Proceedings of the Linux Symposium

July 13th–17th, 2009
Montreal, Quebec
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*

Programme Committee

Andrew J. Hutton, *Steamballoon, Inc., Linux Symposium, Thin Lines Mountaineering*
James Bottomley, *Novell*
Bdale Garbee, *HP*
Dave Jones, *Red Hat*
Dirk Hohndel, *Intel*
Gerrit Huizenga, *IBM*
Alasdair Kergon, *Red Hat*
Matthew Wilson, *rPath*

Proceedings Committee

Robyn Bergeron
Chris Dukes, *workfrog.com*
Jonas Fonseca
John 'Warthog9' Hawley

With thanks to
John W. Lockhart, *Red Hat*

Authors retain copyright to all submitted papers, but have granted unlimited redistribution rights to all as a condition of submission.

I/O Topology

Martin K. Petersen
Oracle
martin.petersen@oracle.com

Abstract

The smallest atomic unit a storage device can access is called a sector. With very few exceptions, a sector size of 512 bytes has been akin to a mathematical constant in the storage industry for decades. That picture is now rapidly changing with hard drives moving to 4KB sectors. Flash-based solid state drives and enterprise RAID arrays also have alignment and block size requirements above and beyond what we have traditionally been honoring.

This paper will present a set of changes that expose the characteristics of the underlying storage to the Linux kernel. This information can be used by partitioning tools and filesystem formatters to lay out data in an optimal fashion.

1 Disk Drives and Block Sizes

Until recently what has been commonly referred to as *sector size* has been both the unit used by the programming interface to address a location on disk as well as the size used internally by the drive firmware to organize user data.

In the never-ending quest for increased capacity, disk drive manufacturers are now switching to a 4KB sector size, or *physical block size*. This allows them to increase yield due to less overhead per sector (see Figure 1). I.e. more of the physical capacity can be used for user data as opposed to sync marks, error correction and other fields used internally by the drive firmware.

The industry migration to bigger sectors is just beginning and is scheduled to complete in 2011. Enterprise drives are expected to switch directly to 4KB sectors but can be formatted down to 512-byte blocks at a slight loss in user-visible capacity.

For compatibility with legacy operating systems such as Windows XP, desktop and laptop class disks will continue to present 512-byte sectors to the operating system despite using 4KB blocks internally. This means that we will continue to use increments of 512 bytes to address data on disk. In protocol parlance this is referred to as the *logical block size*. We refer to a sector offset on disk as the *logical block address* or *LBA*.

The backwards compatibility comes at a cost. If the operating system submits a request smaller than 4KB, or if the request submitted is misaligned and straddles two physical blocks, the drive firmware will have to perform a read-modify-write cycle. This incurs a significant performance penalty as the drive will have to perform an extra platter rotation. First the partial sector needs to be read into a buffer, then new data added, and then the same location will need to come back under the head to get written.

For large sequential writes the impact of the read-modify-write cycle is fairly small as only the first and last physical block will be affected. However, small random write workloads are significantly slowed down so it is imperative to prevent misalignment.

2 Partitions

Linux normally uses 4KB filesystem blocks and consequently writes smaller than the physical block size are rare. However, we need to make sure that the filesystem is layed out so that the filesystem blocks are aligned with the physical blocks of the underlying disk. Aligning on a 4KB boundary may seem like a trivial task at first but once again backwards compatibility considerations mean that special care must be taken.

Traditionally Linux, like Windows, has been using the DOS partition table format. This format defaults to putting the first partition at sector 63 which is not on

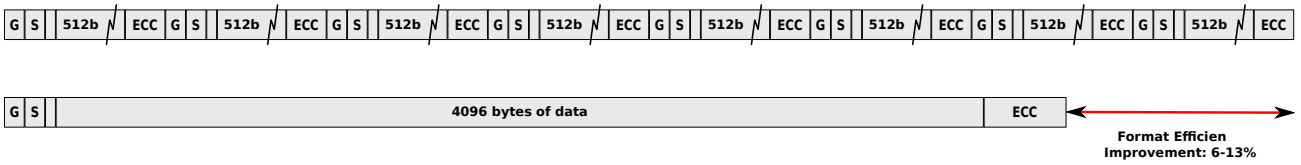

Figure 1: Top line illustrates how 512-byte sectors are stored on disk: Gap, Sync & Address Mark fields, followed by 512 bytes of data and finally an ECC. 8 sectors are required to store 4KB of user data. On the bottom line: A drive with 4KB sectors causes much less overhead.

a 4KB boundary. And consequently all I/O to that partition would be misaligned causing a performance degradation.

Because it is impossible to retroactively change Windows XP to align on something other than sector 63, the drive manufacturers instead opted to ensure that LBA 63 is aligned on the physical block size boundary. This in turn means that the lowest naturally aligned LBA is 7.

There is no guarantee that any subsequent partitions will be aligned on a 4KB boundary, neither from the beginning of the drive, nor from the beginning of the first partition. This means that drive firmware artificially aligning LBA 63 only helps Windows users as XP rarely uses more than one partition. With Linux, however, the first partition is usually used for the `/boot` filesystem where performance is less important.

To make things even more complicated, enterprise class drives are as mentioned above switching to 4KB sectors as well. SAS and Fibre Channel drives will use 4KB logical and physical block sizes. SATA Nearline drives will use 4KB physical and 512-byte logical blocks. In both cases LBA 0 will be naturally aligned, so we can not simply assume that LBA 63 is always aligned on a physical block boundary.

3 Alignment and Block Size Reporting

To remedy this both the SCSI and ATA protocols have been expanded with fields that inform us of the drive's alignment. For SCSI drives physical block size and alignment are reported in the `READ CAPACITY(16)` command. And for ATA the same values can be found in `IDENTIFY` words 106 and 209 respectively.

Until now the Linux block layer has used the queue parameter `hardsect_size` to indicate the sector size for an underlying device. In 2.6.31 this value has been deprecated in favor of `logical_block_size` and `physical_block_size` respectively. Both values are exported to user applications via sysfs.

Alignment is also reported, both for the whole block device as well as for each partition. The byte offset from the underlying drive's physical block alignment can be found in sysfs' `alignment_offset` parameter.

These three values enable tools such as `fdisk` and `parted` to align properties on a natural boundary, preventing the read-modify-write cycle and the resulting performance degradation.

4 Virtual Block Devices

A significant amount of Linux deployments use either software RAID via the MD driver or logical volume management via the DM driver. In both cases it is crucial to ensure that the virtual block devices exported by the drivers will have their first LBA naturally aligned to the underlying storage.

Virtual block device drivers have traditionally used a stacking function provided by the block layer to ensure that various limits such as the sector size were compatible with the underlying storage. This stacking function has been extended so that alignment and physical block size are taken into account when devices are layered. The drivers can pass in an offset to the stacking function to compensate for space used by their internal superblocks.

The stacking function will make sure that all component devices use compatible alignment and physical block sizes. It even handles corner cases such as combining mismatched devices for example a 512-byte sector drive and a 4KB ditto in a RAID1 setup. In this case the alignment of the 512-byte drive will be scaled up to match that of the 4KB drive.

As it is the case with low-level block devices, both MD and DM will expose the block size and `alignment_offset` parameters in sysfs, meaning that filesystem utilities no longer have to have special cases for extracting this type of information for MD and DM devices. All block devices now export exactly the same set of characteristics.

5 Performance Hints

Filesystems such as XFS are designed to be aware of the topology of the underlying storage. When an XFS filesystem is created on top of an MD or LVM device it will query the device to figure out the stripe chunk size as well as the stripe width. XFS will then lay out its important data structures on stripe boundaries.

So far topology reporting has been done using either `ioctl` calls or by wrapping LVM command line tools and parsing their output. However, these approaches are quite inflexible and restricted to virtual block devices.

Hardware RAID arrays have provided means to extract this type of information as well but in a vendor-specific, proprietary fashion. As such it has been up to the system administrator to query the storage device and pass the appropriate layout parameters to the `mkfs` utility.

To remedy this a recent addition to the SCSI Block Commands specification permits devices to export performance characteristics using a common mechanism known as the Block Limits VPD page. Support for this VPD has been added to the SCSI disk driver in 2.6.31 and Linux will now export the values in sysfs if the storage device reports them.

The MD and DM drivers have also been updated and export their respective stripe chunk and stripe width sizes using the same sysfs files. This means that filesystem utilities can gain access to the device performance hints without employing MD and DM specific code.

The first hint is `minimum_io_size` which is the optimal request size granularity for the device, typically the RAID chunk size. A properly aligned multiple of this value is the preferred request size for workloads where a high number of I/O operations per second are desired.

The other hint is `optimal_io_size` which corresponds to the optimal unit of sustained I/O. Typically this is the stripe width for RAID arrays. A properly aligned multiple of this value is the preferred request size for workloads where a high throughput is desired.

6 Solid State Drives

Hard drive performance is heavily constricted by the speed at which the read/write heads can be moved across the platter as well as rotational latency. Low-end disk drives spin at 5400 or 7200rpm, whereas enterprise drives spin at 10000 or 15000 rotations per minute, but despite this, a contemporary drive can usually only service between 100 and 250 *IOPS* (I/O operations per second). Once the head is correctly positioned and the platter lined up, however, a modern drive can stream data at a rate in excess of 100 MBps. Consequently, a lot of work has gone into optimizing filesystems and the entire I/O stack to minimize head movement and to place data sequentially on disk.

Flash-based solid state drives are now commonplace in the market. While they look and act like hard drives from a programming perspective they have very different performance characteristics. Because there are no heads to move and no platters to rotate it is possible to achieve a very high degree of parallelism inside the drive. Therefore, an SSD drive has the potential to deliver several orders of magnitude more IOPS than a disk drive.

Internally, flash drives often use page sizes bigger than 512 bytes to organize data, typically 4KB. That makes them similar to harddrives with 512-byte logical, 4KB physical block size. SSD drives can use the same ATA protocol parameters as regular drives to communicate their alignment and block sizes to the kernel.

7 Conclusion

Until now the Linux kernel has only been aware of one characteristic of the underlying storage, namely the sector size. Other parameters, such as stripe size and width, have been relegated to special case code in filesystem utilities.

Starting with 2.6.31, the Linux kernel is now aware of more of the hardware capabilities such as physical block size and alignment as well as the extra performance hints exported by some devices. All these characteristics are exported in a common fashion regardless of whether the device is a piece of hardware or a virtual disk exported by MD or DM. The common interface makes it easy for partitioning and filesystem tools, as well as `mdadm` and `dmsetup` to ensure that filesystems and data

are layed out in a way that ensures optimal performance
and correctness.