Western Digital®

zonefs: Mapping the POSIX File System Interface to Zoned Block Device Accesses

Damien Le Moal, Western Digital Research
Ting Yao, Huazhong University

2020 Linux Storage and Filesystems Conference, VAULT'20

Damien presenting at LSF/MM

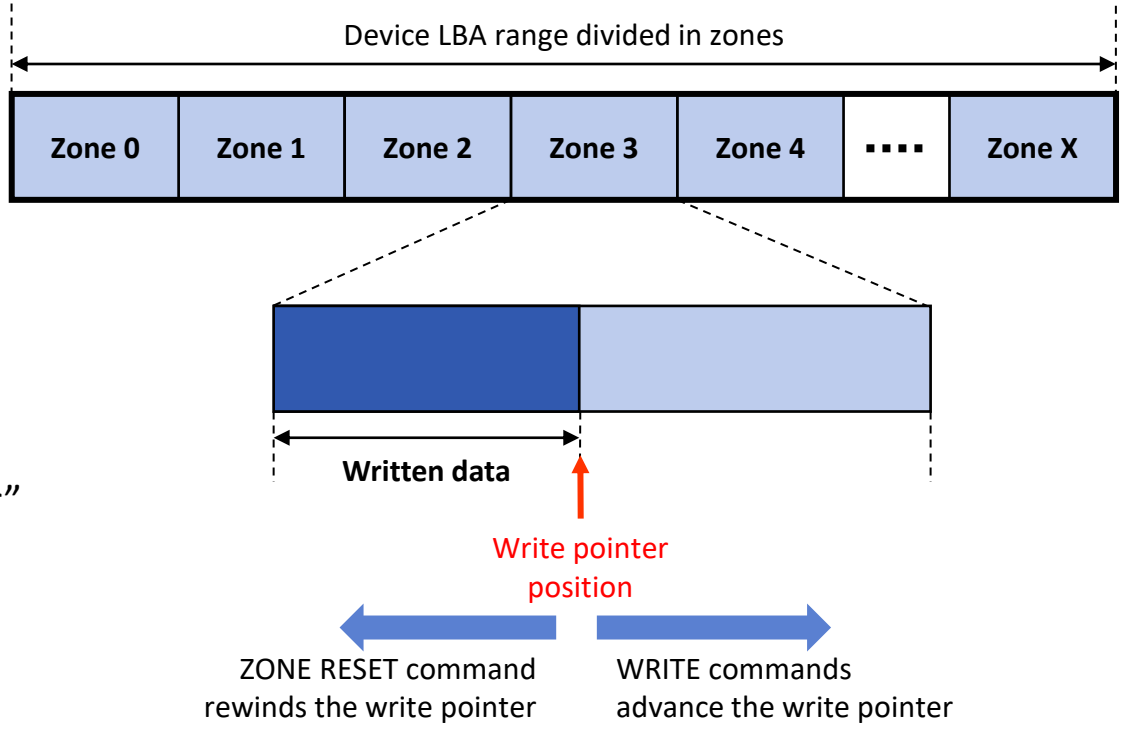
Outline

- Background
  - Zoned block devices principles
  - Linux support
- zonefs
  - Overview
  - File tree, format options and mount options
  - File operations mapping to zoned block device commands
  - I/O error handling
- Example use
  - LevelDB prototype implementation
- Future work and conclusion

Zoned Block Devices

Random reads but sequential writes

- Commonly found today in the form of SMR hard-disks (Shingled Magnetic Recording)
  - Interface defined by the ZBC (SCSI) and ZAC (ATA) standards
- LBA range divided into zones of different types
  - Optional conventional zones
    - Accept random writes
  - Sequential write required zones
    - Writes must be issued sequentially starting from the “write pointer”
    - Zones must be reset before rewriting
      - “rewind” write pointer to beginning of the zone
- NVMe Zoned Namespace defines a similar interface for NVMe SSDs
  - But no conventional zones



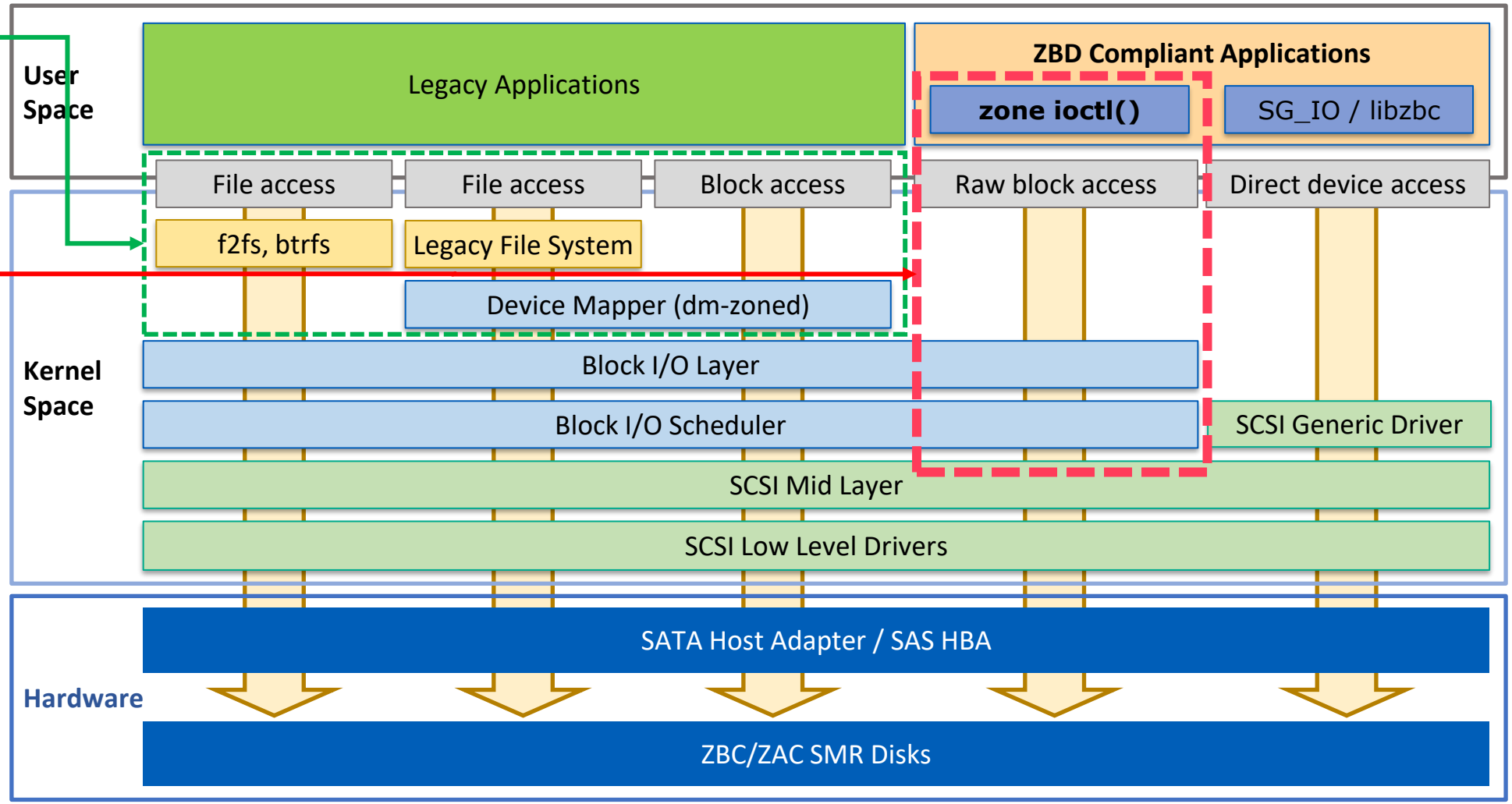
Linux Kernel Zoned Block Device Support

Since kernel version 4.10

Advanced support:
- **POSIX defined behavior**

Minimal support:
- Device sequential write constraint exposed to users
- Zone management API (ioctl) and write ordering guarantees

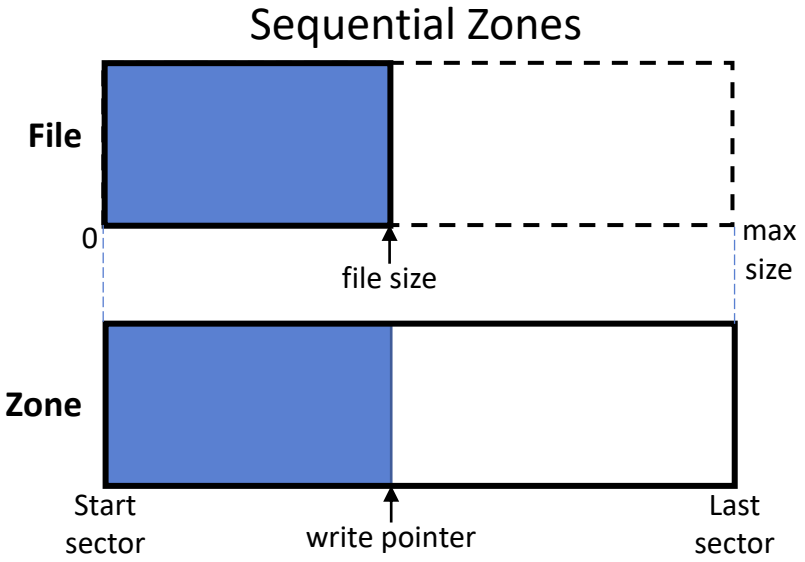
Zonefs goal
Simplify raw block device zone access and management

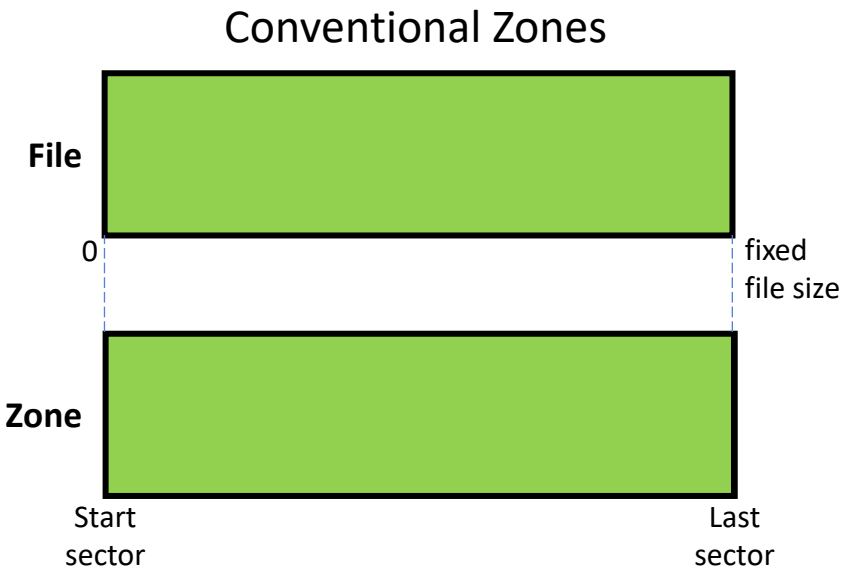


Zonefs: Overview

Expose each zone as a file

- Device zones are exposed as regular files
  - File size determined from its zone type and its zone write pointer position
- Zone information obtained from the device is used as inode metadata
  - On-disk metadata reduced to a static superblock (first zone)
    - No journaling needed
- File I/O block mapping implemented using iomap
  - No buffer-heads, static block mapping per file
- Immutable file names
  - Zone number per sub-group type
- File attributes control
  - Per zone UID, GID, access permissions



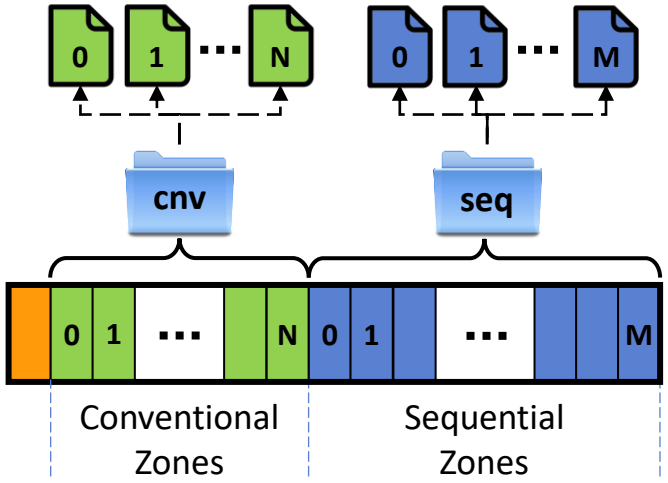


Zonefs: File Tree

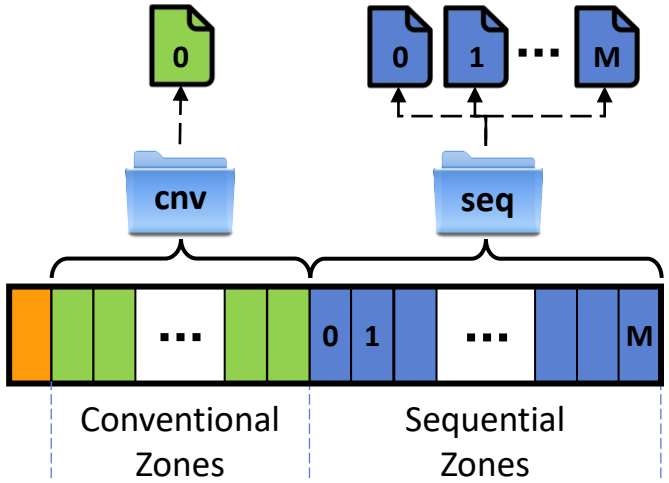
First zone used for the static superblock

- Files are grouped per zone type in different sub-directories
  - "cnv" for conventional zones
  - "seq" for sequential write required or preferred zones
- Contiguous conventional zones can be aggregated into a single file

No zone aggregation (default)



Aggregated conventional zones



Zonefs: Format and Mount Options

First zone used for the static superblock

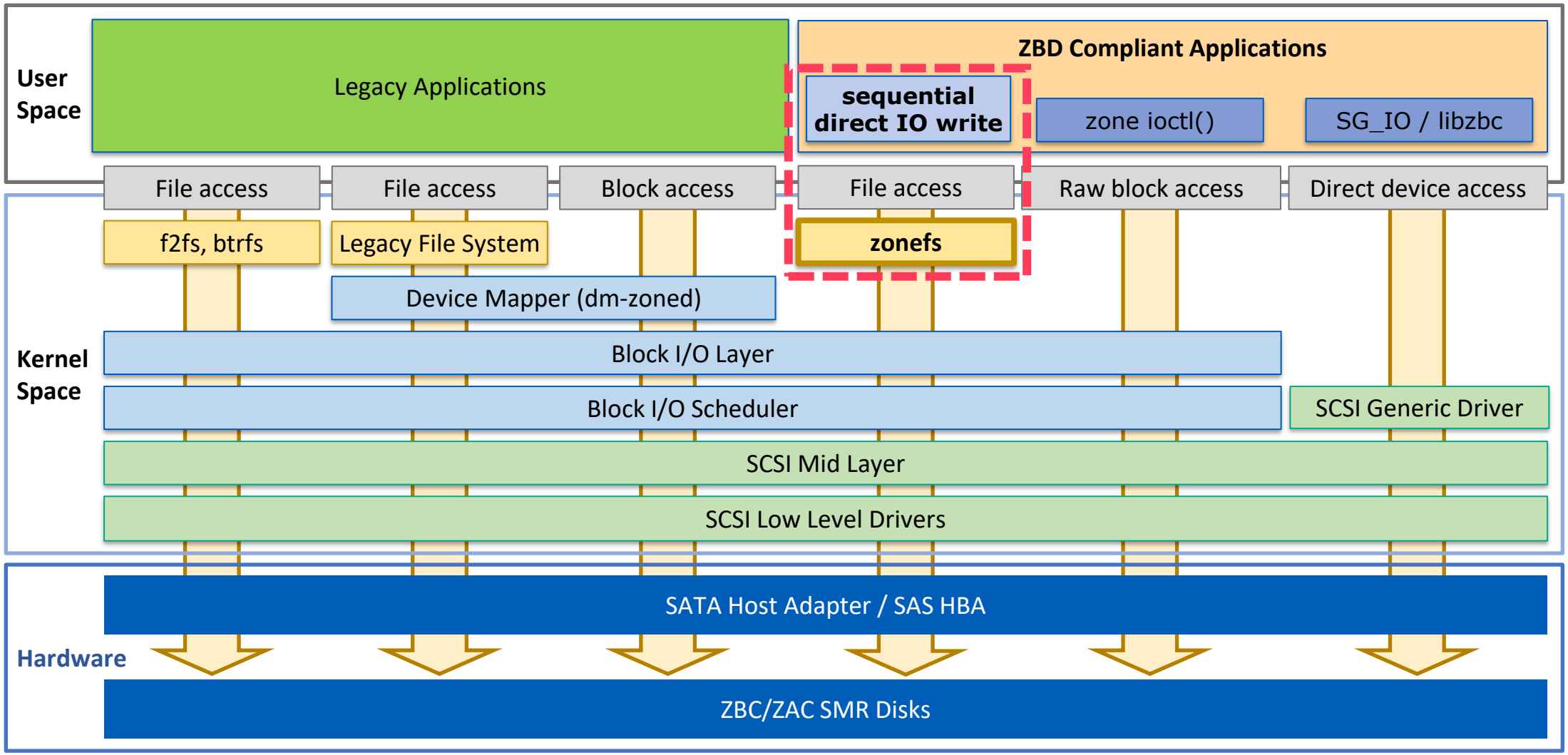
- Format options
  - File attributes: default UID, GID and access permissions
  - Conventional zones aggregation: on/off
- Mount options
  - Define behavior on IO error and zone condition changes
    - Handle unexpected change to a sequential zone write pointer
      - E.g. If a large write operation partially fails
    - Handle device transition of “bad” zones to OFFLINE or READONLY state
  - Defined behaviors:
    - remount-ro: File system remounted read-only
    - zone-ro: affected zone goes read-only
    - zone-offline: affected zone assumed to be offline
      - No accesses possible
    - repair: use zone write pointer to fix the file size and continue

zonefs error handling options

error=xxx mount option	Device zone condition	Post error recovery state				
		File size	Access permissions			
			File		Device zone	
			Read	Write	Read	Write
remount-ro (default)	Good Read-only Offline	Fixed Fixed 0	Yes Yes No	No No No	Yes Yes No	Yes No No
zone-ro	Good Read-only Offline	Fixed Fixed 0	Yes Yes No	No No No	Yes Yes No	Yes No No
zone-offline	Good Read-only Offline	0 0 0	No No No	No No No	Yes Yes No	Yes No No
repair	Good Read-only Offline	Fixed Fixed 0	Yes Yes No	Yes No No	Yes Yes No	Yes No No

Zonefs is *NOT* a Regular POSIX Filesystem

Requires ZBD compliant applications



Example: 15TB SMR Disk

524 conventional zones and 55356 sequential zones

- First conventional zone used for the super block

```
# mkzonefs -f /dev/sdi
/dev/sdi: 29297213440 512-byte sectors (13970 GiB)
  Host-managed device
  55880 zones of 524288 512-byte sectors (256 MiB)
  524 conventional zones, 55356 sequential zones
  0 read-only zones, 0 offline zones
Format:
  55879 usable zones
  Aggregate conventional zones: disabled
  File UID: 0
  File GID: 0
  File access permissions: 640
  FS UUID: 67730d07-34c3-472c-9fde-22d3c705f231
Resetting sequential zones
Writing super block
# mount –t zonefs /dev/sdi /mnt
# ls -l /mnt
total 0
dr-xr-xr-x 2 root root   523 Feb 17 10:40 cnv
dr-xr-xr-x 2 root root 55356 Feb 17 10:40 seq
```

Number of files

Conventional zone file size is fixed to the zone size

```
# ls -lv /mnt/cnv
total 137101312
-rw-r----- 1 root root 268435456 Feb 17 10:43 0
-rw-r----- 1 root root 268435456 Feb 17 10:43 1
-rw-r----- 1 root root 268435456 Feb 17 10:43 2
...
-rw-r----- 1 root root 268435456 Feb 17 10:43 521
-rw-r----- 1 root root 268435456 Feb 17 10:43 522
```

Sequential zone file size indicate the amount of written data

```
# ls -lv /mnt/seq
total 14511243264
-rw-r----- 1 root root        0 Feb 17 10:43 0
-rw-r----- 1 root root  1048576 Feb 17 10:45 1
-rw-r----- 1 root root        0 Feb 17 10:43 2
-rw-r----- 1 root root 268435456 Feb 17 10:45 3
-rw-r----- 1 root root        0 Feb 17 10:43 4
...
-rw-r----- 1 root root        0 Feb 17 10:43 55354
-rw-r----- 1 root root        0 Feb 17 10:43 55355
```

Example: 15TB SMR Disk

524 conventional zones and 55356 sequential zones

- With aggregated conventional zones

```
# mkzonefs -f -o aggr_cnv /dev/sdi
/dev/sdi: 29297213440 512-byte sectors (13970 GiB)
  Host-managed device
  55880 zones of 524288 512-byte sectors (256 MiB)
  524 conventional zones, 55356 sequential zones
  0 read-only zones, 0 offline zones
Format:
  55879 usable zones
  Aggregate conventional zones: enabled
  File UID: 0
  File GID: 0
  File access permissions: 640
  FS UUID: af10a4cd-8732-4400-bb2c-61889a12a35e
Resetting sequential zones
Writing super block
# mount –t zonefs /dev/sdi /mnt
# ls -l /mnt
total 0
dr-xr-xr-x 2 root root     1 Feb 17 10:51 cnv
dr-xr-xr-x 2 root root 55356 Feb 17 10:51 seq
```

All conventional zones aggregated into a single file

The file size is the total size of all aggreagted zones

```
# ls -lv /mnt/cnv/
total 137101312
-rw-r----- 1 root root 140391743488 Feb 17 10:51 0
```

Aggregated zone file can be used as a regular file, as a disk through loopback, etc

```
# mkfs.ext4 /mnt/cnv/0
.. .
# mount -o loop /mnt/cnv/0 /data
# ls -la /data
total 24
drwxr-xr-x   3 root root  4096 Feb 17 10:54 .
dr-xr-xr-x. 22 root root  4096 Feb 17 10:55 ..
drwx------   2 root root 16384 Feb 17 10:54 lost+found
```

File Operations: Discovery

How many zones and zones information

- Raw block device case
  - BLKNRZONES and BLKREPORTZONE ioctl()
  - struct blk_zone contains all information for a zone
    - Zone type, write pointer, start sector, size

- Zonefs case
  - stat()/fstat()
    - Zone group directory size indicates the number of zones
    - Zone write pointer: file size (stat.st_size)
    - Zone size: file blocks (stat.st_blocks << 9)
      - Maximum file size

```
/* How many zones ? */
fd = open("/dev/sdX", O_RDONLY);
ioctl(fd, BLKGETNRZONES, &nr_zones);

/* Zones information */
rep = malloc(sizeof(struct blk_zone_report)
        + sizeof(struct blk_zone) * nr_zones);
ioctl(fd, BLKREPORTZONE, &rep);

for (i = 0; i < nr_zones; i++) {
                wp = rep.zones[i].wp;
                ...
}
```

```
/* How many zones ? */
stat("/mnt/seq", &stat);
nr_zones = stat.st_size;

/* Zones information */
for (i = 0; i < nr_zones; i++) {
                sprint(filename, "/mnt/seq/%d", i);
                stat(filename, &stat);
                ...
}
```

File Operations: Sequential Writes

O_APPEND and zone isolation

- Raw block device case
  - pwrite()
  - Write offset allows reaching any zone
    - A bug can corrupt another zone

```
/* Write zone i */
fd = open("/dev/sdX", O_RDWR | O_DIRECT);

while (ofst < rep.zones[i].length << 9) {
                pwrite(fd, buf, size, ofst);
                ofst += size;
                ...
}
```

- Zonefs case
  - Regular write() with O_APPEND or pwrite()
  - Write operation limited to the open zone file
    - Cannot corrupt another zone

```
/* Write zone i */
sprint(filename, "/mnt/seq/%d", i);
fd = open(filename, O_RDWR | O_DIRECT | O_APPEND);

while (stat.st_blocks) {
                write(fd, buf, size);
                stat.st_blocks -= size >> 9;
                ...
}
```

File Operations: Zone Management

Zone reset and zone finish

- Raw block device case
  - BLKRESETZONE and BLKFINISHZONE ioctl()
- Zonefs case
  - truncate()/ftruncate() to 0 for zone reset
  - truncate()/ftruncate() to maximum file size for zone finish

```
fd = open("/dev/sdX", O_RDWR);

/* Reset zone i */
range.sector = rep.zones[i].start;
range.nr_sectors = rep.zones[i].length;
ioctl(fd, BLKRESETZONE, &range);

/* Finish zone i */
range.sector = rep.zones[i].start;
range.nr_sectors = rep.zones[i].length;
ioctl(fd, BLKFINISHZONE, &range);
```

```
sprint(filename, "/mnt/seq/%d", i);

/* Reset zone i */
truncate(filename, 0);

/* Finish zone i */
truncate(filename, stat.st_blocks << 9);
```

Use Case Example: LevelDB

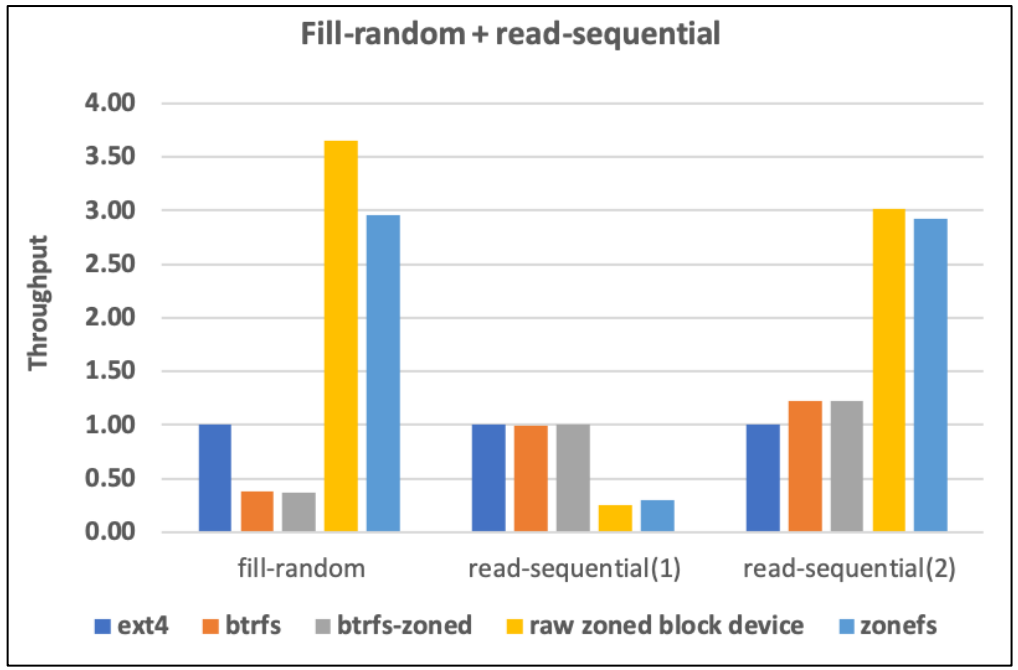
Use zone files to store SSTables

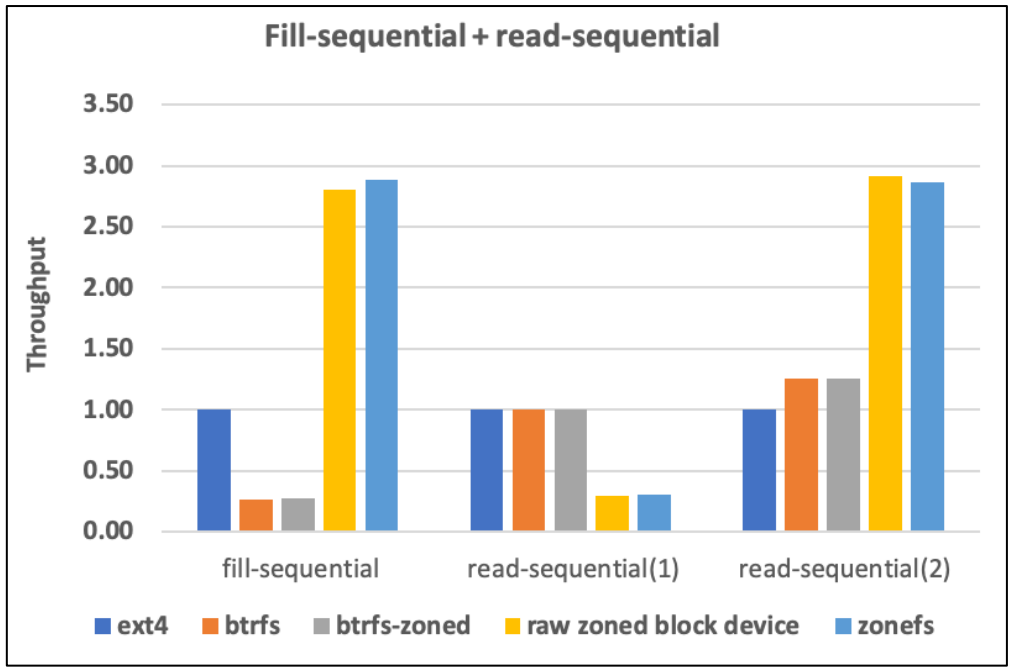
- Modified LevelDB implementation to use zone files for storing SSTable files
  - Use direct IO writes to zones
    - Similar modification to also add raw zoned block device support
  - Buffered and mmap reads of SSTables
- Experiment: Regular NVMe SSD vs prototype NVMe ZNS drive
  - Regular SSD: ext4 (baseline) and btrfs
  - Prototype ZNS drive: btrfs-zoned (on-going work), raw block device and zonefs
  - 16B keys and 4KB values
  - Execute db-bench with the sequences:
    - fillrandom, readseq, readseq
    - fillseq, readseq, readseq
- Results normalized to the Regular NVMe SSD + ext4 baseline case
  - All results are averaged over of 5 runs

Use Case Example: LevelDB

Random and sequential write operations followed by read operations

- 2.5 to 3 times better throughput for ingest (random and sequential)
  - File system journaling overhead avoided
- Direct IO write operations result in lower first-time read performance
  - No data in page cache after writes
  - But up to 3x throughput for second read with warm cache





Current Status

Initial release included with upstream kernel

- Initial pull accepted for Linux 5.6-rc1
  - Selection under “File systems” menu
  - Requires *CONFIG_BLK_DEV_ZONED* selection
    - Zoned block device support in “Enable the block layer” menu
- Userspace tool available on github
  - https://github.com/damien-lemoal/zonefs-tools
  - Provides the format utility *mkzonefs* (*mkfs.zonefs*)
- xfstests support not planned
  - Too few common test cases with regular POSIX file systems
  - A special test suite is provided with zonefs-tools

Future Work

Extend file operation mapping to zone operations

- Better handling of *IOCB_NOWAIT* for asunchronous I/Os
  - Currently silently ignored since it can cause IO reordering if enabled
- Continue integration of zone management commands
  - Zone explicit open/close with file (inode) open()/close()
    - Can improve performance for ZNS SSDs (control of active resources)
  - Integrate NVMe ZNS “zone append” command use
    - For asynchronous write operations specifying RWF_APPEND and/or files opened with O_APPEND
- Read-after-write performance improvements
  - Explore new “*RWF_CACHED*” flag: *O_SYNC* like behavior while retaining direct-IO alignement constraint
    - Warm up page cache on direct writes for page aligned writes
- Continue exploring different use cases to identify potential areas of improvement
  - RocksDB on-going
  - Clearly separate application problems vs zonefs performance limitations
    - For now, read-after-write problem is the most obvious

Questions ?

Western Digital®