SDC 20

BY Developers FOR Developers

Storage Developer Conference
September 22-23, 2020

Zoned Block Device Support in Hadoop HDFS

Shin'ichiro Kawasaki

Western Digital Corporation

Agenda

- Background
- Overview of HDFS zoned block device support
- Performance evaluation
- Conclusion and future work

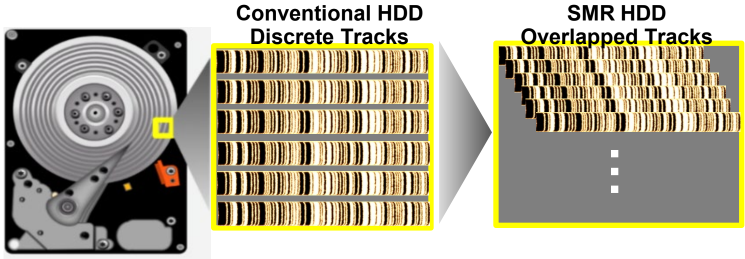
Background

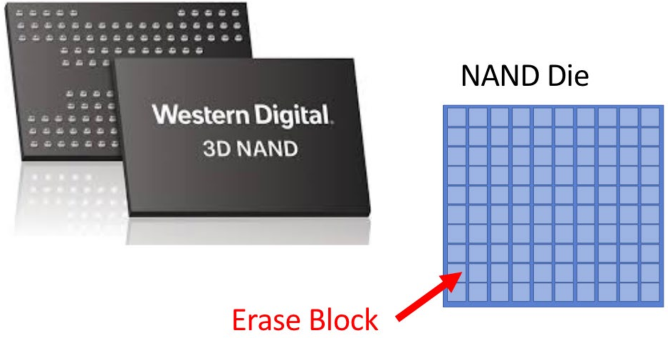
Why Zoned Block Device?

Increased capacity and performance reduces storage cost ($/TB)

- HDDs: shingled magnetic recording
  - Areal density growth with zones of overlapped tracks
- SSDs: expose sequential write in erase blocks and let the host contribute to data placement

Improved capacities (lower OP for SSDs)
Reduces Write Amplification
Improves latency and throughput

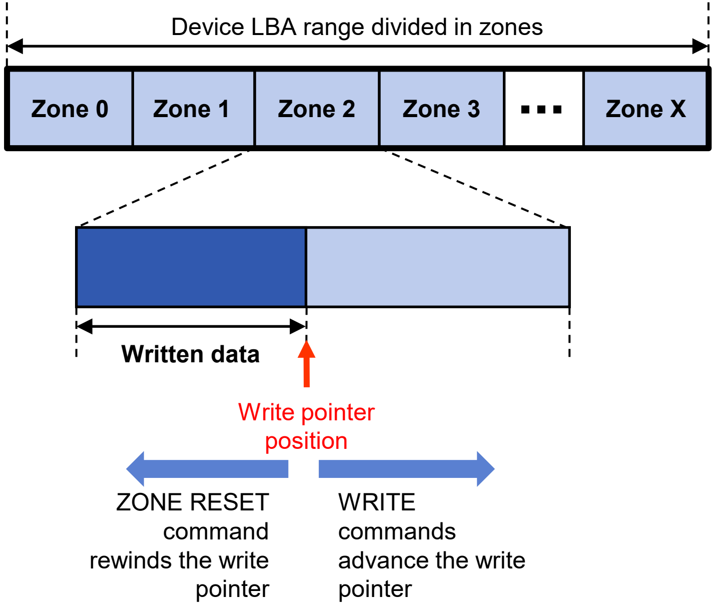




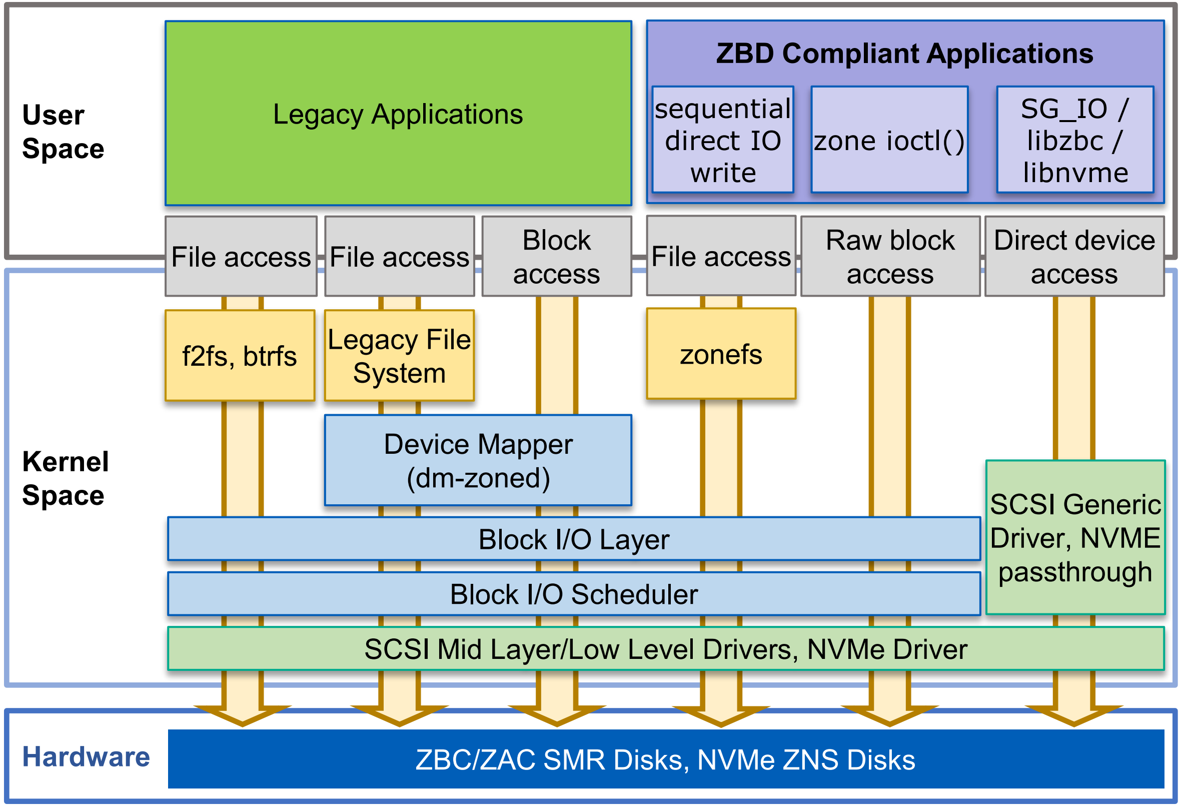
Zoned Block Devices

Random reads but sequential writes

- All standards (SCSI ZBC, ATA ZAC and NVMe Zoned Namespace) define nearly identical commands and behavior
    - Linux abstraction: **Zoned Block Devices**
- LBA range divided into zones of different types
    - Conventional zones
        - Accept random writes
        - Optional on SMR HDDs
        - Not defined for NVMe ZNS
    - Sequential write required zones
        - Writes must be issued sequentially starting from the zone write pointer
        - Zones must be reset before rewriting
            - Rewind write pointer to beginning of the zone
- 5 zone management commands defined: report, reset, open, close and finish


Device LBA range divided in zones
Zone 0
Zone 1
Zone 2
Zone 3
...
Zone X
Written data
Write pointer position
ZONE RESET command rewinds the write pointer
WRITE commands advance the write pointer

Linux Kernel Support and Ecosystem



Sequential writes handled by lower layers:

- No software changes needed: legacy applications can be used
- Performance overhead

Raw zoned block device access

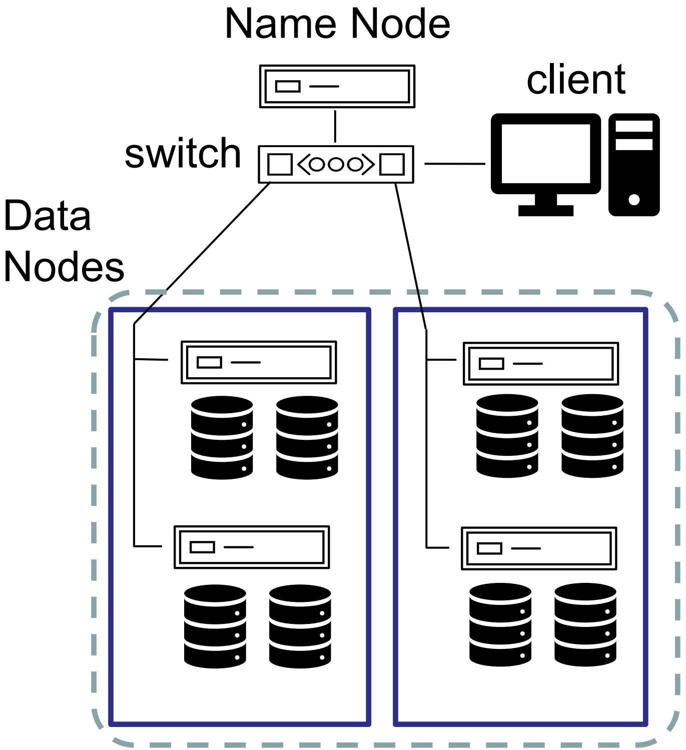
- Application must write sequentially: software changes needed
- But can be more effective
- Applications already writing mostly sequentially are good candidates:
  - Rocks DB, Level DB
  - Hadoop HDFS

Hadoop HDFS

“Distributed file system with common hardware”

- Apache Hadoop
  - “A framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models”
  - Achieves data processing “scale-up” with parallel processing, moving processing rather than data
- HDFS: Hadoop Distributed File System
  - Distributed file system with common hardware
  - Single “Name Node” controls many “Data Nodes”
  - Large data sets is the target use case
    - HDFS files have size of giga bytes or tera bytes
  - Write-once-read-many semantics is assumed

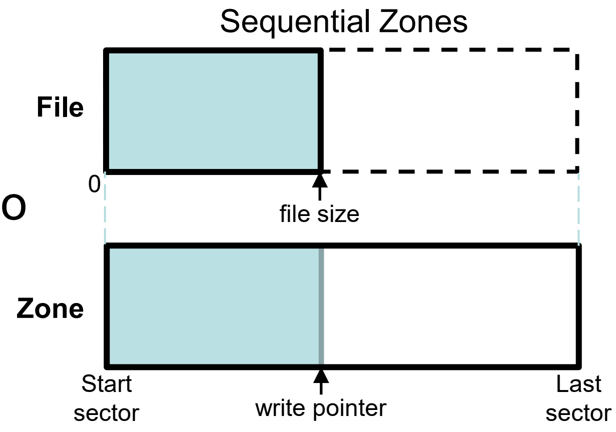
High affinity with Zoned Block Device sequential write requirements

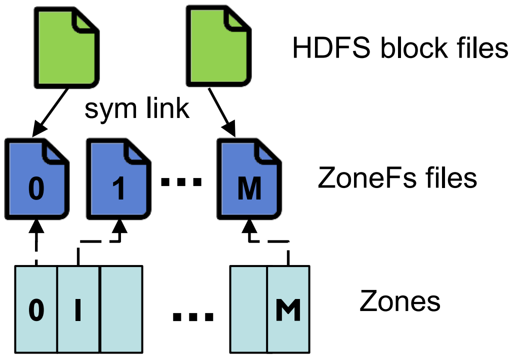


HDFS zoned block device support

Minimize Changes Using *zonefs*

- zonefs: simple Linux zoned block device file system since kernel version 5.6
  - Exposes each zone of a device as a file
  - Zone abstraction and operations mapped to file information and system calls
    - Zone write pointer position -> file size
    - Zone write -> file append write by direct I/O
    - Zone reset -> file truncate()
- HDFS support approach:
  - Map each HDFS block file to a zone file using a symbolic link
    - Many file operations can be reused, e.g. read()
    - Main changes are for HDFS block files write
      - Direct I/Os





HDFS Block Files Direct IO Writes

Problems

A. Direct write
- Open zonefs files with O_DIRECT.
- Java standard file open does not support this

B. Device sector aligned writes (O_DIRECT !)
- Write buffer memory address passed to write() system calls must be aligned to the device physical sector size
- Java memory allocation does not support this

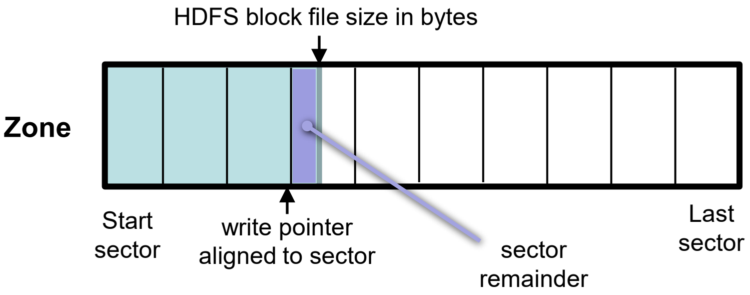
C. Data size
- All writes must be aligned to the device physical sector size
- Data write requests to HDFS block files may not be aligned

Solutions

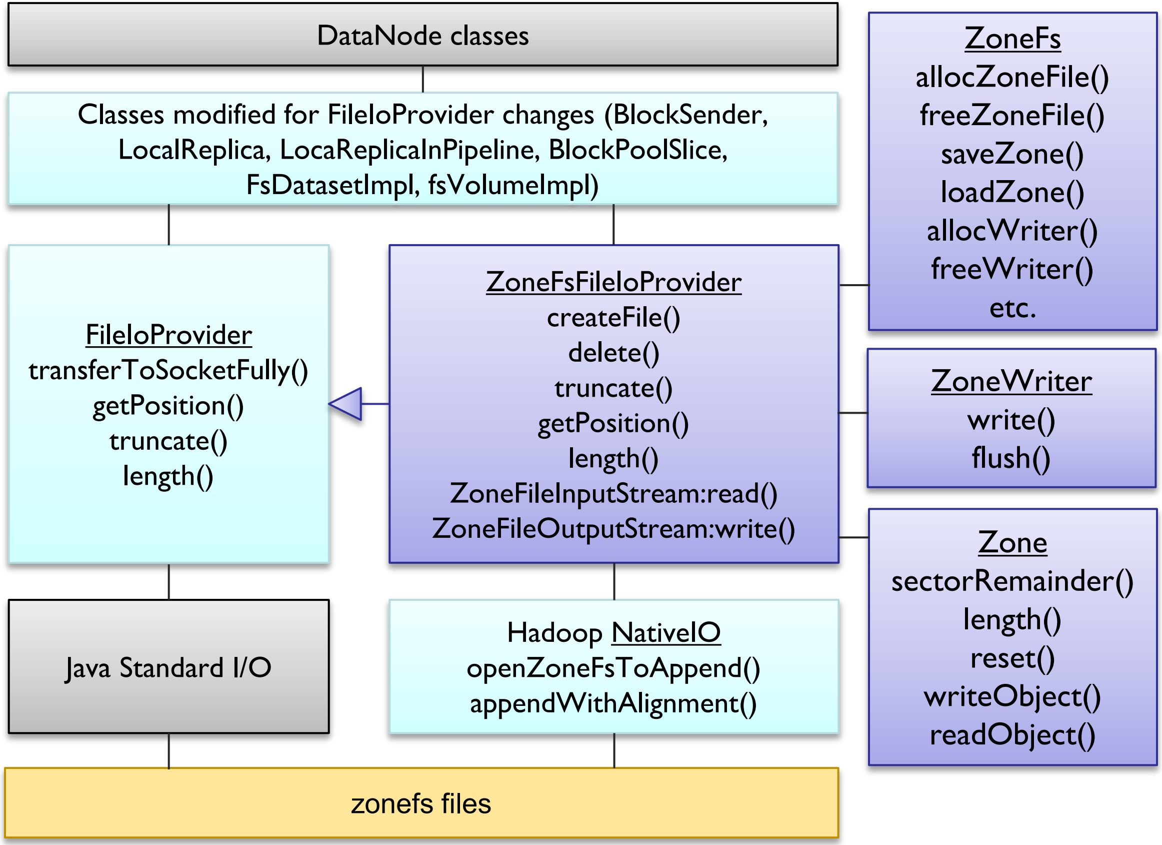
A. New method for file open
- openZoneFsToAppend()

B. New JNI method
- appendWithAlignment()

C. Buffer sector remainder data in memory
- Save on disk at write stream close
- Load the saved remainder at DataNode reboot


HDFS block file size in bytes
Zone
Start sector
write pointer aligned to sector
sector remainder
Last sector

HDFS DataNode Code Changes



4 new classes

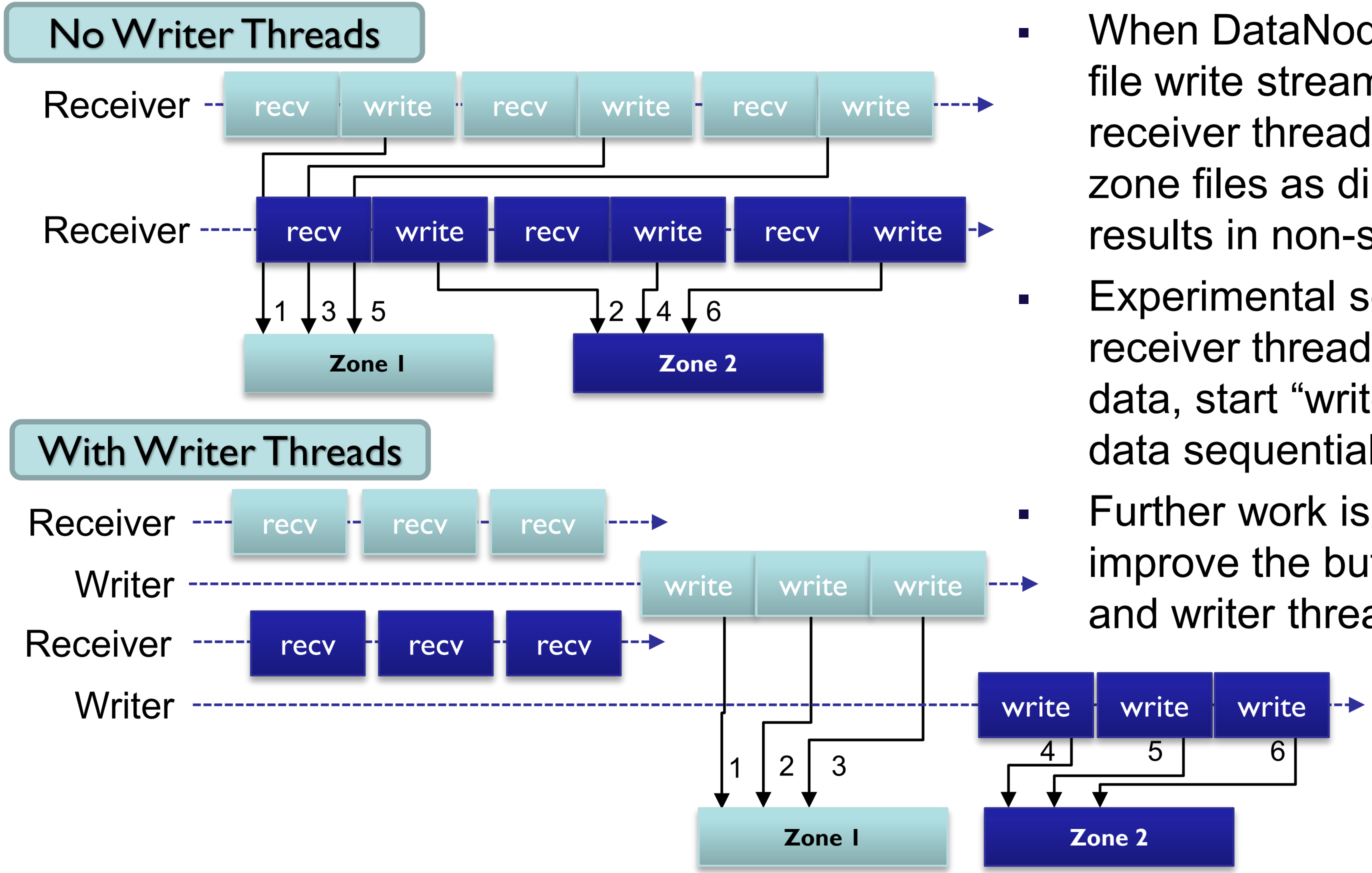
- ZoneFs and Zone
- ZoneFsFileIoProvider
  - Encapsulate zonefs unique operations
- ZoneWriter
  - Write data buffer
  - Thread for delayed write

7 modified classes

- NativeIO
  - New methods for Direct I/O and aligned write.
- FileIoProvider
  - Interface changes for ZoneFsFileIoProvider
- Other
  - Adjust to new FileIoProvider interface

Added lines: 1.5K+

Parallel Write with Writer Threads



- When DataNode receives block file write streams in parallel, receiver threads write to multiple zone files as direct I/O. This results in non-sequential writes.
- Experimental solution: after the receiver thread buffer whole zone data, start “writer thread” to write data sequentially.
- Further work is in progress to improve the buffer memory size and writer threads.

Performance Evaluation

Evaluation Environment

- Server Hardware
  - CPU: Xeon Silver 4210, 10 core
  - DRAM: 64GiB DRAM
- 5 Servers
  - 1 NameNode
  - 3 Data Nodes / Node Manager
  - 1 Client / Resource Manager
  - Conneted over 10GiB Ethernet
- Storage
  - 15 TB SMR HDD
  - 12 TB CMR HDD
    - Mechanically identical to SMR HDD, i.e. same raw performance

- HDFS Configuration
  - x3 Replica
  - 256MiB block size
    - SMR HDD zone size
- Software Verions
  - Hadoop/HDFS: 3.3.0
    - JDK: OpenJDK 1.8.0
  - OS: Fedora 32
    - Linux kernel: 5.7.16

SMR: Shingled Magnetic Recording
CMR: Conventional Magnetic Recording

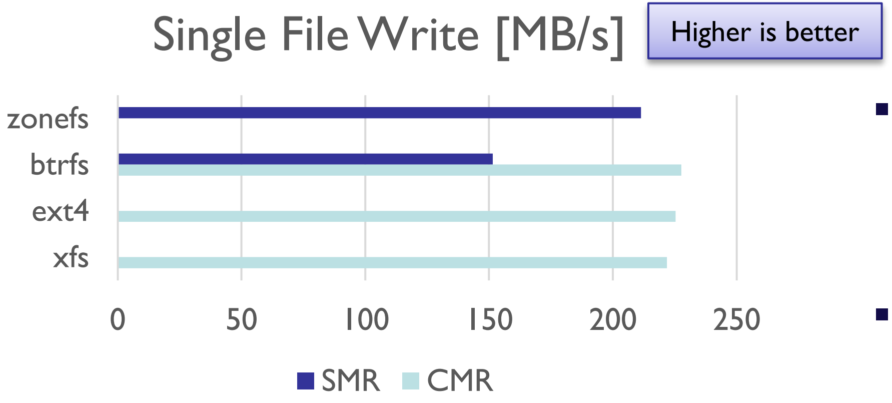
Evaluation Setups

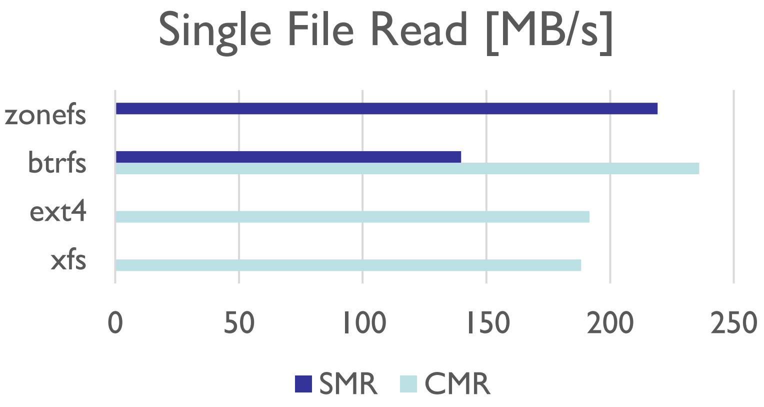
- Evaluated 5 different setups on DataNodes

#	File system	HDD	Note
1	zonefs	SMR	With HDFS support for zonefs
2	btrfs	SMR	Btrfs SMR support is a prototype
3	btrfs	CMR	
4	ext4	CMR	
5	xfs	CMR	

- Used 2 benchamrks for evaluation
  - DFSIO
  - TeraSort

DFSIO: Single I/O



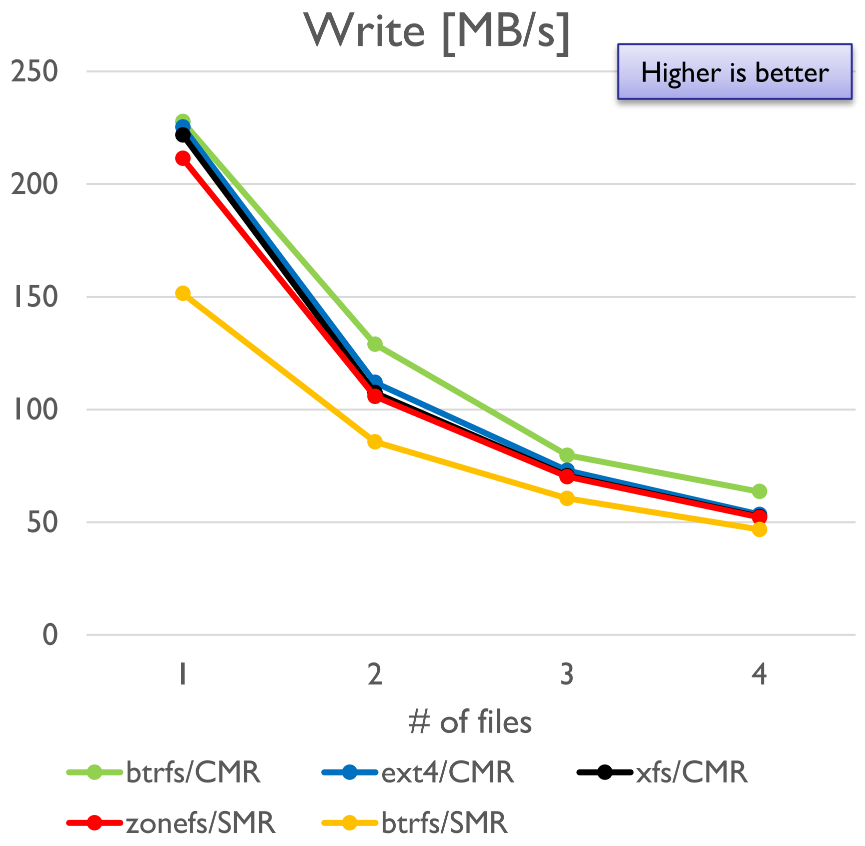


- File systems with buffered write show better write performance with CMR drive
- Btrfs on CMR drive shows the highest read performance
- For SMR drives, zonefs provides the highest performance

Evaluation Condition:

- File size: 128GB
- Flashed OS page cache after write before read

DFSIO: Parallel Write

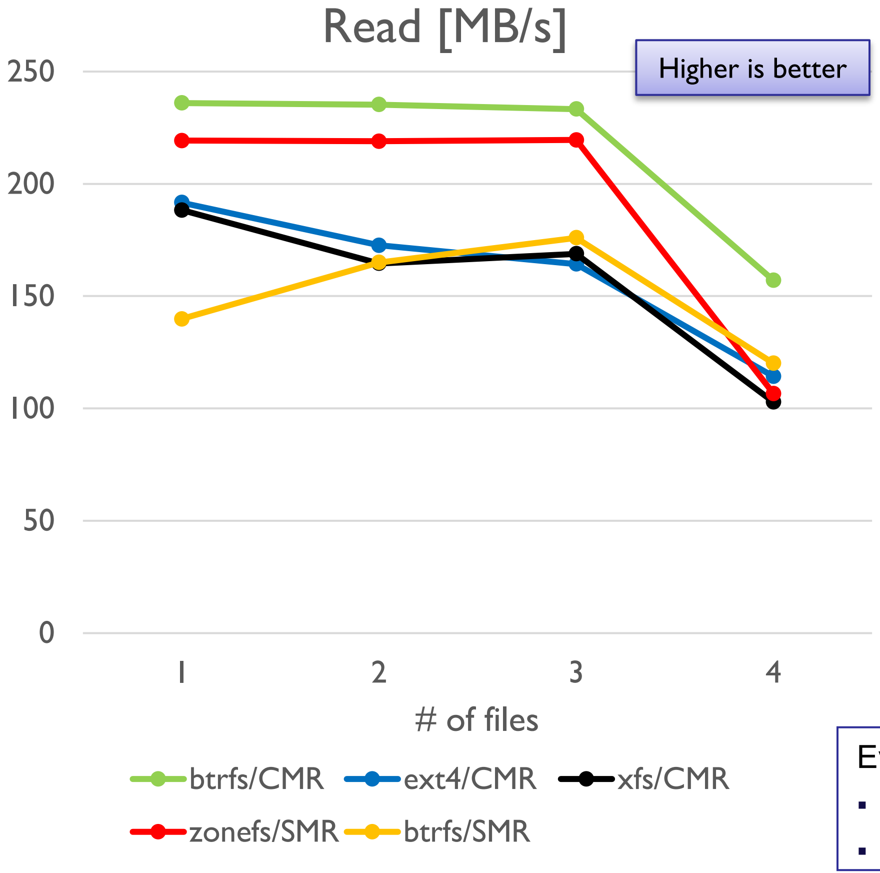


- As number of parallel write streams increases, write performance decreases.
- Zonefs write performance on SMR drives is comparable with file systems on CMR drives, with writer threads to reduce the parallel write overhead by non-sequential writes

Evaluation Condition:

- file size: 128GB

DFSIO: Parallel Read

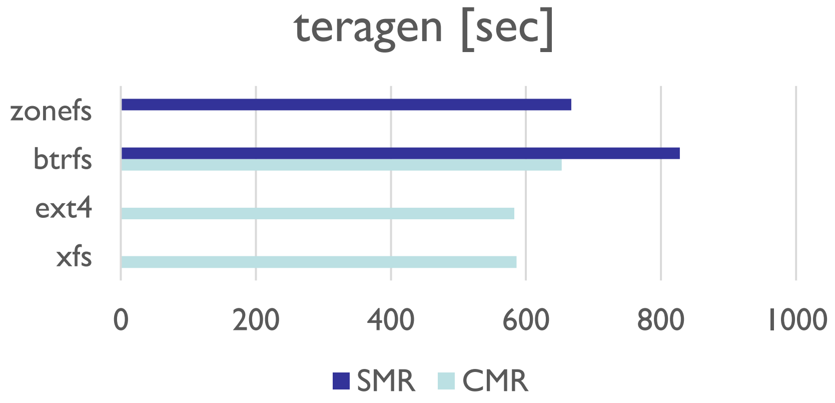


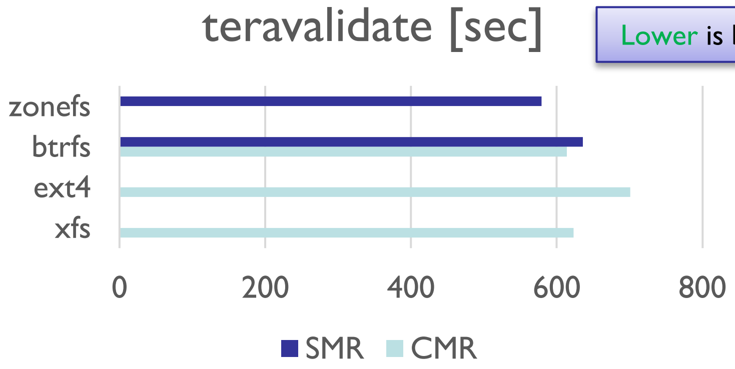
- Up to 3 parallel read has no performance degradation since workloads are assigned to each of 3 DataNodes
- Zonefs files on SMR drives have no fragmentation whereas file data on ext4/xfs and CMR drives is not written sequentially and has lower throuhput

Evaluation Condition:

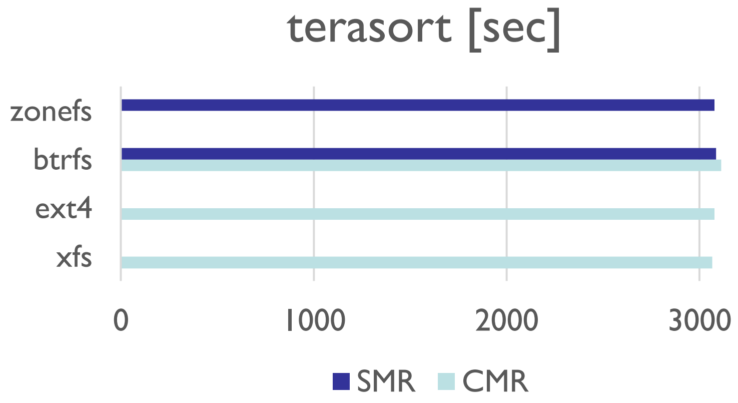
- file size: 128GB
- Flashed OS page cache after write before read

Tera Sort


teragen [sec]


teravalidate [sec]

Lower is better


terasort [sec]

- teragen: CMR HDD has better performance
- terasort: Performance is comparable across all file systems and CMR/SMR
- teravalidate: zonefs and SMR HDD gives highest performance

Evaluation Condition:

- 128GB sort data (64GB file x 2)

Conclusion and Future Work

Conclusion

- Zoned block devices can be supported in HDFS with 1.5K+ lines changes using zonefs
  - Low overhead file system
- Large file I/O performance of zonefs on SMR drives is comparable to other file systems on CMR drives
  - Read performance is better than ext4 and xfs on CMR drives
  - Zonefs also results in better performance than btrfs on SMR drives

Future Work

- Evaluate stability and performance with larger number of nodes
  - Peta byte scale data set
- Work on write buffer memory size for writer threads
- Propose code changes to Hadoop project

Thank You!