ORACLE®

I/O Topology – Getting to Know Your Storage

Martin K. Petersen <martin.petersen@oracle.com>
Consulting Software Developer, Linux Kernel Engineering

The Story So Far...

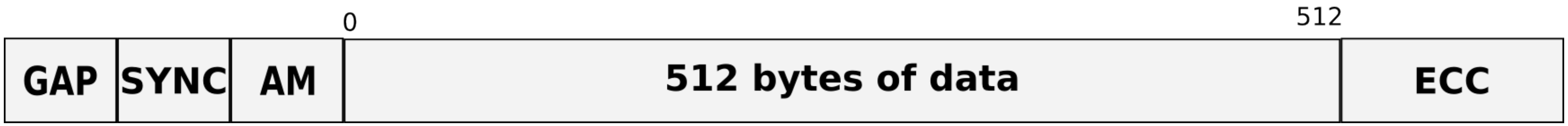
- All block devices essentially look the same
- Speak ATA or SCSI, regardless of transport (SPI, SAS, FC, USB, FireWire)
- Regardless of whether it is a cheap USB stick or a million dollar array with non-volatile cache treat them exactly the same. Like a single spindle disk drive
- Part of that abstraction has been inherent in the protocols but we're reaching the end of the useful life of that simplistic view
- It's time to get to know our storage...

Disk Drives

Disk Drives: Block Sizes and Sectors

- Most block-based storage devices use 512-byte sectors
- So far these sectors have constituted both the physical size of allocation on the disk as well as the unit used to address a particular location
- However, for RAID arrays and solid state devices the internal allocation unit is often bigger, despite pretending to be 512-byte based as viewed by the operating system
- From now on we'll have to distinguish between:
  - Physical block: The storage device's internal allocation unit
  - Logical block: The way we address a location on the device

Disk Drives: 512-byte Physical Blocks

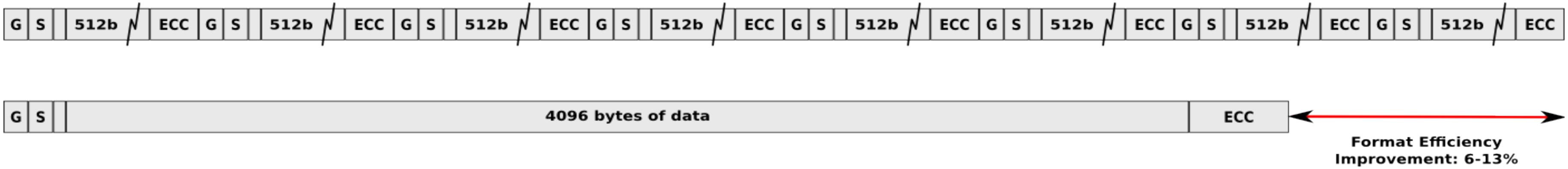


0 512

GAP | SYNC | AM | 512 bytes of data | ECC

- Each sector on disk is actually quite a bit bigger than 512 bytes thanks to fields used internally by the drive firmware
- These fields help to position the read/write head, help ensure the right location is found and contain an ECC that protects the data portion of the sector
- Together these fields eat up a lot of physical storage space and disk drive manufacturers are pretty close to the physical limits as far as track density goes
- This means the only way to increase capacity is to reduce overhead

Disk Drives: 4KB Physical Blocks


G S 512b ECC G S 512b ECC G S 512b ECC G S 512b ECC G S 512b ECC G S 512b ECC G S 512b ECC G S 512b ECC
G S 4096 bytes of data ECC

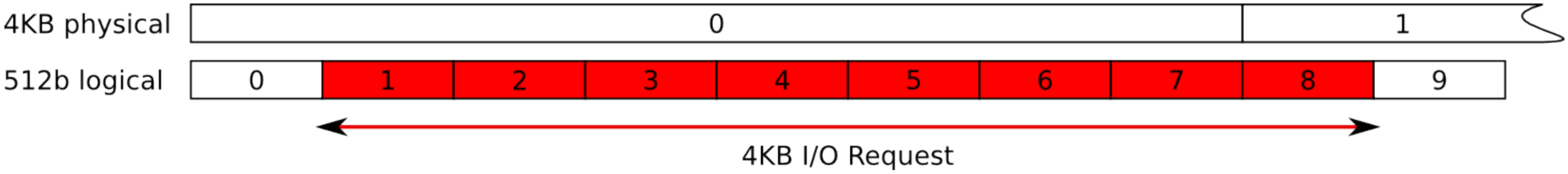
Format Efficiency Improvement: 6-13%

- The solution is to switch to 4KB blocks
- Despite potentially having multiple sync fields per blocks and a bigger ECC, there's still a substantial gain in capacity
- Most operating systems use 4KB pages and filesystem blocks so moving away from 512 byte increments is not a big deal
- However, legacy operating systems are hardwired to 512 and can not use drives which expose 4KB blocks

Disk Drives: Desktop vs. Enterprise

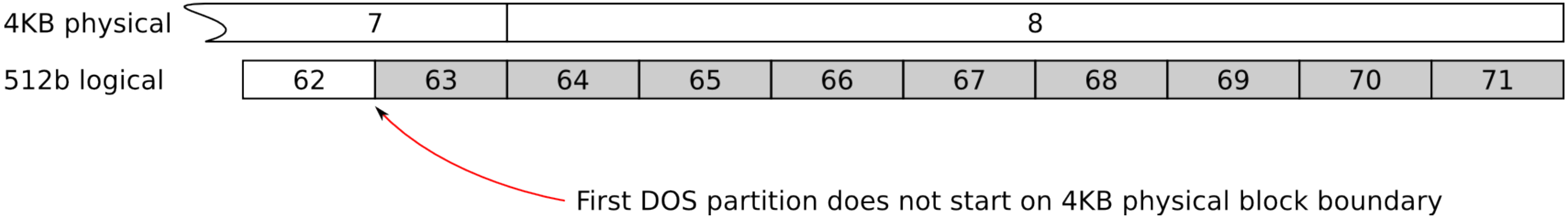
- Because of the desire to keep supporting legacy desktop operating systems, drive vendors will keep shipping drives which use 512-byte logical blocks but which use 4KB physical blocks internally
- SCSI-class drives will switch to 4KB logical *and* physical blocks because most server operating systems can handle bigger sectors just fine
- SCSI-class drives can be formatted down to 512/520/528-byte blocks with a loss in user-accessible capacity due to increased overhead

Disk Drives: Alignment



- The desktop class drives are only *emulating* 512-byte sectors. If you submit a misaligned request, the drive will have to resort to read-modify-write
- This means the platter has to do an extra revolution, inducing latency and lowering IOPS
- Vendors are working on techniques to mitigate this in drive firmware. Without mitigation, the drop in performance is quite significant

Alignment vs. DOS Partitions



First DOS partition does not start on 4KB physical block boundary

- DOS put first partition on LBA 63 by default and now we're stuck with it
- Consequently, laptop/desktop drives may ship formatted so that LBA 63 is aligned on a 4KB physical boundary to ease the pain for XP users
- Only the first partition will be naturally aligned. And only if DOS partition tables are used
- Vista and Windows 7 will align first partition on a 1MB+ε boundary

Alignment vs. Linux Partitions

- Linux has traditionally been using DOS partitions
- With 4KB drives that generally means we'll get optimal performance on `/boot` and everything else will suffer
- Thankfully both ATA and SCSI drives report alignment and Linux now retrieves this information
- physical_block_size, logical_block_size, alignment_offset are exported in /sys/block/*foo*/queue
- With this information exposed it is up to partition tools to ensure that filesystems start on a naturally aligned boundary
- fdisk/parted changes are underway

Alignment vs. MD and DM

- The block layer provides a generic device stacking function that now handles logical vs. physical block sizes and verifies compatible alignment
- This stacking function is used by both MD and DM and will warn if adding a device will cause misalignment
- It is up to the userland utilities (mdadm and dmsetup) to ensure that the beginning of the virtual block device is on a naturally aligned boundary
- DM userland changes done but not upstream yet
- mdadm in progress

Solid State Drives

Solid State Drives

- 512-byte physical blocks were a trade-off that made sense in the 80s
- However, almost everything these days is working on multiples of 4KB
- Flash chips are not an exception, most of the devices out there use 4KB pages
- Misaligned requests to a low- to mid-range SSD will suffer just like misaligned requests to a 4KB disk drive
- Thankfully SSD vendors have the option of filling out the same fields as used by disk drives
- And if they do, Linux will now do the right thing wrt. alignment

Solid State Drives

- More device characteristics are either available or in the pipeline
- One that we are using already is the rotational parameter that indicates whether we are dealing with a spinning disk or not
- The rotational parameter is not just for solid state drives. RAID arrays which often have large caches can set it too and we can avoid optimizing head movement across a platter that does not exist

RAID Arrays

RAID Arrays

- Arrays generally use blocks bigger than 512 byte internally. 4, 16, 64 KB are normal. Sometimes even bigger
- This means a small I/O request will cause read-modify-write cycle
- And writes smaller than the stripe size will cause a parity update which may incur another penalty
- The SCSI protocol has been expanded with knobs that tell us the array's preferred I/O granularity for random I/O as well as the optimal sustained I/O size
- These usually correspond to chunk and stripe size respectively

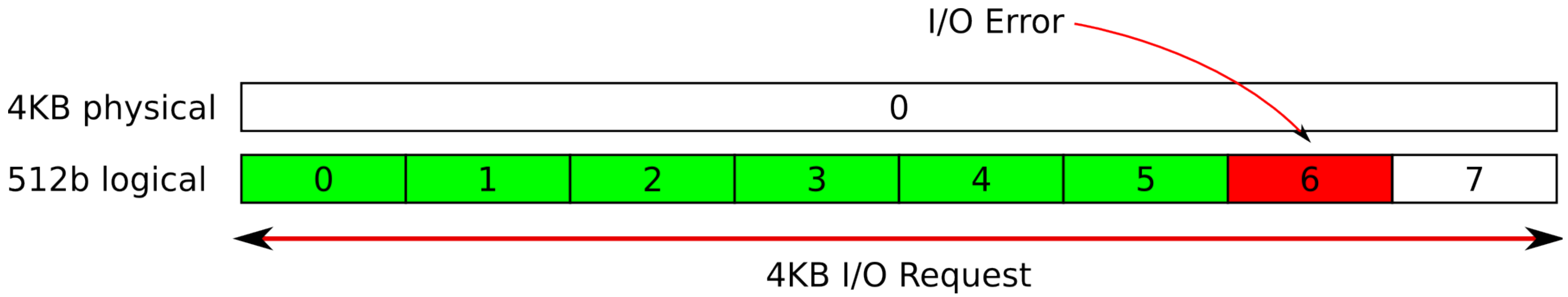
RAID Arrays

- Linux will now query and export these values in /sys/block/*foo*/queue/minimum_io_size and optimal_io_size
- Filesystems can use these to chose block and allocation sizes and to naturally align data and metadata
- These I/O hints are actually provided for *all* block devices, not just for hardware RAID arrays that explicitly report them.
- DM and MD fill them out according to the RAID level in question
- Look, Ma! No hacky ioctls! One-stop shopping for filesystem utilities

Data Integrity

- This is not just about performance. In some cases alignment is an absolute requirement for correctness!

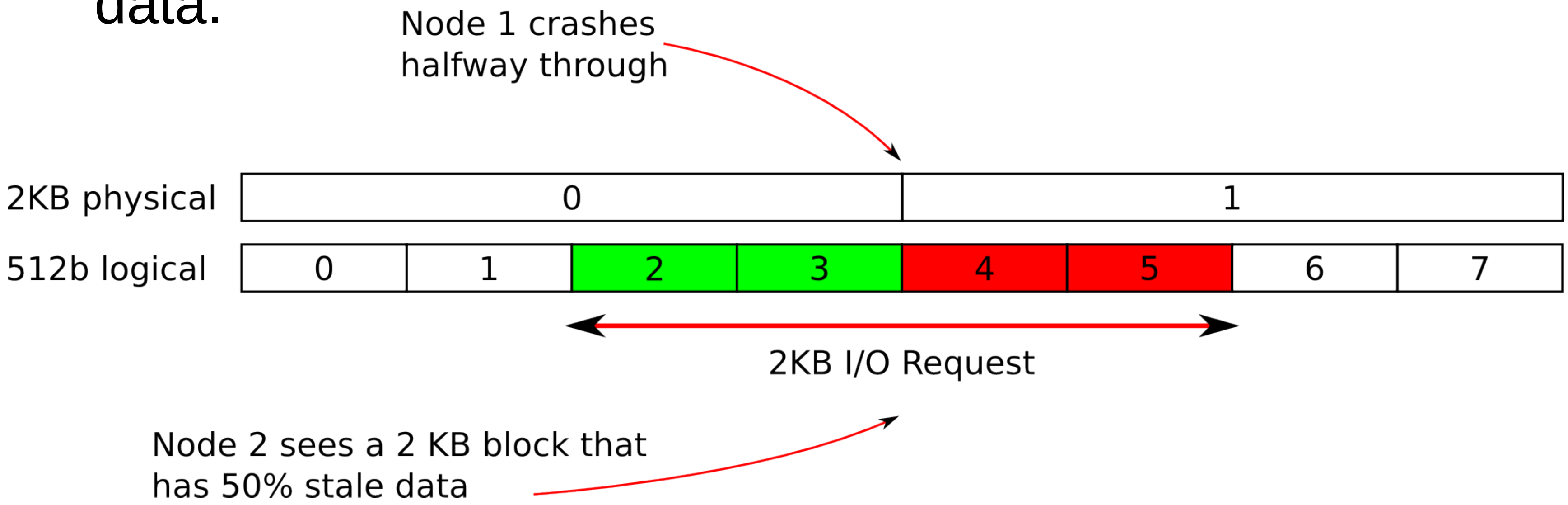
  Example: 512-byte logical / 4KB physical drive experiencing a write error may invalidate logical blocks that have been previously acknowledged as written to the kernel:



Data Integrity

- Shared storage setups are common in the enterprise

  Example: Imagine an array with 2KB physical blocks shared between two nodes. Node 1 crashing during a write may cause Node 2 to see stale/inconsistent data.



Node 1 crashes halfway through

2KB physical	0				1			
512b logical	0	1	🟩 2	🟩 3	🟥 4	🟥 5	6	7

2KB I/O Request

Node 2 sees a 2 KB block that has 50% stale data

Conclusion

- I/O topology changes merged in 2.6.31
  - Common interface for all block devices
  - ATA + SCSI devices supported
  - MD and most of DM support has landed
  - DM userland utilities done
  - mdadm in next on my list
  - fdisk & parted need work