ZoneFS - Zone filesystem for Zoned block devices

Introduction

zonefs is a very simple file system exposing each zone of a zoned block device as a file. Unlike a regular POSIX-compliant file system with native zoned block device support (e.g. f2fs), zonefs does not hide the sequential write constraint of zoned block devices to the user. Files representing sequential write zones of the device must be written sequentially starting from the end of the file (append only writes).

As such, zonefs is in essence closer to a raw block device access interface than to a full-featured POSIX file system. The goal of zonefs is to simplify the implementation of zoned block device support in applications by replacing raw block device file accesses with a richer file API, avoiding relying on direct block device file ioctls which may be more obscure to developers. One example of this approach is the implementation of LSM (log-structured merge) tree structures (such as used in RocksDB and LevelDB) on zoned block devices by allowing SSTables to be stored in a zone file similarly to a regular file system rather than as a range of sectors of the entire disk. The introduction of the higher level construct "one file is one zone" can help reducing the amount of changes needed in the application as well as introducing support for different application programming languages.

Zoned block devices

Zoned storage devices belong to a class of storage devices with an address space that is divided into zones. A zone is a group of consecutive LBAs and all zones are contiguous (there are no LBA gaps). Zones may have different types.

- Conventional zones: there are no access constraints to LBAs belonging to conventional zones. Any read or write access can be executed, similarly to a regular block device.
- Sequential zones: these zones accept random reads but must be written sequentially. Each sequential zone has a write pointer maintained by the device that keeps track of the mandatory start LBA position of the next write to the device. As a result of this write constraint, LBAs in a sequential zone cannot be overwritten. Sequential zones must first be erased using a special command (zone reset) before rewriting.

Zoned storage devices can be implemented using various recording and media technologies. The most common form of zoned storage today uses the SCSI Zoned Block Commands (ZBC) and Zoned ATA Commands (ZAC) interfaces on Shingled Magnetic Recording (SMR) HDDs.

Solid State Disks (SSD) storage devices can also implement a zoned interface to, for instance, reduce internal write amplification due to garbage collection. The NVMe Zoned NameSpace (ZNS) is a technical proposal of the NVMe standard committee aiming at adding a zoned storage interface to the NVMe protocol.

Zonefs Overview

Zonefs exposes the zones of a zoned block device as files. The files representing zones are grouped by zone type, which are themselves represented by sub-directories. This file structure is built entirely using zone information provided by the device and so does not require any complex on-disk metadata structure.

On-disk metadata

zonefs on-disk metadata is reduced to an immutable super block which persistently stores a magic number and optional feature flags and values. On mount, zonefs uses blkdev_report_zones() to obtain the device zone configuration and populates the mount point with a static file tree solely based on this information. File sizes come from the device zone type and write pointer position managed by the device itself.

The super block is always written on disk at sector 0. The first zone of the device storing the super block is never exposed as a zone file by zonefs. If the zone containing the super block is a sequential zone, the mkzonefs format tool always "finishes" the zone, that is, it transitions the zone to a full state to make it read-only, preventing any data write.

Zone type sub-directories

Files representing zones of the same type are grouped together under the same sub-directory automatically created on mount.

For conventional zones, the sub-directory "cnv" is used. This directory is however created if and only if the device has usable conventional zones. If the device only has a single conventional zone at sector 0, the zone will not be exposed as a file as it will be used to store the zonefs super block. For such devices, the "cnv" sub-directory will not be created.

For sequential write zones, the sub-directory "seq" is used.

These two directories are the only directories that exist in zonefs. Users cannot create other directories and cannot rename nor delete the "cnv" and "seq" sub-directories.

The size of the directories indicated by the st_size field of struct stat, obtained with the stat() or fstat() system calls, indicates the number of files existing under the directory.

Zone files

Zone files are named using the number of the zone they represent within the set of zones of a particular type. That is, both the "cnv" and "seq" directories contain files named "0", "1", "2", ... The file numbers also represent increasing zone start sector on the device.

All read and write operations to zone files are not allowed beyond the file maximum size, that is, beyond the zone capacity. Any access exceeding the zone capacity is failed with the -EFBIG error.

Creating, deleting, renaming or modifying any attribute of files and sub-directories is not allowed.

The number of blocks of a file as reported by stat() and fstat() indicates the capacity of the zone file, or in other words, the maximum file size.

Conventional zone files

The size of conventional zone files is fixed to the size of the zone they represent. Conventional zone files cannot be truncated.

These files can be randomly read and written using any type of I/O operation: buffered I/Os, direct I/Os, memory mapped I/Os (mmap), etc. There are no I/O constraint for these files beyond the file size limit mentioned above.

Sequential zone files

The size of sequential zone files grouped in the "seq" sub-directory represents the file's zone write pointer position relative to the zone start sector.

Sequential zone files can only be written sequentially, starting from the file end, that is, write operations can only be append writes. Zonefs makes no attempt at accepting random writes and will fail any write request that has a start offset not corresponding to the end of the file, or to the end of the last write issued and still in-flight (for asynchronous I/O operations).

Since dirty page writeback by the page cache does not guarantee a sequential write pattern, zonefs prevents buffered writes and writeable shared mappings on sequential files. Only direct I/O writes are accepted for these files. zonefs relies on the sequential delivery of write I/O requests to the device implemented by the block layer elevator. An elevator implementing the sequential write feature for zoned block device (ELEVATOR_F_ZBD_SEQ_WRITE elevator feature) must be used. This type of elevator (e.g. mq-deadline) is set by default for zoned block devices on device initialization.

There are no restrictions on the type of I/O used for read operations in sequential zone files. Buffered I/Os, direct I/Os and shared read mappings are all accepted.

Truncating sequential zone files is allowed only down to 0, in which case, the zone is reset to rewind the file zone write pointer position to the start of the zone, or up to the zone capacity, in which case the file's zone is transitioned to the FULL state (finish zone operation).

Format options

Several optional features of zonefs can be enabled at format time.

- Conventional zone aggregation: ranges of contiguous conventional zones can be aggregated into a single larger file instead of the default one file per zone.
- File ownership: The owner UID and GID of zone files is by default 0 (root) but can be changed to any valid UID/GID.
- File access permissions: the default 640 access permissions can be changed.

IO error handling

Zoned block devices may fail I/O requests for reasons similar to regular block devices, e.g. due to bad sectors. However, in addition to such known I/O failure pattern, the standards governing zoned block devices behavior define additional conditions that result in I/O errors.

- A zone may transition to the read-only condition (BLK_ZONE_COND_READONLY): While the data already written in the zone is still readable, the zone can no longer be written. No user action on the zone (zone management command or read/write access) can change the zone condition back to a normal read/write state. While the reasons for the device to transition a zone to read-only state are not defined by the standards, a typical cause for such transition would be a defective write head on an HDD (all zones under this head are changed to read-only).
- A zone may transition to the offline condition (BLK_ZONE_COND_OFFLINE): An offline zone cannot be read nor written. No user action can transition an offline zone back to an operational good state. Similarly to zone read-only transitions, the reasons for a drive to transition a zone to the offline condition are undefined. A typical cause would be a defective read-write head on an HDD causing all zones on the platter under the broken head to be inaccessible.
- Unaligned write errors: These errors result from the host issuing write requests with a start sector that does not correspond to a zone write pointer position when the write request is executed by the device. Even though zonefs enforces sequential file write for sequential zones, unaligned write errors may still happen in the case of a partial failure of a very large direct I/O operation split into multiple BIOs/requests or asynchronous I/O operations. If one of the write request within the set of sequential write requests issued to the device fails, all write requests queued after it will become unaligned and fail.
- Delayed write errors: similarly to regular block devices, if the device side write cache is enabled, write errors may occur in ranges of previously completed writes when the device write cache is flushed, e.g. on fsync(). Similarly to the previous

immediate unaligned write error case, delayed write errors can propagate through a stream of cached sequential data for a zone causing all data to be dropped after the sector that caused the error.

All I/O errors detected by zonefs are notified to the user with an error code return for the system call that triggered or detected the error. The recovery actions taken by zonefs in response to I/O errors depend on the I/O type (read vs write) and on the reason for the error (bad sector, unaligned writes or zone condition change).

- For read I/O errors, zonefs does not execute any particular recovery action, but only if the file zone is still in a good condition and there is no inconsistency between the file inode size and its zone write pointer position. If a problem is detected, I/O error recovery is executed (see below table).
- For write I/O errors, zonefs I/O error recovery is always executed.
- A zone condition change to read-only or offline also always triggers zonefs I/O error recovery.

Zonefs minimal I/O error recovery may change a file size and file access permissions.

- File size changes: Immediate or delayed write errors in a sequential zone file may cause the file inode size to be inconsistent with the amount of data successfully written in the file zone. For instance, the partial failure of a multi-BIO large write operation will cause the zone write pointer to advance partially, even though the entire write operation will be reported as failed to the user. In such case, the file inode size must be advanced to reflect the zone write pointer change and eventually allow the user to restart writing at the end of the file. A file size may also be reduced to reflect a delayed write error detected on fsync(): in this case, the amount of data effectively written in the zone may be less than originally indicated by the file inode size. After such I/O error, zonefs always fixes the file inode size to reflect the amount of data persistently stored in the file zone.
- Access permission changes: A zone condition change to read-only is indicated with a change in the file access permissions to render the file read-only. This disables changes to the file attributes and data modification. For offline zones, all permissions (read and write) to the file are disabled.

Further action taken by zonefs I/O error recovery can be controlled by the user with the "errors=xxx" mount option. The table below summarizes the result of zonefs I/O error processing depending on the mount option and on the zone conditions:

"errors=xxx" mount option	device zone condition	Post error state: file size	file access permissions: read	file access permissions: write	device zone access permissions: read	device zone access permissions: write
remount-ro (default)	good	fixed	yes	no	yes	yes
	read-only	as is	yes	no	yes	no
	offline	0	no	no	no	no
zone-ro	good	fixed	yes	no	yes	yes
	read-only	as is	yes	no	yes	no
	offline	0	no	no	no	no
zone-offline	good	0	no	no	yes	yes
	read-only	0	no	no	yes	no
	offline	0	no	no	no	no
repair	good	fixed	yes	yes	yes	yes
	read-only	as is	yes	no	yes	no
	offline	0	no	no	no	no

Further notes:

- The "errors=remount-ro" mount option is the default behavior of zonefs I/O error processing if no errors mount option is specified.
- With the "errors=remount-ro" mount option, the change of the file access permissions to read-only applies to all files. The file system is remounted read-only.
- Access permission and file size changes due to the device transitioning zones to the offline condition are permanent. Remounting or reformatting the device with mkfs.zonefs (mkzonefs) will not change back offline zone files to a good state.
- File access permission changes to read-only due to the device transitioning zones to the read-only condition are permanent. Remounting or reformatting the device will not re-enable file write access.
- File access permission changes implied by the remount-ro, zone-ro and zone-offline mount options are temporary for zones in a good condition. Unmounting and remounting the file system will restore the previous default (format time values) access rights to the files affected.
- The repair mount option triggers only the minimal set of I/O error recovery actions, that is, file size fixes for zones in a good condition. Zones indicated as being read-only or offline by the device still imply changes to the zone file access permissions as noted in the table above.

Mount options

zonefs define the "errors=<behavior>" mount option to allow the user to specify zonefs behavior in response to I/O errors, inode size inconsistencies or zone condition changes. The defined behaviors are as follow:

- remount-ro (default)

- zone-ro
- zone-offline
- repair

The run-time I/O error actions defined for each behavior are detailed in the previous section. Mount time I/O errors will cause the mount operation to fail. The handling of read-only zones also differs between mount-time and run-time. If a read-only zone is found at mount time, the zone is always treated in the same manner as offline zones, that is, all accesses are disabled and the zone file size set to 0. This is necessary as the write pointer of read-only zones is defined as invalib by the ZBC and ZAC standards, making it impossible to discover the amount of data that has been written to the zone. In the case of a read-only zone discovered at run-time, as indicated in the previous section. The size of the zone file is left unchanged from its last updated value.

A zoned block device (e.g. an NVMe Zoned Namespace device) may have limits on the number of zones that can be active, that is, zones that are in the implicit open, explicit open or closed conditions. This potential limitation translates into a risk for applications to see write IO errors due to this limit being exceeded if the zone of a file is not already active when a write request is issued by the user.

To avoid these potential errors, the "explicit-open" mount option forces zones to be made active using an open zone command when a file is opened for writing for the first time. If the zone open command succeeds, the application is then guaranteed that write requests can be processed. Conversely, the "explicit-open" mount option will result in a zone close command being issued to the device on the last close() of a zone file if the zone is not full nor empty.

Zonefs User Space Tools

The mkzonefs tool is used to format zoned block devices for use with zonefs. This tool is available on Github at:

https://github.com/damien-lemoal/zonefs-tools

zonefs-tools also includes a test suite which can be run against any zoned block device, including null_blk block device created with zoned mode.

Examples

The following formats a 15TB host-managed SMR HDD with 256 MB zones with the conventional zones aggregation feature enabled:

```
# mkzonefs -o aggr_cnv /dev/sdX
# mount -t zonefs /dev/sdX /mnt
# ls -l /mnt/
total 0
dr-xr-xr-x 2 root root     1 Nov 25 13:23 cnv
dr-xr-xr-x 2 root root 55356 Nov 25 13:23 seq
```

The size of the zone files sub-directories indicate the number of files existing for each type of zones. In this example, there is only one conventional zone file (all conventional zones are aggregated under a single file):

```
# ls -l /mnt/cnv
total 137101312
-rw-r----- 1 root root 140391743488 Nov 25 13:23 0
```

This aggregated conventional zone file can be used as a regular file:

```
# mkfs.ext4 /mnt/cnv/0
# mount -o loop /mnt/cnv/0 /data
```

The "seq" sub-directory grouping files for sequential write zones has in this example 55356 zones:

```
# ls -lv /mnt/seq
total 14511243264
-rw-r----- 1 root root 0 Nov 25 13:23 0
-rw-r----- 1 root root 0 Nov 25 13:23 1
-rw-r----- 1 root root 0 Nov 25 13:23 2
...
-rw-r----- 1 root root 0 Nov 25 13:23 55354
-rw-r----- 1 root root 0 Nov 25 13:23 55355
```

For sequential write zone files, the file size changes as data is appended at the end of the file, similarly to any regular file system:

```
# dd if=/dev/zero of=/mnt/seq/0 bs=4096 count=1 conv=notrunc oflag=direct
1+0 records in
1+0 records out
4096 bytes (4.1 kB, 4.0 KiB) copied, 0.00044121 s, 9.3 MB/s

# ls -l /mnt/seq/0
-rw-r----- 1 root root 4096 Nov 25 13:23 /mnt/seq/0
```

The written file can be truncated to the zone size, preventing any further write operation:

```
# truncate -s 268435456 /mnt/seq/0
# ls -l /mnt/seq/0
```

```
-rw-r----- 1 root root 268435456 Nov 25 13:49 /mnt/seq/0
```

Truncation to 0 size allows freeing the file zone storage space and restart append-writes to the file:

```
# truncate -s 0 /mnt/seq/0
# ls -l /mnt/seq/0
-rw-r----- 1 root root 0 Nov 25 13:49 /mnt/seq/0
```

Since files are statically mapped to zones on the disk, the number of blocks of a file as reported by stat() and fstat() indicates the capacity of the file zone:

```
# stat /mnt/seq/0
File: /mnt/seq/0
Size: 0             Blocks: 524288     IO Block: 4096   regular empty file
Device: 870h/2160d  Inode: 50431       Links: 1
Access: (0640/-rw-r-----)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2019-11-25 13:23:57.048971997 +0900
Modify: 2019-11-25 13:52:25.553805765 +0900
Change: 2019-11-25 13:52:25.553805765 +0900
Birth: -
```

The number of blocks of the file ("Blocks") in units of 512B blocks gives the maximum file size of 524288 * 512 B = 256 MB, corresponding to the device zone capacity in this example. Of note is that the "IO block" field always indicates the minimum I/O size for writes and corresponds to the device physical sector size.