SMR Impact on Linux Storage Subsystem

Jorge Campello, Adam Manzanares

HGST, a Western Digital Company.
Magnetic Recording System Technologies

New recording system technologies are needed to keep the HDD industry on its historical track of delivering capacity improvements over time.

- **Longitudinal Magnetic Recording (LMR)**
  - 1 Tb/in²
  - 10 Tb/in²

- **Perpendicular Magnetic Recording (PMR)**
  - 150 Gb/in²
  - 5 Tb/in²

- **Discrete Track Recording (DTR)**
  - (Limited Gains)

- **Shingled Magnetic Recording (SMR)**

- **Heat Assisted Magnetic Recording (HAMR)**

- **Microwave Assisted Magnetic Recording (MAMR)**

- **Bit Patterned Magnetic Recording (BPMR)**

Future recording technologies will build on SMR, not replace it.

---

Y. Shiroishi, Intermag 2009, FA-01
What is Shingled Magnetic Recording (SMR)?

SMR write head geometry extends well beyond the track pitch in order to generate the field necessary for recording. Tracks are written sequentially in an overlapping manner forming a pattern similar to shingles on a roof.

SMR Constraint: Rewriting a given track will damage one or more subsequent tracks.

Wood, Williams, et al., IEEE TRANSACTIONS ON MAGNETICS, VOL. 45, NO. 2, FEBRUARY 2009
## SMR Types

<table>
<thead>
<tr>
<th>SMR category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drive managed (Autonomous)</td>
<td>No host changes. SMR device manages all requests. Performance is unpredictable in some workloads. Backward compatible</td>
</tr>
</tbody>
</table>

| Host aware | Host uses new commands & information to optimize write behavior. If host sends sub-optimal requests the SMR device accepts the request but performance may become unpredictable. Backward compatible |

| Host Managed | Host uses new commands & information to optimize write behavior. Performance is predictable. If host sends sub-optimal requests the SMR device rejects the request. Not backward compatible |

ZBC = Zoned Block Commands

ZAC = Zoned ATA Commands
Zoned Block Device

- The device is divided into Zones
- Two type of Zones
  - Conventional Zones
  - Write Pointer Zones
- Conventional Zones
  - Behave according to the direct access block device type model in SBC-3.
- Write Pointer Zones
  - Behave according to the new Zoned Block Device Model, and varies depending on device type
    - There is write pointer (WP), writes should be at WP.
- Two device types
  - Host Managed Zoned Block Device
    - Writes not at WP, or spanning Zones are not allowed.
    - Reads not allowed to span Zones, or cross WP.
  - Host Aware Zoned Block Device
    - Non sequential writes allowed, spanning zones allowed for R/W.

Note: These are high-level simplifications. For a more accurate description see the relevant T10/T13 documents.
# ZBC/ZAC Device Types – current drafts

<table>
<thead>
<tr>
<th></th>
<th>Direct Access</th>
<th>Host Aware</th>
<th>Host Managed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peripheral Device Type</td>
<td>00h</td>
<td>00h</td>
<td>14h</td>
</tr>
<tr>
<td>HAW_ZBC</td>
<td>0b</td>
<td>1b</td>
<td>0b</td>
</tr>
<tr>
<td>Conventional zones</td>
<td>n/a</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>Seq’l wr preferred zones</td>
<td>n/a</td>
<td><strong>Mandatory</strong></td>
<td>Disallowed</td>
</tr>
<tr>
<td>Seq’l wr only zones</td>
<td>n/a</td>
<td>Disallowed</td>
<td><strong>Mandatory</strong></td>
</tr>
<tr>
<td>Reads and writes crossing seq’l write only zone boundaries</td>
<td>n/a</td>
<td>n/a</td>
<td>Disallowed</td>
</tr>
<tr>
<td>REPORT ZONES</td>
<td>Disallowed</td>
<td>Mandatory</td>
<td>Mandatory</td>
</tr>
<tr>
<td>RESET WRITE POINTER</td>
<td>Disallowed</td>
<td>Mandatory</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
SMR Introduction Models

User Space

Kernel

Hardware

ZBC/ZAC
Host Aware
Host Managed

ZBC/ZAC
Host Aware
Host Managed
libzbc and lkvs:
Linux ZBC library and Linear Key Value Store Application
libzbc

- Allows Linux applications access to ZBC host-managed disks
  - Access to disk zone information and read/write operations in zones through direct SCSI command execution (SG_IO)
  - ZAC drives will be supported by libzbc as well

- Additionally, provide a ZBC emulation layer for operation on top of standard SAS/SATA block devices
  - Zone configuration of the disk is emulated within the library
# libzbc Interface

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>Input</th>
<th>Output</th>
<th>SCSI command (native mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>zbc_open</td>
<td>Open a device</td>
<td>Device file path</td>
<td>Device handle</td>
<td>INQUIRY, READ CAPACITY 16</td>
</tr>
<tr>
<td>zbc_close</td>
<td>Close an open device</td>
<td>Device handle</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>zbc_get_device_info</td>
<td>Get a device information (size, sector size,...)</td>
<td>Device handle</td>
<td>Device information</td>
<td>None</td>
</tr>
<tr>
<td>zbc_report_zones</td>
<td>Get information on zones following a specified LBA</td>
<td>Device handle, zone start LBA, zone filter</td>
<td>Zone information</td>
<td>REPORT ZONES</td>
</tr>
<tr>
<td>zbc_reset_write_pointer</td>
<td>Reset the write pointer of an open or full zone</td>
<td>Device handle, zone start LBA</td>
<td>None</td>
<td>RESET WRITE POINTER</td>
</tr>
<tr>
<td>zbc_pread</td>
<td>Read data from a zone</td>
<td>Device handle, Zone to read, LBA offset in the zone, number of sectors to read, data buffer</td>
<td>Amount of sectors read and data</td>
<td>READ 16</td>
</tr>
<tr>
<td>zbc_pwrite</td>
<td>Write data to a zone</td>
<td>Device handle, Zone to write, LBA offset in the zone, number of sectors to write, data buffer</td>
<td>Amount of sectors written</td>
<td>WRITE 16</td>
</tr>
</tbody>
</table>
libzbc Interface (Emulation Mode)

- These functions are used to initialize an emulated ZBC device
  - Write pointer persistency is also emulated
    » Zone configuration and current write pointer values are saved to the disk on execution of the zbc_close function

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>Input</th>
<th>Output</th>
<th>SCSI command (native mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>zbc_set_zones</td>
<td>Configure the zones of an emulated device</td>
<td>Device handle, size of conventional zone, size of sequential write zones</td>
<td>None</td>
<td>None*</td>
</tr>
<tr>
<td>zbc_set_write_pointer</td>
<td>Change a zone write pointer LBA value</td>
<td>Device handle, zone start LBA, write pointer value</td>
<td>None</td>
<td>None*</td>
</tr>
</tbody>
</table>
Linear Key Value Store (lkvs) Application
Linear Key Value Store Architecture

- **lkvs**
  - Implements a simple append only KVS as an example use of libzbc
  - Queries drive info (write pointer, zone information) through libzbc
  - Read/write executed through libzbc

- **libzbc**
  - Provides zone information, write pointers, to lkvs

Applications link with libzbc

lkvs gets ZBC device information and read/write operations are performed through libzbc

- lkvs
- libzbc
- device file
- ZBC Device
## lkvs Interface

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>openDev</strong></td>
<td>Open a device</td>
<td>Device file path, format flags</td>
<td>Bool success</td>
</tr>
<tr>
<td><strong>Put</strong></td>
<td>Insert key/value pair into the store</td>
<td>Key string, value buffer, size</td>
<td>Bool success</td>
</tr>
<tr>
<td><strong>Get</strong></td>
<td>Get key/value pair form the store</td>
<td>Key string, value buffer, size</td>
<td>Bool success</td>
</tr>
<tr>
<td><strong>List</strong></td>
<td>List key/value pairs on the device (Not Finalized)</td>
<td>TBD</td>
<td>TBD</td>
</tr>
</tbody>
</table>