mnt/cloudfs/diff4", offset=1024, size=512...)
```

```
GET /diff4 HTTP/1.1
Host: ravello-vm-disks.s3.amazonaws.com
x-amz-date: Wed, 18 Oct 2017 21:32:02 GMT
Range: bytes=1024-1535
```

CloudFS - Write Flow

- A new tip to the QCow chain is created: *qemu-img create*
  - Before a VM starts
  - Before a snapshot (using QMP): *blockdev-snapshot-sync*
- The tip is uploaded to the cloud storage:
  - After the VM stops
  - During a snapshot

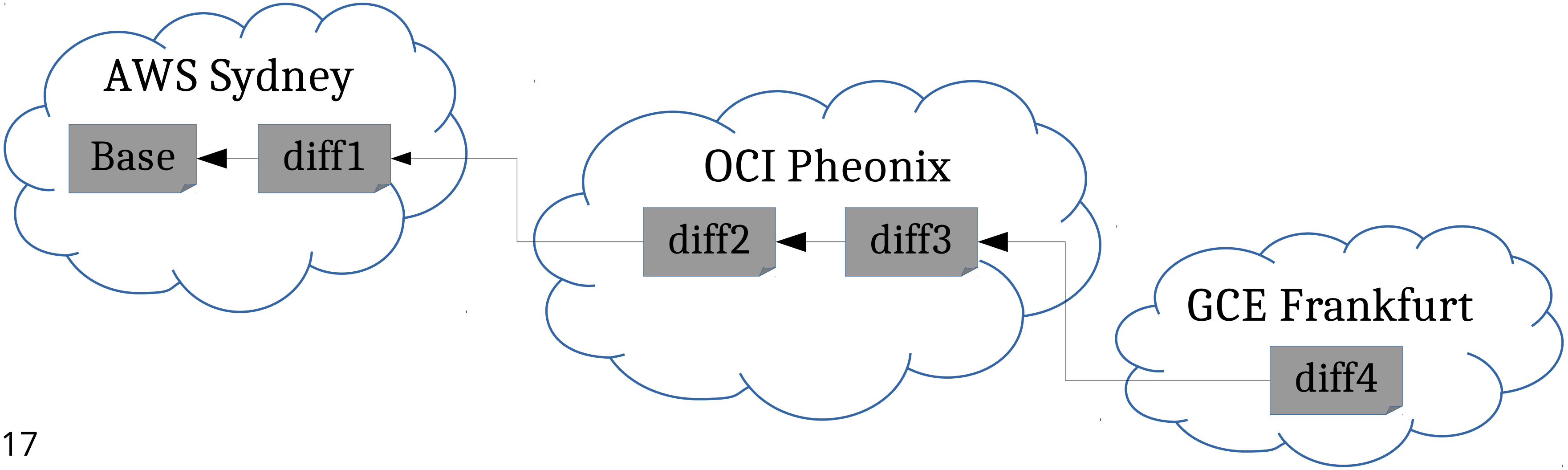


Accelerate Remote Access

- Small requests are extended to 2MB requests
  - Assume data read locality
  - Latency vs. Throughput
  - Experiments show that 2MB is optimal
- QCow2 chain files have random names
  - They hit different cloud workers for cloud requests

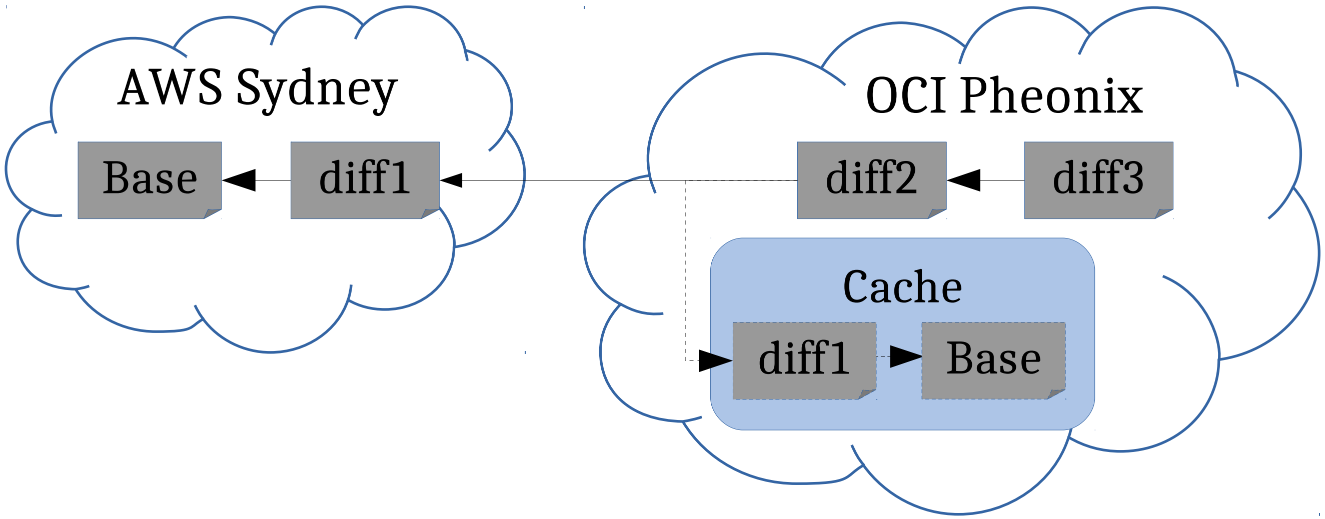
Globally Distributed Chains

- A VM can start on any cloud or region
- New data is uploaded to the same local region
  - Data locality is assumed
- Globally distributed chains are created
- **Problem:** Reading data from remote regions could be long



Globally Distributed Chains - Solution

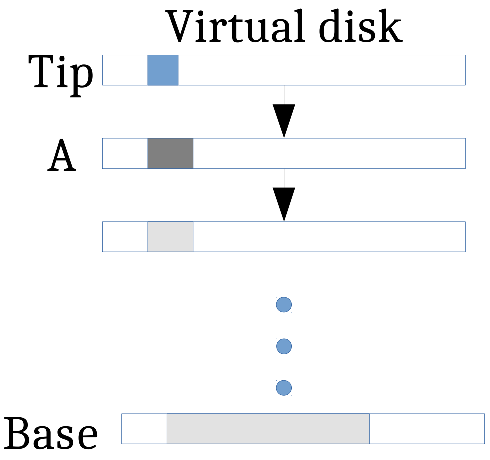
- Every region has its own cache for parts of the chain from different regions
- The first time the VM starts in a new region – every remote sector read is copied to the regional cache



Performance Drawbacks of QCow Chains

- QCow keeps minimal information about the entire chain its backing file
  - QEMU must “walk the chain” to load image metadata (L1 table) to RAM
- Some metadata (L2 tables) is spread across the image
  - A single disk read creates multiple random remote reads of metadata from multiple remote files
- *qemu-img* commands work on the whole virtual disk
  - Hard to bound execution time

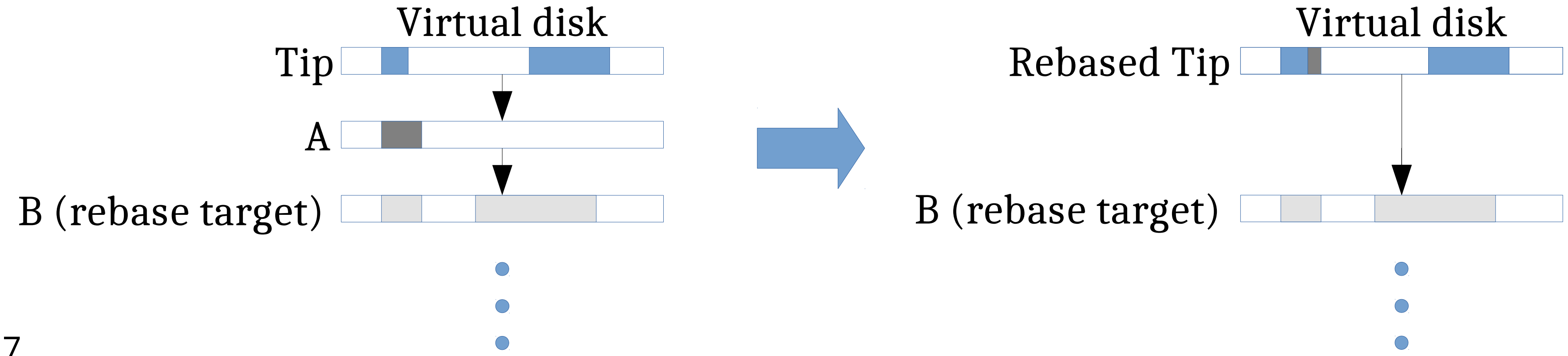
Keep QCow2 Chains Short



- A new tip to the QCow chain is created:
  - Each VM starts
  - Each snapshot
- **Problem:** Chains are getting longer!
  - For Example: a VM with 1 Disks that started 100 times has a chain 100 links deep.
- Long chains cause:
  - High latency: Data/metadata read requires to “walk the chain”
  - High memory usage: Each file has its own metadata (L1 tables).
    1MB (L1 size) * 100 (links) = 100MB per disk. Assume 10 VMs with 4 Disks each: 4G of memory overhead

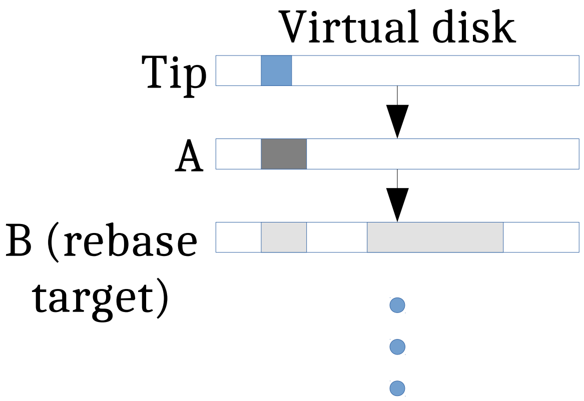
Keep QCow2 Chains Short (Cont.)

- **Solution:** merge tip with backing file before upload
  - Rebase the tip over the grandparent.
  - Only when backing file is small (~300MB) to keep snapshot time minimal
- This is done live/offline:
  - **Live:** using QMP *block-stream* job command
  - **Offline:** using *qemu-img rebase*



qemu-img rebase

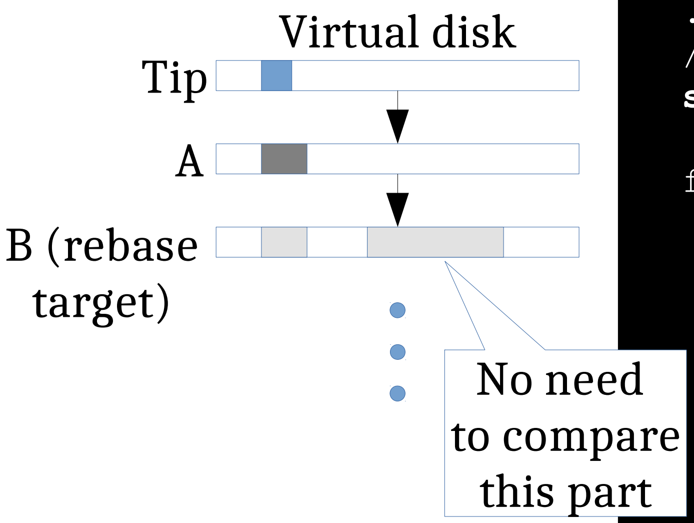
- **Problem:** per-byte comparison between **ALL** allocated sectors not present in tip
  - Logic is different then QMP block-stream rebase
  - Requires fetching these sectors



```
static int img_rebase(int argc, char **argv)
{
  ...
  for (sector = 0; sector < num_sectors; sector += n) {
    ...
    ret = blk_pread(blk_old_backing,
              sector << BDRV_SECTOR_BITS,
              buf_old, n << BDRV_SECTOR_BITS);
    ...
    ret = blk_pread(blk_new_backing,
              sector << BDRV_SECTOR_BITS,
              buf_new, n << BDRV_SECTOR_BITS);
    ...
    while (written < n) {
      if (compare_sectors(buf_old + written * 512,
          buf_new + written * 512, n - written, &pnum)) {
        ret = blk_pwrite(blk,
                  (sector + written) << BDRV_SECTOR_BITS,
                  buf_old + written * 512,
                  pnum << BDRV_SECTOR_BITS, 0);
        }
        written += pnum;
    }
  }
}
```

qemu-img rebase (2)

- **Solution:** Optimized rebase in the same image chain
  - Only Compare sectors that were changed after the rebase target

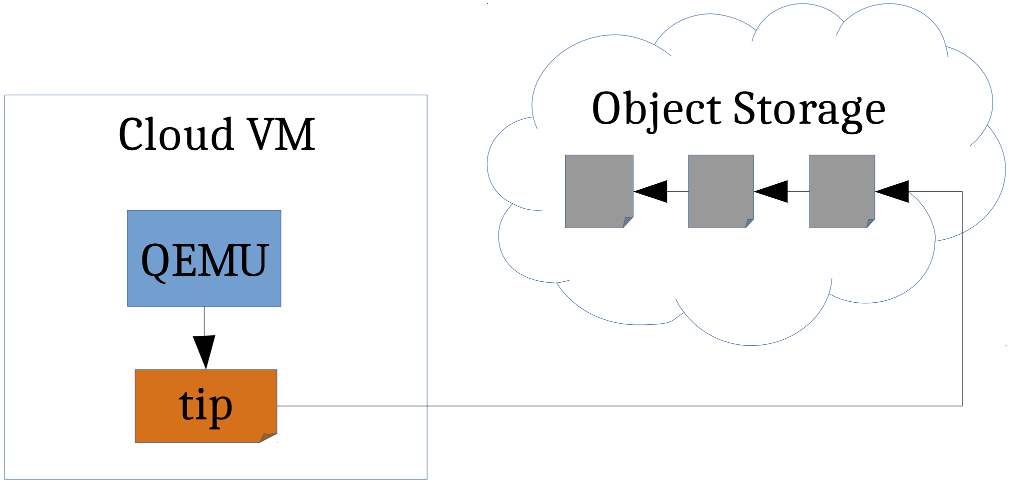


```
static int img_rebase(int argc, char **argv)
{
   ...
   // check if blk_new_backing and blk are in the same chain
   same_chain = ...

   for (sector = 0; sector < num_sectors; sector += n) {
      ...
      m = n;
      if (same_chain) {
         ret = bdrv_is_allocated_above(blk, blk_new_backing,
                                       sector, m, &m);
         if (!ret) continue;
      }
   ...
```

Reduce first remote read latency

- **Problem:** High latency on first data remote read
  - Prolongs boot time
  - Prolongs user application startup
  - Gets worse with long chains (more remote reads)

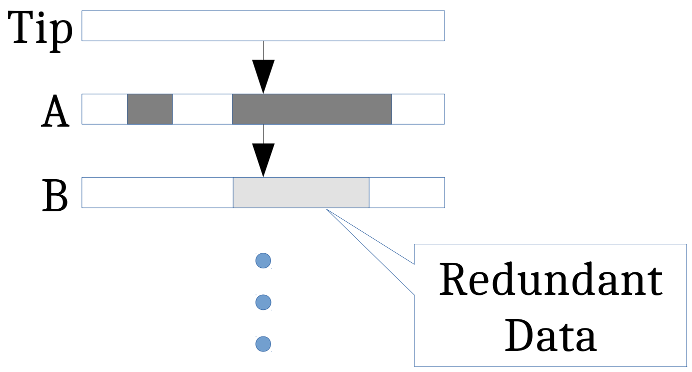


Prefetch Disk Data

- **Solution:** Prefetch disk data
  - While the VM is running, start reading the disks data from the cloud
  - Read all disks in parallel
  - Only in relatively idle times

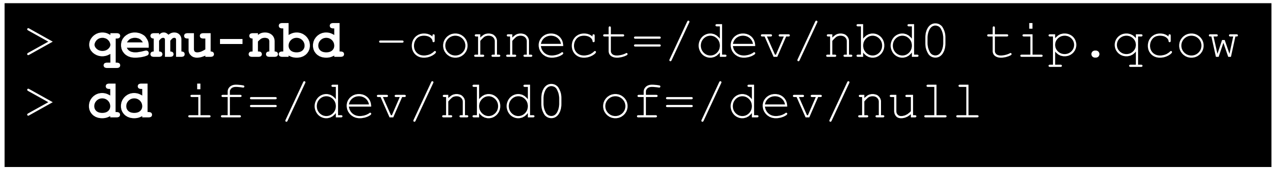
Prefetch Disk Data

- **Naive solution:** read **ALL** the files in the chain
- **Problem:** We may fetch a lot of redundant data
  - An image may contain overwritten data

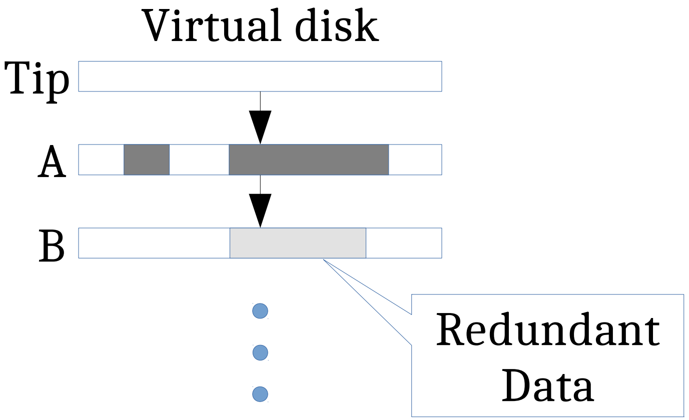


Avoid pre-fetching redundant data

- **Solution:** Fetch data from the virtual disk exposed to the guest
  - Mount the tip image as a block device
  - Read data from the block device
  - QEMU will fetch only the relavent data



```
> qemu-nbd –connect=/dev/nbd0 tip.qcow
> dd if=/dev/nbd0 of=/dev/null
```



Avoid pre-fetching redundant data (2)

- **Problem:** Reading raw block device read **ALL** sectors
  - Reading unallocated sectors wastes CPU cycles
- **Solution:** use *qemu-img map*
  - Returns a map of allocated sectors.
  - Allows us to read only allocated sectors.

```
qemu-img map tip.qcow
```

Avoid pre-fetching redundant data (3)

- **Problem:** *qemu-img map* works on the whole disk
  - Takes a long time to finish
  - We can’t prefetch data during map

Avoid pre-fetching redundant data (4)

- **Solution:** split the map of the disk
  - We added offset and length parameter to the operation
  - Bounds execution time
  - Starts prefetch data quickly

```
qemu-img map -offset 0 -length 1G tip.qcow
```

Summary

- Oracle Ravello storage layer is implemented using QCow2 chains
  - Stored on the public cloud’s object storage
- QCow2 and QEMU implementations are not ideal for our use case
  - QCow2 keeps minimal metadata about the entire chain
  - Qcow2 metadata is spread across the file
  - QEMU must often “walk the chain”
- We would like to work with the community to improve performance in usecases such as ours

Questions?

Thank you!