Linux & Advanced Storage Interfaces

Martin K. Petersen, Oracle Linux Engineering <martin.petersen@oracle.com>

Introduction

The industry migration towards harddrives with 4KiB physical blocks has caused us to make some changes to the Linux block I/O stack. These changes leverage existing parameters in SCSI Block Commands and the ATA Command Set to provide reporting of alignment and preferred I/O request sizes. The values provided are used by partition tools, LVM and MD as well as filesystems to ensure that data is layed out in an optimal fashion.

Furthermore, advanced technologies such as SSD, DIF & DIX, and thin provisioning have required changes to the way Linux queries and drives storage devices.

This document provides a set of recommendations for storage device vendors about which values to report to make Linux perform optimally.

1 Block Sizes & I/O Topology (Linux 2.6.31)

1.1 Logical vs. Physical Block Size

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 this document (and in the Linux kernel) we distinguish between:

- *physical block size* is the size used internally by the device firmware
- *logical block size* is the unit used to address a location on the storage

1.2 ATA Command Set

Linux will generally assume that an ATA device has both a logical and physical block size of 512 bytes. If that is not the case the following values must be reported by the device.

1.2.1 512-byte Logical Block Size, 4096-byte Physical Block Size

Word 106 bit 15 must be 0 and bit 14 must be 1. This signals that word 106 is valid. Bit 13 must be set to indicate that bits 3:0 are valid and that word 209 is specified.

Bits 3:0 indicate the physical block size exponent. I.e. 2^n logical blocks/physical block. For a drive with 4096-byte physical blocks and 512-byte logical blocks, a value of 3 is reported in bits 3:0 ($2^3 = 8$).

Command	Word	Bits	Hex
IDENTIFY DEVICE	106	0110 0000 0000 0011	0x6003
IDENTIFY DEVICE	209	0100 0000 0000 0000	0x4000

Table 1 512-byte logical, 4096-byte physical, 0-aligned

If the first logical block does not begin on a boundary aligned to the physical block size, word 209 should be reported by the device. Bit 15 must be zero, bit 14 must be 1. Bits 13:0 indicate the logical sector offset within the first physical sector where the first logical sector is placed.

For compatibility with legacy operating systems some vendors may align the blocks so that LBA 63 falls on a physical block boundary. In that case an alignment of 1 should be reported.

Command	Word	Bits	Hex
IDENTIFY DEVICE	106	0110 0000 0000 0011	0x6003
IDENTIFY DEVICE	209	0100 0000 0000 0001	0x4001

Table 2 512-byte logical, 4096-byte physical, 1-aligned

1.2.2 4096-byte Logical Block Size, 4096-byte Physical Block Size

If a drive uses 4KiB logical and physical blocks, IDENTIFY DEVICE words 106, 117, and 118 must be specified.

Word 106 bit 15 must be 0 and bit 14 must be 1. This signals that word 106 is valid. Bit 12 must be set to indicate that words 117 and 118 are specified.

Word 117 (LSB) and 118 (MSB) specify the logical block size in words. I.e. for a 4KiB drive that would be 2048.

Command	Word	Bits	Hex
IDENTIFY DEVICE	106	0101 0000 0000 0000	0x5000
IDENTIFY DEVICE	117	0000 1000 0000 0000	0x0800
IDENTIFY DEVICE	118	0000 0000 0000 0000	0x0000

Table 3 4096-byte logical, 4096-byte physical

1.2.3 Medium Rotation Rate

If the storage device is non-rotational (i.e. SSD or array) word 217 of IDENTIFY DEVICE should be set to one. The I/O scheduler may then be less agressive in terms of seek avoidance. Any other value reported in 217 will be ignored.

Command	Word	Bits	Hex
IDENTIFY DEVICE	217	0000 0000 0000 0001	0x0001

Table 4 Medium Rotation Rate

1.3 SCSI Block Commands

Linux will generally assume that a SCSI block device has matching logical and physical block sizes. If that is not the case, the device must report the following in READ CAPACITY(16):

Bytes 8 (MSB) – 11 (LSB) indicate the device's logical block size.

Byte 13 indicates the physical block size exponent. I.e. 2^n logical blocks/physical block. For a drive with 4096-byte physical blocks and 512-byte logical blocks, a value of 3 is reported in bits 3:0 ($2^3 = 8$).

Byte 14, bits 5:0 (MSB) and byte 15 specify the lowest aligned LBA.

Also note that Linux will only issue READ CAPACITY(16) to devices that report a VERSION of at least 0x5 (SCSI Primary Commands 3) in standard INQUIRY response. Consequently, any device whose physical block size is different from its logical ditto must report compliance to at least SPC version 3 / SBC version 2.

1.3.1 Identical Logical and Physical Block Sizes

Byte	Value (Hex)	Byte	Value (Hex)
8	0x00	8	0x00
9	0x00	9	0x00
10	0x02	10	0x10
11	0x00	11	0x00
13	0x00	13	0x00
14	0x00	14	0x00
15	0x00	15	0x00
512-byte blocks		4096-byte blocks	

Table 5 Identical logical and physical block sizes

1.3.2 512-byte Logical Block Size, 4096-byte Physical Block Size

When the storage device has a discrepancy between logical and physical block size, the logical blocks per physical block exponent must be specified in READ CAPACITY(16), byte 13.

For compatibility with legacy operating systems some vendors may align the blocks so that LBA 63 falls on a physical block boundary. In that case an alignment of 7 should be reported in byte 15 since 7 is the lowest physically aligned LBA on the device.

Byte	Value (Hex)	Byte	Value (Hex)
8	0x00	8	0x00
9	0x00	9	0x00
10	0x02	10	0x02
11	0x00	11	0x00
13	0x03	13	0x03
14	0x00	14	0x00
15	0x00	15	0x07
0-aligned		1-aligned	

Table 6 512-byte logical, 4096-byte physical

1.3.3 Medium Rotation Rate

If the storage device is non-rotational (i.e. SSD or array) the MEDIUM ROTATION RATE of the BLOCK DEVICE CHARACTERISTICS (0xB1) VPD page should be set to one. The I/O scheduler may then be less agressive in terms of seek avoidance. Any other value reported will be ignored.

Byte	Value (Hex)
4 (MSB)	0x00
5 (LSB)	0x01

Table 7 Block Device Characteristics

1.3.4 Block Limits

SCSI Block Commands version 2 introduced the BLOCK LIMITS (0xB0) VPD as a generic interface for array controllers to report preferred I/O sizes.

Bytes 6 (MSB) and 7 (LSB) indicate the OPTIMAL TRANSFER LENGTH GRANULARITY. Transfers with lengths not equal to a multiple of this many logical blocks may incur significant delays in processing.

Bytes 12 (MSB) – 15 (LSB) indicates the OPTIMAL TRANSFER LENGTH in blocks. Transfers with lengths exceeding this value may incur significant delays in processing.

Byte	Value (Hex)	
6 (MSB)	0x00	
7 (LSB)	0x80	128 × 512-byte blocks = 64 KiB
12 (MSB)	0x00	
13	0x00	
14	0x02	512 × 512-byte blocks = 256 KiB
15 (LSB)	0x00	

Table 8 Example Block Limits response page: 512-byte logical blocks, 64 KiB chunk size, 4-way striping

1.4 Block & Request Sizes

Internally the Linux kernel operates with several parameters for block devices:

- `logical_block_size` - used to address a location on the device
- `physical_block_size` - smallest unit the device can operate on
- `minimum_io_size` - device's preferred minimum unit for random I/O
- `optimal_io_size` - device's preferred unit for streaming I/O
- `alignment_offset` - the number of bytes the beginning of the Linux block device (partition/MD/LVM device) is offset from the underlying physical alignment

The logical and physical block sizes are set based on the values reported by the physical storage. Generally, `logical_block_size` ≦ `physical_block_size` ≦ `minimum_io_size`. If no value is specified for the physical block size, the logical one applies. And if the storage device does not set OPTIMAL TRANSFER LENGTH GRANULARITY in the BLOCK LIMITS VPD, then the physical block size applies.

Linux will attempt to align data structures on a `minimum_io_size` boundary, compensating for any device alignment offset reported. Linux will also attempt to issue I/O in multiples of `minimum_io_size`.

The `optimal_io_size` parameter is set based upon the value OPTIMAL TRANSFER LENGTH found in the BLOCK LIMITS VPD. If the BLOCK LIMITS VPD is not supported, or a value of 0 is reported, applications will make no assumptions about the `optimal_io_size`. While not enforced, it is expected that the OPTIMAL TRANSFER LENGTH is a multiple of OPTIMAL TRANSFER LENGTH GRANULARITY.

The `optimal_io_size` value is intended to be used by filesystems and applications to optimize for sustained write throughput. Filesystems may attempt to align data to an

`optimal_io_size` boundary, compensating for any device alignment reported. Filesystems may also attempt to issue I/O in units of `optimal_io_size`.

When stacking devices, Linux will scale `minimum_block_size` to a value suitable for all underlying devices. Linux will also attempt to align partitions, MD and LVM devices so they start on a physically aligned boundary. The alignment handling code is capable of handling mismatched devices. For instance a 512-byte logical/physical drive combined in a mirror with a 512-byte logical/4096-byte physical drive with 1-alignment will be adjusted to start on LBA 7.

It has been common practice for some RAID devices to compensate for the DOS LBA 63 misalignment internally. Typically this is done implicitly when selecting a Windows or a Linux personality on the LUN in question. Going forward, it is imperative that any alignment compensation is reported in READ CAPACITY(16).

1.5 Examples

The following table indicates the parameters resulting from the devices in the left column being registered with the Linux kernel. Note that the `min_io` value is the one that filesystems will strive to use both for block size and alignment (taking `align_off` into account).

The RAID array examples all provide a BLOCK LIMITS VPD to augment the information provided in READ CAPACITY(16).

Device	logical	physical	min_io	opt_io	align_off
Disk 512/512	512	512	512	0	0
Disk 512/4KiB	512	4096	4096	0	0
Disk 512/4KiB, 1-aligned	512	4096	4096	0	3585
Disk 4 KiB/4 KiB	4096	4096	4096	0	0
RAID0, 64 KiB × 4 drives	512	512	65536	262144	0
RAID1, 16 KiB	512	512	16384	0	0
RAID5, 8 KiB × 3 drives, 1-aln.	512	512	8192	16384	3584

Table 9 Common device types and their resulting parameters

2 Trim & Thin Provisioning (Linux 2.6.33)

At the block device level Linux implements support for a generic discard operation that allows filesystems or databases to indicate to the underlying storage that a given block range is no longer in use. For solid state drives this is being used to prevent wear leveling on unused blocks. For SCSI arrays the same operation is being used to implement space reclamation for devices that support thin provisioning.

Filesystems generally do no know whether they are sitting on top of an SSD or a SCSI array. Due to block device stacking, software RAID, etc. the filesystem could even span both types of devices. So the discard functionality has been defined in an abstract way with semantics that do not imply neither the SSD, nor the thin provisioning behavior.

Partitioning tools and filesystem mkfs utilities will generally issue a discard when they start using a block device. Filesystems may issue discards when a file is deleted or when they free up metadata.

2.1 ATA Command Set

In ATA the discard functionality is implemented via the DATA SET MANAGEMENT command with the TRIM bit set. For discard to be enabled the device must report compliance with at least ATA/ATAPI-7 and have bit 0 of IDENTIFY DEVICE word 169 set.

If the device has bits 14 and 5 in IDENTIFY DEVICE word 69 set, then the device must consistently return zeroed data buffers for any logical block that has previously been discarded using a TRIM operation.

If IDENTIFY DEVICE word 105 is specified, then Linux will use this value to scale the number of 512-byte blocks containing LBA Ranges that will be sent out as part of a DATA SET MANAGEMENT command. If word 105 is 0, a single 512-byte logical block will be used. Linux will not currently send commands with a payload bigger than 4KB. The LBA ranges in each payload will be contiguous and not aligned to any particular boundary.

Command	Word	Bits	Hex
IDENTIFY DEVICE	69	0100 0000 0010 0000	0x4020
IDENTIFY DEVICE	105	0000 0000 0000 1000	0x0008
IDENTIFY DEVICE	169	0000 0000 0000 0001	0x0001

Table 10 Typical SSD configuration

Linux does not currently coalesce discard requests and TRIM commands may be issued in realtime. It is expected that the storage device will queue the TRIM operation for later execution without incurring any significant delay to normal command processing.

Note that while Linux does not coalesce *discontiguous* ranges into a single command, the kernel may use the entire payload to describe a single *contiguous* range of blocks to be discarded.

Despite supporting payloads up to 4KB, Linux will currently only issue TRIM command with a payload size of 512. This is due to several devices repoting support for bigger payloads despite not actually supporting them.

2.2 SCSI Block Commands

In the SCSI block protocol the discard functionality is implemented via the WRITE SAME(10), WRITE SAME(16) or UNMAP commands. For discard to be enabled the device must report a VERSION of at least 5 (SPC-3) in the standard INQUIRY response, the LBPME bit of READ CAPACITY(16) must be set, the LOGICAL BLOCK PROVISIONING VPD page must be supported, and at least one of the LBPU, LBPWS or LBPWS10 bits must be set.

If the LBPRZ bit of READ CAPACITY(16) or the LOGICAL BLOCK PROVISIONING mode page is set, then the device must consistently return zeroed data buffers for any logical block that has previously been discarded using a WRITE SAME or UNMAP operation.

If the the BLOCK LIMITS VPD page is supported and its page length is reported to be 0x3c, Linux will parse the UNMAP parameters provided.

If MAXIMUM UNMAP LBA COUNT and MAXIMUM UNMAP BLOCK DESCRIPTOR COUNT are bigger than 0 in the BLOCK LIMITS VPD page and LBPU is set in the LOGICAL BLOCK PROVISIONING VPD page, Linux will issue UNMAP commands. Linux currently only issues commands with a single UNMAP block descriptor.

Response	Byte	Value (Hex)	Fields
INQUIRY	2	0x05	VERSION = SPC-3
READ CAPACITY(16)	14	0xC0	LBPME = 1, LBPRZ = 1
BLOCK LIMITS VPD	20–23	0xFFFFFFFF	MAX UNMAP LBA COUNT = ∞
″	24–27	0x00000001	MAX UNMAP DESCRIPTOR COUNT = 1
″	28–31	0x00000020	OPT UNMAP GRANULARITY = 32
″	32–35	0x00000000	UGAVALID, UNMAP ALIGNMENT = 0
LBP VPD	5	0x80	LBPU = 1

Table 11 Thin provisioning example, UNMAP, 16 KiB granularity

Otherwise, if LBPWS is set in the LOGICAL BLOCK PROVISIONING VPD page, Linux will issue WRITE SAME(16) with the UNMAP bit set. Linux will limit the maximum WRITE SAME(16) LBA count to the value reported in MAXIMUM UNMAP LBA COUNT in the BLOCK LIMITS VPD page, if supported.

Otherwise, if LBPWS10 is set in the LOGICAL BLOCK PROVISIONING VPD page, Linux will issue WRITE SAME(10) with the UNMAP bit set. Linux will limit the maximum WRITE SAME(10) LBA count to the value reported in MAXIMUM UNMAP LBA COUNT in the BLOCK LIMITS VPD page, if supported. WRITE SAME(10) support was added in Linux 2.6.39.

If OPTIMAL UNMAP GRANULARITY is reported in the BLOCK LIMITS VPD page then Linux will expose this value to filesystems which may use it to influence their block allocation

Response	Byte	Value (Hex)	Fields
INQUIRY	2	0x05	VERSION = SPC-3
READ CAPACITY(16)	14	0xC0	LBPME = 1, LBPRZ = 1
LBP VPD	5	0x20	LBPWS = 1
BLOCK LIMITS VPD	36–43	0xFFFFFFFF	MAX WRITE SAME LEN = 2TB

Table 12 Thin provisioning example, WRITE SAME(16) with UNMAP bit

algorithms. Similarly, if UGAVALID is set then Linux will expose UNMAP GRANULARITY ALIGNMENT to filesystems. Note that these values are only hints and they do not affect command invocation in any way. I.e. Linux may issue discard requests smaller than the reported granularity, and it may issue requests that do not start on the reported alignment boundary.

Linux does not coalesce discard requests and they are issued in realtime. It is expected that the storage device will queue the WRITE SAME or UNMAP operation without incurring any significant delay to normal command processing.

3 Data Integrity (Linux 2.6.27)

The T10 SCSI Block Commands version 2 standard introduced the T10 Protection Information Model (formerly known as DIF or Data Integrity Field). These additions to the block protocol standardize 8 bytes of protection information that is attached to each logical block. The 8 bytes are split into three fields, or tags, that take different meanings depending on the protection type the target has been formatted with.

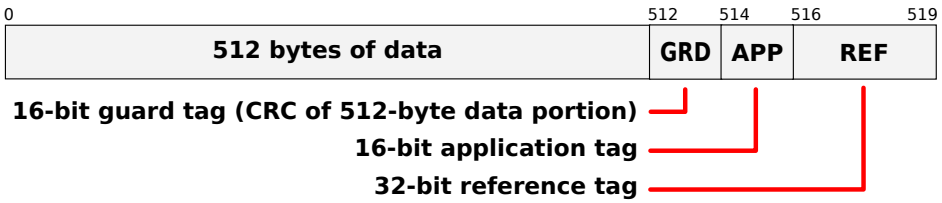

Figure 1 T10 Data Integrity Fields

3.1 Discovery

At initialization time device drivers can set a mask which indicates whether the HBA is capable of sending/receiving protection information. If the controller is capable and the target is formatted with DIF, Linux will issue read/write requests with protection information, i.e. RDPROTECT or WRPROTECT will be set to 1 on read and write requests respectively.

The target device must report a VERSION of at least 5 and have the PROTECT bit set in the standard INQUIRY response.

3.2 Protection Types

The PROT_EN bit must be set, and the P_TYPE field of READ CAPACITY(16) must be 0, 1 or 2, corresponding to protection types 1, 2, and 3 respectively.

Linux treats Type 1 and Type 2 protection identically. I.e. the reference tag is always treated as the lower 32 bits of the target LBA. For Type 2 the EXPECTED INITIAL REFERENCE TAG field in the 32-byte CDB is filled out accordingly.

Type 3 is supported but is currently not seeing any use.

Response	Byte	Value (Hex)	Fields
INQUIRY	2	0x05	VERSION = SPC-3
INQUIRY	5	0x01	PROTECT = 1
READ CAPACITY(16)	12	0x01	PROT_EN = 1, P_TYPE = 0
CONTROL MODE PAGE	5	0x80	ATO = 1

Table 13 T10 Protection Information example, Type 1

3.3 Application Tag

If the application tag space is available the storage device must set the ATO bit in the CONTROL MODE PAGE. If the application tag is available Linux expects it to be persistent storage. Any value written to the tag must be returned when the block subsequently read back. Note that Linux stores an individual application tag per logical block, not a constant value for the entire I/O request.

For Type 3 protection the reference tag behaves like the application tag. I.e. Linux expects to be able to store and retrieve 4 + 2 bytes of extra data per logical block.

Even if ATO is not set Linux may still submit a value in the application tag field for writes. The device must ignore the tag in this case. Linux does not expect to retrieve the same value on read.

4 SCSI ATA Translation

It is common for SATA devices to be deployed behind a SAS initiator. In that case it is up to the initiator's SCSI ATA Translation layer to express the ATA device's capabilities using the appropriate conduits in the SCSI protocol.

In addition to the translation required for basic device discovery, read/write I/O, and error handling, a direct block mapping SATL should correctly translate:

IDENTIFY word	SCSI byte	Description
	READ CAPACITY(16)	
69 [14, 5]	14 [6]	DRAT + RZAT ⇒LBPRZ
169 [0]	14 [7]	TRIM ⇒LBPME + LBPWS
106 [15:12, 3:0]	13 [3:0]	PHYS BLOCK EXPONENT
209 [15:0]	14 [5:0], 15[15:0]	PHYS BLOCK ALIGNMENT
	BLOCK DEVICE CHARACTERISTICS	
217	4, 5	MEDIA ROTATION RATE
	BLOCK LIMITS	
105	36 – 43	MAX WRITE SAME LENGTH

Table 14 ATA to SCSI translation

Note that ATA alignment reporting is different from SCSI READ CAPACITY(16). ATA indicates (negative) offset from the beginning of the physical block. T10 reports the lowest aligned LBA.

If a direct mapped ATA device implements TRIM, then the SATL should report Thin Provisioning support via either WRITE SAME or UNMAP as described in section section 2.2.

The BLOCK LIMITS VPD page MAX WRITE SAME LENGTH is calculated by multiplying the value of IDENTIFY DEVICE word 105 with 4194240 (65535 blocks per LBA range × 64 LBA ranges per 512-byte block).

Revision History

Date	Change
2011-07-02	(mkp) Correction: We require devices to report the SCSI revision level in the VERSION field, not the PRODUCT REVISION LEVEL field of the standard INQUIRY response. Update the Thin Provisioning section with the new LBP conventions and include support for WRITE SAME(10).
2010-08-08	(mkp) Clarify LBA range scaling and IDENTIFY DEVICE word 105. Update SCSI discard section with THIN PROVISIONING VPD page and TPU and TPWS bits.
2010-04-21	(mkp) Updated block limits VPD portion based on comments from Rob Elliott.
2009-12-07	(mkp) Minor edits.
2009-11-24	(mkp) Included and updated DIF/DIX and TP portions from other doc. Applies to Linux version 2.6.33+.
2009-08-17	(mkp) A few clarifications
2009-07-23	(mkp) Initial Version