An Exploration of New Hardware Features for Lustre

Nathan Rutman
Motivation

- Open-source
- Hardware-agnostic Linux
- Least-common-denominator hardware
Contents

- Hardware CRC
- MDRAID
- T10 DIF
- End-to-end data integrity
- Flash drive use
- Hybrid volumes
- HA and faster failover
Hardware CRC

• Nehalem CRC-32C
  – BZ23549
  – LU-241

Shuichi Ihara’s graphs from BZ23549

• Intel Westmere, AMD Bulldozer (2011)
  – PCLMULQDQ (64bit carryless multiply)
  – speed up CRC32, Adler
MDRAID

• Ongoing improvements in Linux SW RAID
  – Hardening
  – Zero copy writes
  – Performance
  – RAID 6, 6E, 10, etc

• Still to do
  – Zero copy reads
  – PDRAID
  – Hardware parity math acceleration
T10DIF and End-to-End Data Integrity
The guard tag protects the data portion of the sector. The application tag is simply opaque storage. And finally, the reference tag is being used to protect against out-of-order and misdirected write scenarios.

<table>
<thead>
<tr>
<th>512 bytes of data</th>
<th>GRD</th>
<th>APP</th>
<th>REF</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit guard tag (CRC of 512-byte data portion)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16-bit application tag</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32-bit reference tag</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: DIF tuple contents

Standardizing the contents of the protection information enables all nodes in the I/O path, including the disk itself, to verify the integrity of the data block.

Comparison of I/O Paths

A typical I/O submission scenario in an enterprise configuration is illustrated in figure 2. The only entity capable of using the 8 bytes of protection information is the array firmware.

A similar DIF-enabled configuration will look like figure 3. The I/O controller generates and appends the protection information and every subsequent node in the I/O path can verify the data integrity.

Combining T10 DIF with the Data Integrity Extensions allows the protection information to be attached even higher up in the stack–either in the application or in the operating system. The entire I/O path is protected and true end-to-end data integrity protection is achieved.

Figure 2: Normal I/O write: Application writes byte stream to OS. Filesystem writes in logical blocks that are multiples of 512-byte sectors. Depending on physical transport a CRC may be applied on the wire. Array firmware generates 8 bytes of proprietary protection information. Disk stores 520-byte sectors and generates its own CRC.

Figure 3: DIF I/O write: Application writes byte stream to OS. Filesystem writes in logical blocks that are multiples of 512-byte sectors. HBA generates protection information and sends out 520-byte sectors. SAN switch can optionally check protection information. Array firmware verifies protection information, optionally remaps reference tags and writes to disk. Disk verifies protection information before storing request.

Figure 4: DIX I/O write: Application writes byte stream to OS, optionally including protection information. Filesystem writes in logical blocks that are multiples of 512-byte sectors. If no protection information has been generated, OS automatically does so and attaches it to the I/O. HBA verifies data integrity, merges data and protection scatterlists and sends out 520-byte sectors. SAN switch can optionally check protection information. Array firmware verifies protection information, optionally remaps reference tags and writes to disk. Disk verifies protection information before storing request.

Figure 5 illustrates the protection envelopes of the protection schemes mentioned above. The Normal I/O line illustrates the disjoint integrity coverage offered using a current operating system and standard hardware. The HARD line shows the protection envelope offered by the Oracle Database accessing a disk array with HARD capability. DIF shows coverage...
Figure 3

DIF I/O write: Application writes byte stream to OS. Filesystem writes in logical blocks that are multiples of 512-byte sectors. HBA generates protection information and sends out 520-byte sectors. SAN switch can optionally check protection information. Array firmware verifies protection information, optionally remaps reference tags and writes to disk. Disk verifies protection information before storing request.

Figure 4

DIX I/O write: Application writes byte stream to OS, optionally including protection information. Filesystem writes in logical blocks that are multiples of 512-byte sectors. If no protection information has been generated, OS automatically does so and attaches it to the I/O. HBA verifies data integrity, merges data and protection scatterlists and sends out 520-byte sectors. SAN switch can optionally check protection information. Array firmware verifies protection information, optionally remaps reference tags and writes to disk. Disk verifies protection information before storing request.

Figure 5 illustrates the protection envelopes of the protection schemes mentioned above. The Normal I/O line illustrates the disjoint integrity coverage offered using a current operating system and standard hardware. The HARD line shows the protection envelope offered by the Oracle Database accessing a disk array with HARD capability. DIF shows coverage...
Write
- Userspace computes GRD per sector (optional)
- Client recomputes GRD after kernel copy
- Client adds GRD to BIO bulk descriptor
- OST pulls BIO
- OST passes PI to ldiskfs
- MDRAID maps REF tags
- HBA takes SG lists of data and PI
- HBA recalculates GRD (retries from client)
- Disk verifies REF and GRD (retries from HBA)

Read
- Client sets up PI and data buffers, sends BIO
- OST sets up PI and data buffers
- MDRAID requests data and parity blocks
- Disk verifies REF and GRD
- HBA maps data and PI to buffers
- MDRAID verifies parity, reconstructs corrupted data
- OST sends data
- Client verifies GRD
- Userspace reverifies GRD after kernel copy (optional)
Flash Drives
Flash Drives

• SSDs are fast, but not large

• Caching

• Lustre persistent data files
  – last_rcvd, last_objid, open_write

• Metadata
  – journals
  – WIBs
  – Lustre FS metadata

• Data
  – Small files
EXT4 Hybrid Volumes
EXT4 Hybrid Volumes

• Problems today
  – Metadata distributed around disk, breaking up large disk chunks and slowing fsck
  – Small files treated the same as large files (e.g. same RAID level)

• Hybrid Volume
  – A single filesystem spanning a group of local devices with different RAID striping, speeds, hardware, or access patterns
  – Single device loss will kill the FS, so each device must be safe (RAID)

• Change allocator
  – Put metadata together, all on fast RAID1
  – Put small files after this on RAID1
  – Put large files aligned on 1MB boundaries on RAID6
  – EXT4 online defragmenter can migrate them
EXT4 Hybrid Volume Layout

- **flex_bg**
  - Group descriptors
  - Inode Bitmaps
  - Inode Tables
  - Directories
  - EAs

- bg
  - Directories
  - EAs
  - Small files

- bg
  - Large files

BG0 | BG1 | BG2 | BG3 | …
---|---|---|---|---
**MD** | small files | large file stripes

**raid**
- RAID 0
- RAID 1
- RAID 6

Disk 0 | Disk 1 | Disk 2 | Disk 3 | Disk 4
EXT4 Hybrid Volume Advantages

• Larger transfer sizes and reduced seek time for each type
  – Don’t need to skip over data to get to metadata, and vice-versa
• Eliminate seek time between types
  – Leave the data volume read head waiting at the next 1MB boundary
  – Leave the MD head waiting at the allocator bitmaps
• Avoid RAID6 parity recompute penalty for small files
• Smaller metadata volume makes SSD, RAID1 practical
• FSCK times reduced
  – location implies content type (don’t need to read the whole disk for each pass)
  – ordered metadata is faster to read
• Multiple data volumes
  – Can sleep volumes as needed
HA and Failover

• HW Monitoring and Imperative Recovery
  – don’t wait for Lustre timeouts
  – more important in larger clusters

• Persistent RAM
  – RAMdisk MDT
  – Quick flush to SSD on power loss
Thank You