Physical Storage

Cloud and Virtualization Workshop

UNIVERSITY OF OREGON

NSRC
Network Startup Resource Center

Overview

What fails first?

- Drives, fans, power
- We review options for storage and hardware

Physical storage options and configuration

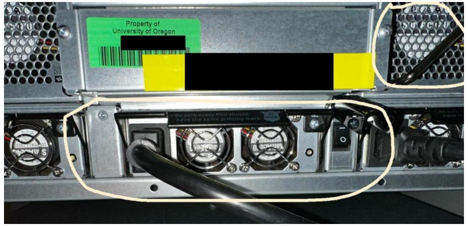
- Partitioning vs. logical volume manager
- Disk image files and formats
- ZFS with ZVOLs

Physical storage failures and options

- RAID
- ZFS
- Monitoring status

Confronting failure

- Drive errors
- Drive type
- Writing mechanisms
- Detecting failure



Dead

Dead

New

2+ dead drives 2024

Using physical storage

- Storage is the critical component of a virtual machine: persisting the VM's state and storing your application data
- Choice of storage affects the performance, cost and reliability of your system
- Storage is the part which *fails the most often** so you will have to design for this
- Quoted capacities are powers of 10
  - e.g. 500GB ≈ 500,000,000,000 bytes

* Dual power supplies are for redundant power feeds, not because power supplies are particularly unreliable!

Hard drives (HDD)

- Spinning metal platters with moveable read/write heads
  - Slow to seek to data (random access): 150 ~ 200 seeks per second. Higher rotational speed improves this a little.
  - Fast to stream sequential data
  - High capacity, low cost per byte
- Usual form factors: 3.5", 2.5"
- Usual interfaces: SATA, SAS*

*More details at https://simple.wikipedia.org/wiki/Serial_Attached_SCSI

Solid state drives (SSD, Flash)

- Silicon memory cells
  - No moving parts, but wear out after repeated writes
  - Very fast random access, fast data transfer
  - Low power consumption
- Variety of form factors and interfaces
  - 2.5", SATA / SAS
  - M.2, SATA
  - mSATA
  - M.2, NVMe
  - U.2, NVMe

Block storage internals

- HDDs and SSDs appear the same to the host system
- They are "Linear Block Accessible": read block N, write block N
  - hard drives map this internally to track / head / sector location
  - can also remap individual bad blocks to new locations
- Each block is usually 512 or 4096 bytes
  - 4096 bytes now common, reduces gaps between blocks on HDDs
- SSD internally works on "pages" of typically 128KB
  - You can write less than this, but the SSD will copy the whole 128KB to an empty page. Old pages erased in the background (garbage collection)
  - Controller spreads wear across flash pages as evenly as it can

Interfacing to block storage

- Usually via a "Host Based Adapter" (HBA) or a RAID controller
- Different versions of interfaces have different speeds
  - e.g. SATA 1 / 2 / 3 = 1.5 / 3 / 6 Gbps. Backwards compatible.
- Multiple drives can connect to the same interface
  - via "multiplier" or "expander" backplanes; they share the bandwidth
- Multiple overlapping requests can be sent to the same drive
  - For HDD: allows it to optimise head seeking
  - For SSD: allows multiple controller channels to be active (typ. 4 or 8)
  - Max total throughput when there is *concurrency* in your workload

Filesystems

- To make block storage useful, the OS creates a filesystem
  - Organizes block storage into Files, Directories, and free space
  - Provides higher level operations like "open file", "read", "write", "close"
- Examples
  - Linux: ext4, XFS, ZFS
  - Windows: NTFS
- User applications access the filesystem, not the block device
- Filesystem expects the block device to have a fixed size
  - Resizing is possible, but it is a special operation

Mounting filesystems

- Writing the initial data structure to create an empty filesystem is called "formatting", "making" or "building" the filesystem
- The OS "mounts" the filesystem to read in the metadata and start using it to read and write files
- "Unmounting" the filesystem flushes out any remaining changes
- Two OSes must not mount the same block device at the same time, or data corruption is guaranteed! *

* Unless you are using an esoteric cluster filesystem e.g. GFS, OCFS2

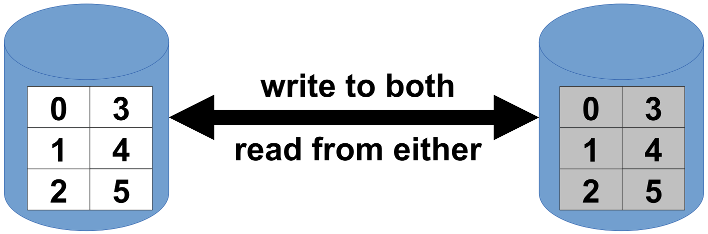
Drive failures and redundancy

Dealing with drive failures

- Both HDDs and SSDs do fail, quite frequently
- Different failure modes, including:
  - Total failure of drive (common with SSDs)
  - Failure to read parts of drive (common with HDDs)
  - Succeeds only after multiple retries (can slow the whole system down)
- Drives validate each block with a checksum (CRC)
  - Means they <u>should</u> return an error, rather than incorrect data
- To keep running, additional copies of data must be available
- On one server: RAID = "Redundant Array of Inexpensive Drives"

Mirroring, aka "RAID1"

- Store identical copies of each block on two or more drives
- Fail to read from drive 1? Then retry from drive 2
  - and write the data *back* to drive 1, so it can replace the failed block
- For writing, slightly *slower* than a single drive
- For reading, it's *faster* than a single drive
  - You have two copies of everything, so can do two reads simultaneously



Other RAID levels

- RAID5: Parity RAID. Use N+1 drives to store N blocks of data
  - the extra block is calculated across the N blocks
  - on loss of any single drive, data can be reconstructed from the others
  - lower storage overhead than mirroring, but very poor write performance
- RAID6: Use N+2 drives to store N blocks of data
  - similar, but can survive loss of any two drives
- RAID0: striping
  - Faster sequential access as N blocks are spread across N drives
  - NO REDUNDANCY. Loss of any one drive loses the entire dataset!
- RAID10 combines mirroring with striping (speed and redundancy)

*Nicely detailed explanations and diagrams at https://phoenixnap.com/kb/raid-levels-and-types

Another option: ZFS

- ZFS is a filesystem, volume manager *and* RAID combined
- Supports mirroring, raidz (=RAID5), raidz2 (=RAID6), raidz3
- Better write performance than traditional parity RAID
- Unlike other options, ZFS can *detect and correct* bad data
  - e.g. if two mirrors have differing data, it will pick the correct one
- Extremely strong data integrity guarantees
  - Meaning: if you read it from ZFS, you can be sure it's correct
  - However, it's still important to keep good backups
  - Interruption to multiple drives can cause total, irrecoverable data loss

How RAID is implemented

- "Software RAID": OS uses directly attached disks (e.g. HBA)
  - Linux: mdraid, dmraid (works with LVM), ZFS raidz
  - Modern CPUs are very fast, and code is highly optimized
- "Hardware RAID": pushes all the RAID logic into a controller card
  - Presents the whole array as one or more virtual volumes
  - Maybe faster? (arguable)
  - More magic, less visibility, special management tools required, proprietary metadata formats. Keep an identical spare controller card!
- If you're using ZFS, you must use HBA not RAID controller, or you lose ZFS's ability to repair data

RAID scrubbing

- If a disk sector goes bad, you won't know about it until you next read it. If all copies have gone bad, you're toast.
- Scrubbing: periodically read across the whole drive set, checking for reads that fail, and rewriting from redundant copies
- ZFS can also detect and repair "bit rot": when the wrong data is present, or the parity copies disagree
  - It's because ZFS stores checksums of all blocks in its data structures
  - If data can't be recovered, it reports on which files are affected
  - ZFS is the only grown-up filesystem to do this (btrfs doesn't count)

Monitoring and repair

- Properly monitoring your array is critical
  - To get notification of failed drives that need replacing
  - To identify drives with long latency or other issues
  - Use a monitoring system (nagios plugins, prometheus/node_exporter, …)
- Replacing a drive has to rewrite all data ("resilvering")
  - has a *big performance impact,* especially with parity RAID
  - can take a long time to complete
  - risk of data loss if another drive fails while this is taking place
  - risk is higher if you build arrays out of large drives, and/or many drives in a single array
  - Increasing an entire array size with new disks can take a while

SMART monitoring

- Can give some advance warning of impending drive failures
- Returns a wide range of stats from the drive; not easy to interpret
  - There is a global "Health OK"; if this says not OK, then replace
- Can request short and long self-tests on the drive
  - Long self-test can take hours to read the whole disk surface
- RAID controllers often make it difficult to access the drives directly to get SMART data
  - This is a big advantage of HBAs and software RAID

Warning: RAID is not backup!

- RAID is *only* for high availability
  - i.e. less downtime when a drive fails
- Multiple or cascading drive failures are not unknown
  - e.g. if an HBA card serving multiple drives fails
  - can cause loss of the entire array
- RAID does not protect against filesystem corruption
  - Consider RAID 1 (mirror), corrupt data is just copied twice…
- RAID does not protect against "fat fingers" or malware
  - Any data destruction is instantly replicated

Questions?

Error Recovery Control*

- Some desktop hard drives perform *infinite retries* on failed read
  - If used with RAID, a single bad sector causes the entire drive to lock up and be kicked out of the array!
- ERC means that drive gives up after a few seconds
  - RAID system can then read the data from other drive(s), and write it back to the bad drive, repairing the data
- *Essential feature.* Test each drive model before buying
  - ATA: `smartctl -l scterc /dev/sda`
  - SAS: `sdparm --get=RTL /dev/sda`

* Also known as Time Limited Error Recovery (TLER) or Command Completion Time Limit (CCTL)

What sort of drives should you buy?

- "Enterprise" drives have similar failure rates to consumer drives!
  - They *might* perform better, be better mechanically isolated, or last longer
  - They *will* have ERC (but some consumer drives do too)
  - Compromise: consider consumer "NAS" drives
- For SSDs: look at endurance figures
  - Triple Level Cell (TLC) and Quad Level Cell (QLC) store more bits in each cell, but have lower write endurance
- Under heavy write load, SSDs may start thermal throttling
  - Drastically reduces performance (factor of 10 or more!)
  - Test under real workloads, consider heatsinks and airflow improvements

TRIM / Discard

- When you delete a file, the directory is updated, but the data blocks remain on disk
  - They are added to free space list, and can be reused later
- This means that SSDs are unable to garbage collect flash pages
  - They don't know that this data is no longer required
  - Smaller pool of free pages means less efficient operation
- Solution: "TRIM" signals to the drive that block can be discarded
  - Some filesystems can do this online (be careful of bugs!)
  - Linux utility "fstrim" can be run periodically to free unused space
  - Also works with thin-provisioned VM images, if enabled in hypervisor

Consistency, performance, and caching

- When the OS or application writes data, these writes may wait around in RAM before reaching disk, and/or be reordered
  - in the guest OS (VFS cache) – as "dirty blocks" to be written later
  - in the hypervisor or host OS
  - in the HBA or RAID controller
  - in the drive itself
- If the power is pulled (or VM uncleanly killed) at the wrong time, only some of these blocks will have made it to disk
- Opportunity for filesystem to end up in an invalid state

Option 1: Write-through

- For every write, wait until the drive has confirmed it has been persisted to disk before writing the next block
- OS writes in an order which ensures the filesystem is always in a consistent state
- Problem: extremely slow
  - latency of waiting for each write to complete
  - loses optimization opportunities, e.g. combining adjacent writes

Option 2: Write-back

- Acknowledge writes as soon as they are in RAM
- Explicitly flush to disk at strategic points ("write barriers")
  - Example: journalling filesystem
  - write data to a journal, flush it, then write the data to final location
  - if data wrote to the journal, missing writes can be replayed on next startup
  - if data didn't fully write to journal, then ignore it. Partial data is lost, but at least the filesystem is in a consistent state (all-or-nothing)
- When flushing, OS waits for write barrier to complete before writing more data

Importance of write barriers

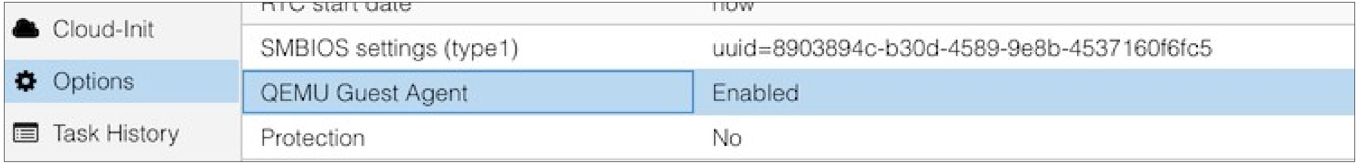
- Write barriers work very well, *if* they are implemented end-to-end
  - i.e. all the way from guest, through hypervisor/LVM/RAID layers, to disk
- Unfortunately, some badly-configured VM hosts tell lies
  - They tell the guest that the write has completed, before it really has
  - Makes write performance better, but risky for data integrity
- Some hard drives lie too
  - May be configurable with hdparm
- Some RAID controllers use battery-backed RAM so they can safely acknowledge writes before they have completed
  - to work around the poor write performance of RAID5/RAID6

Recommendations

- Don't disable write barriers (e.g. qemu's "unsafe" mode)
  - except for temporary VMs where you don't care about data loss
- Check your guest OSes use write barriers
  - if they don't, then you'll need to enable write-through in hypervisor
- Check your hardware implements write barriers correctly
  - if unsure, turn off write caches completely, especially in SATA drives
- Try to avoid unnecessary power loss on VM hosts, and unclean shutdowns of guests

Snapshots

- Taking a snapshot of a VM disk *while it is running* can also lead to inconsistencies
  - There can be data in RAM which has been partially flushed to disk
- Solution 1: make a "live" snapshot which includes the RAM state
  - Considerably larger (could be many GiB) and slower
- Solution 2: signal to the VM to freeze filesystem & flush to disk
  - Install "qemu-guest-agent" inside the VM, and enable in Proxmox
  - Recommended!



The End!