Western Digital

Linux SMR Support Status

Damien Le Moal

Vault – Linux Storage and Filesystems Conference - 2017
March 23rd, 2017

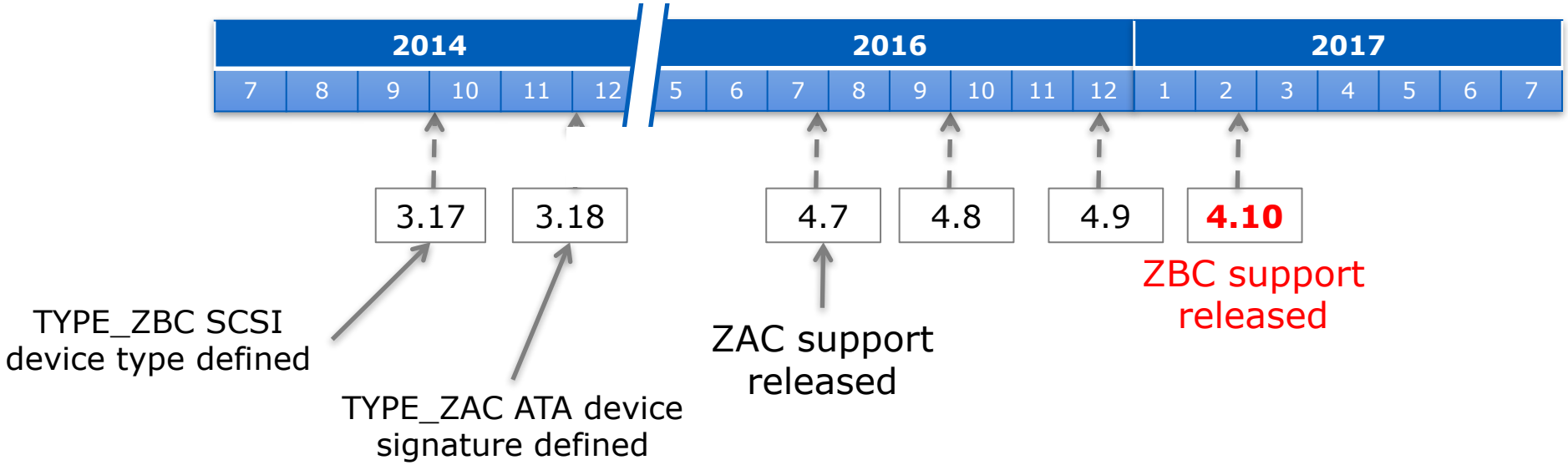
Outline

- Standards and Kernel Support Status
- Kernel Details
  - What was needed
  - Block stack
  - File systems
  - Application level tools
- Ongoing Work
  - Device mapper and file systems
- Performance Evaluation Results
  - dbench
  - Sustained write workloads

Standards and Kernel Support Status

ZBC and ZAC specifications are stable

- ZAC & ZBC specifications r05 forwarded to INCITS for publication
  - Latest SAT4 r06 specifications include zone block commands translation description
- Kernel support finalized with addition of ZBC commands in kernel 4.10
  - Host-managed disks are exposed as "regular" block devices
  - Full support for ZBC to ZAC command translation



What Was Needed ?

All constraints come from host-managed

- SMR comes in different flavors

- Drive-managed
  - The disk presents itself as a regular block device
    - No constraints, no specific optimization possible

No requirement

- Host-aware
  - The disk presents itself as a regular block device
    - No constraints, specific optimization possible

Support for ZBC/ZAC commands (optimization)

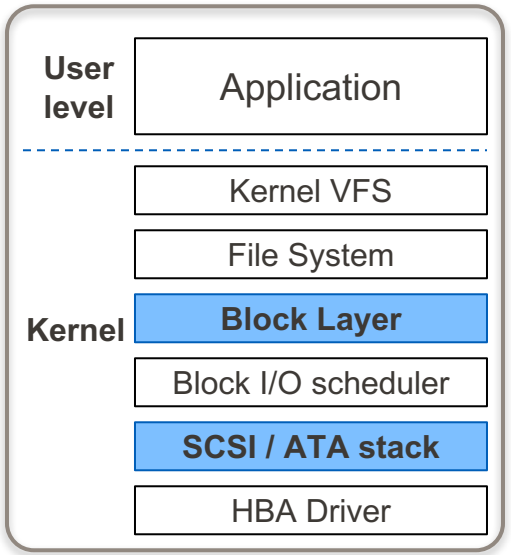
- Host-managed
  - Not a regular block device (different device type)
  - Sequential write constraint

Support for ZBC/ZAC commands, sequential write enforcement, or at least guarantees

Block Stack

No real change, only additions

- Internal API for report zones and reset write pointer
  - No in kernel support for open zone, close zone and finish zone
  - Any zone command allowed in pass-through mode with SG_IO
    - Fully functional SAT layer
- Some limits are imposed on disk zone configurations
  - All zones must have the same size
    - Except for an eventual last smaller "runt" zone
  - Zone size must be a power of 2 number of LBAs
  - Reads are unrestricted (URSWRZ bit set to 1)
- Host-managed disks are seen as almost-regular block devices
  - **Sequential write constraint exposed to the drive user**
    - Disk user must ensure sequential write patterns to sequential zones
    - File systems, device mappers and applications
  - Write ordering guarantees implemented by limiting write queue depth to 1 per zone
    - Also solves many HBA level command ordering problems, including AHCI



Block Stack

Device compliance checked on boot

- Constraint compliance, zone size and device type are checked at boot time
  - On device revalidate

```
[    3.687797] scsi 5:0:0:0: Direct-Access-ZBC ATA      HGST HSH721414AL TE8C PQ: 0 ANSI: 7
[    3.696359] sd 5:0:0:0: Attached scsi generic sg4 type 20
[    3.696485] sd 5:0:0:0: [sdd] Host-managed zoned block device
[    3.865072] sd 5:0:0:0: [sdd] 27344764928 512-byte logical blocks: (14.0 TB/12.7 TiB)
[    3.873046] sd 5:0:0:0: [sdd] 4096-byte physical blocks
[    3.878343] sd 5:0:0:0: [sdd] 52156 zones of 524288 logical blocks
[    3.884591] sd 5:0:0:0: [sdd] Write Protect is off
[    3.889440] sd 5:0:0:0: [sdd] Mode Sense: 00 3a 00 00
[    3.889458] sd 5:0:0:0: [sdd] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
[    4.253140] sd 5:0:0:0: [sdd] Attached SCSI disk
```

- Information available to applications through sysfs files
  - “zoned” file for device type: “host-managed”, “host-aware” or “none”
  - “chunk_sectors” for zone size

```
> cat /sys/block/sdd/queue/zoned
host-managed

> cat /sys/block/sdd/queue/chunk_sectors
524288
```

Block Stack API

Internal kernel functions

- Zone information reported as struct blk_zone

```
> less include/uapi/linux/blkzoned.h
...

struct blk_zone {
        __u64   start;          /* Zone start sector */
        __u64   len;            /* Zone length in number of sectors */
        __u64   wp;             /* Zone write pointer position */
        __u8    type;           /* Zone type */
        __u8    cond;           /* Zone condition */
        __u8    non_seq;        /* Non-sequential write resources active */
        __u8    reset;          /* Reset write pointer recommended */
        __u8    reserved[36];
};
```

- Zone report and reset functions provided
  - Suitable for file systems or device mappers (struct block_device)

```
> less include/uapi/linux/blkzoned.h
...
extern int blkdev_report_zones(struct block_device *bdev,
                               sector_t sector, struct blk_zone *zones,
                               unsigned int *nr_zones, gfp_t gfp_mask);
extern int blkdev_reset_zones(struct block_device *bdev, sector_t sectors,
                              sector_t nr_sectors, gfp_t gfp_mask);
```

Block Stack API

ioctl provided to applications

- Zone report and reset allowed from applications through ioctl
  - Zone report ioctl interface uses same struct blk_zone as the kernel

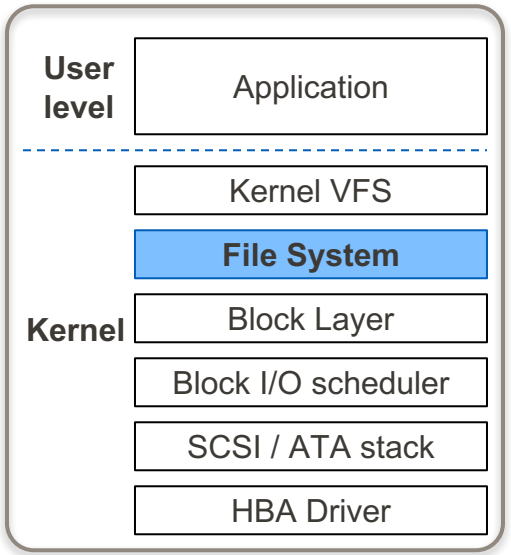
```
> less include/uapi/linux/blkzoned.h
...

/**
 * Zoned block device ioctl's:
 *
 * @BLKREPORTZONE: Get zone information. Takes a zone report as argument.
 *                 The zone report will start from the zone containing the
 *                 sector specified in the report request structure.
 * @BLKRESETZONE: Reset the write pointer of the zones in the specified
 *                sector range. The sector range must be zone aligned.
 */
#define BLKREPORTZONE   _IOWR(0x12, 130, struct blk_zone_report)
#define BLKRESETZONE    _IOW(0x12, 131, struct blk_zone_range)
```

File Systems

Native support for f2fs

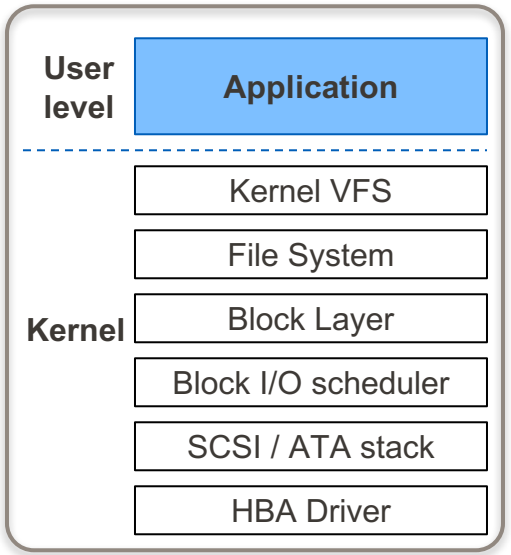
- F2FS native support for zoned block devices included in kernel 4.10
  - Completely hides sequential write constraint to application level
- Added on top of "lfs" mode of f2fs
  - Pure log-structured operation
    - No optimization with update-in-place for metadata blocks
  - Sections (segments group) aligned to device zones
    - Checks in mkfs.f2fs user application
  - Fixed location metadata placed in conventional zones
- Other problems fixed
  - Sequential use of segments within sections
  - Atomic block allocation and I/O issuing operations
    - Also benefits SSDs
  - Discard granularity
    - Per section, trigger zone reset



Application Level Tools

Different choices available

- libzbc (https://github.com/hgst/libzbc)
  - Provides an API library for executing all defined ZBC and ZAC commands
    - Passthrough (SG_IO) or using kernel block device and ioctls
- sg3utils (http://sg.danny.cz/sg/sg3_utils.html)
  - Legacy, well known SCSI command application tool chain
  - The current version 1.42 supports ZBC
    - sg_rep_zones, sg_reset_wp, sg_zone tools provided
  - Rely on SAT layer for ZBC command translation to ZAC command
    - Kernel SAT layer (libata) or HBA SAT layer
- Linux sys-utils
  - Part of util-linux repository (https://github.com/karelzak/util-linux)
  - blkzone utility added
    - Support zone report and zone reset
      - Implemented using the BLKZONEREPORT and BLKZONERESET ioctl



On-Going Work: Device Mapper

dm-zoned: Expose a zoned block device as a regular block device

- Uses conventional zones as on-disk random-write buffers
  - Conventional zone reclaimed on idle time or on-demand
- Minimal capacity loss and resource usage
  - A few 256MB zones on 14 TB disk for internal meta-data
  - 3~4 MB of memory per disk for metadata caching
- Patches submitted for review
- Possible extensions being discussed
  - At LSF/MM: merge with compression target ?

Ongoing Work: File Systems

BtrFS (and XFS)

- BtrFS
  - Some changes at format time
    - Super block copies all in conventional zones
    - Ensure that 1GB block groups aligned to device zones
  - Run time fixes to ensure that 1 GB block groups are written sequentially
    - E.g. if multiple files are being written in the same block group
    - Fixed block pre-allocation
  - Ensure that block allocation and I/O submission are atomic operations
  - Fixed ordering of writes with flush requests
  - Patches almost ready for review
- XFS
  - Introduction of reverse block mapping b-tree makes it possible to implement ZBC support
    - Needed for metadata updates after random file block modification and zone GC
  - Activity not started yet...

Evaluation Results: DBench

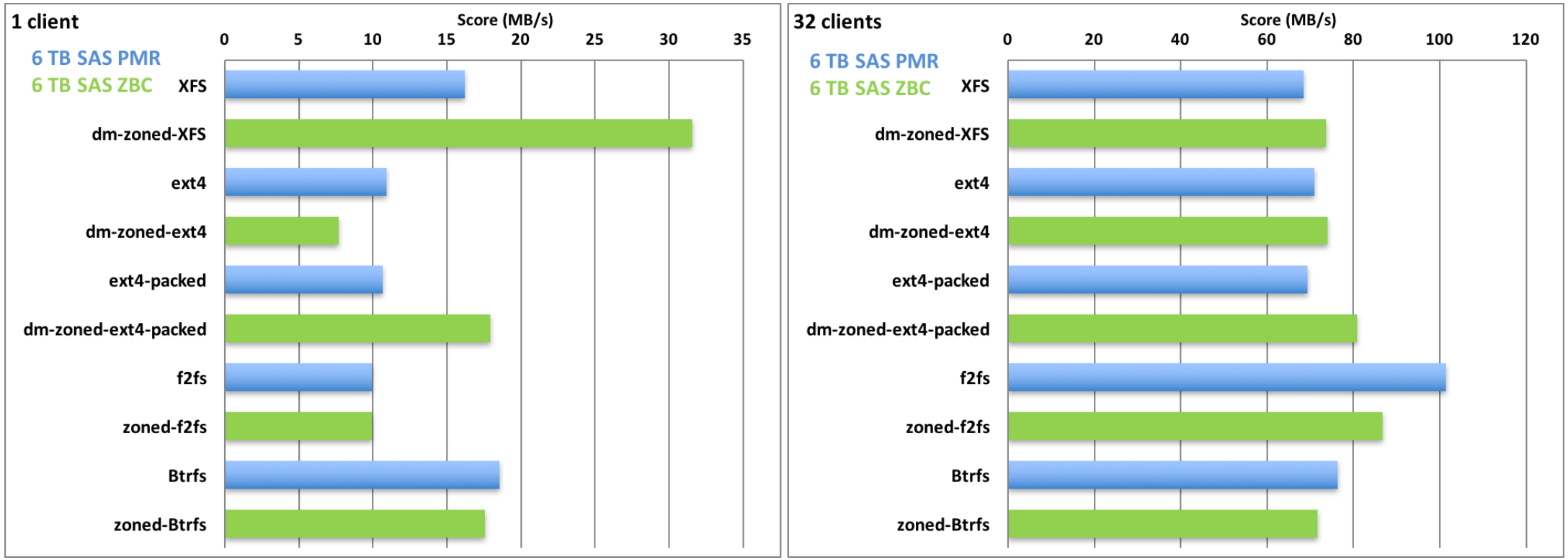
File systems vs device mapper

- Compare software changes, not disk drives !
  - Used same physical disk
  - Regular 6 TB SAS disk firmware changed to add ZBC interface and write constraints
    - Host-managed disk model
    - 22000 zones of 256 MB
    - 1% of zones are CMR at LBA 0
- Regular dbench benchmark
  - No ”SMR” specific optimization
  - 1 and 32 clients

Dbench

No significant performance penalty

- Under mixed read-write workloads, no significant performance degradation with SMR
  - Device mapper can improve performance
    - Random writes become sequential
  - BtrFS and f2fs on SMR slightly lower performance



Evaluation Results: Sustained Write Workload

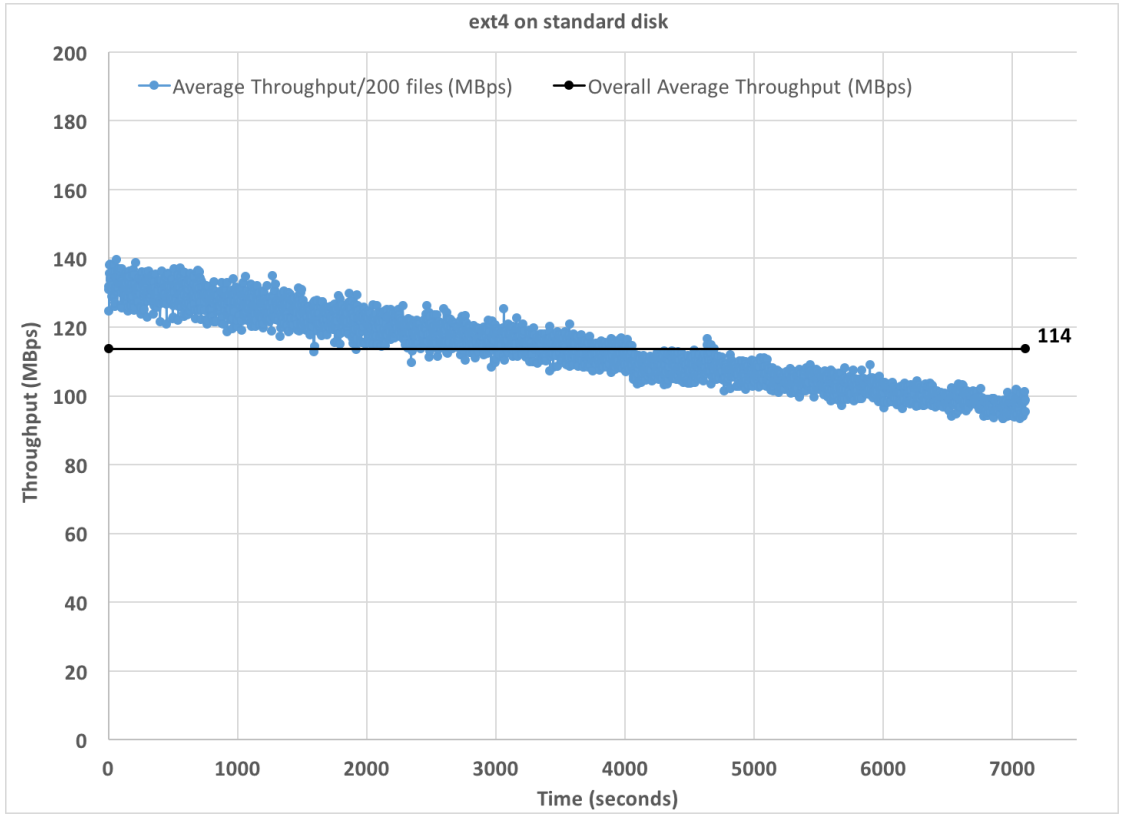
File systems vs device mapper

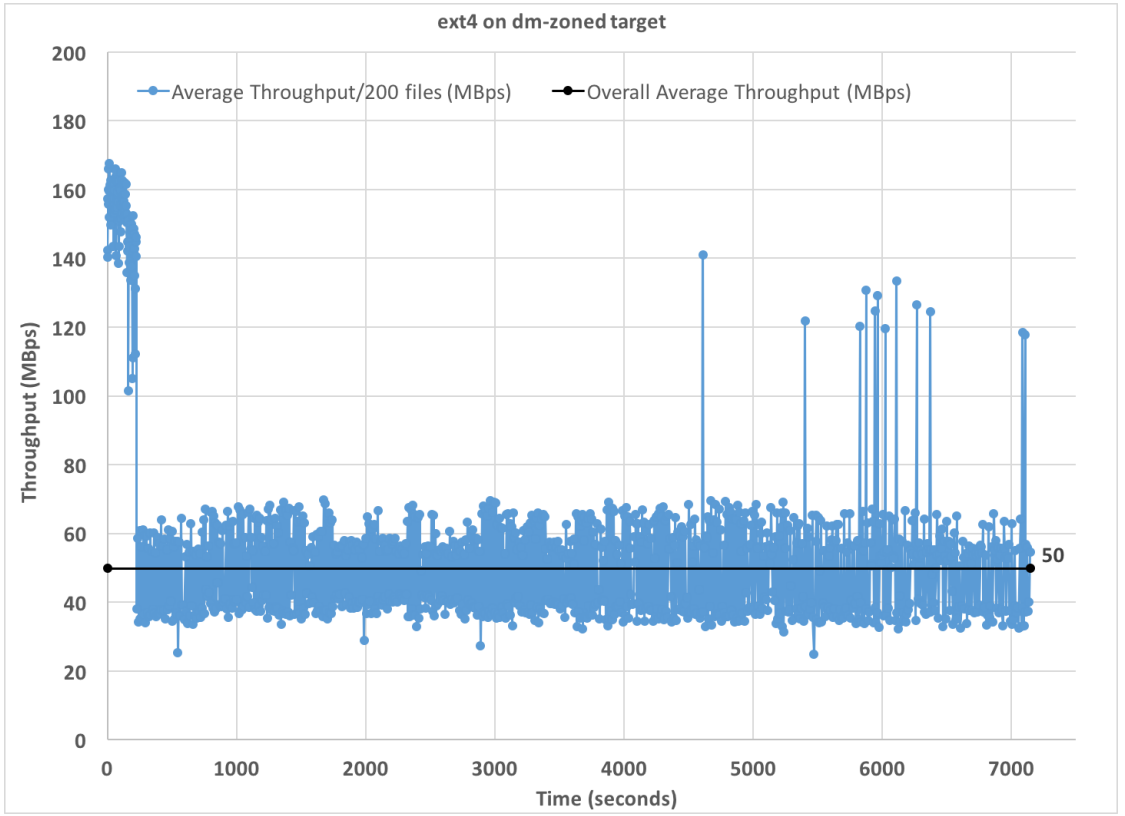
- Same drives
- Simple application writing random size files to different directories
  - 1 to 4 MB random size
    - Think cat pictures
  - 20 concurrent writer processes
  - Measure bandwidth every 200 files written

Sustained Write Workload

ext4

- dm-zoned reclaim of write buffer zones overhead is clear
  - Too many random metadata updates

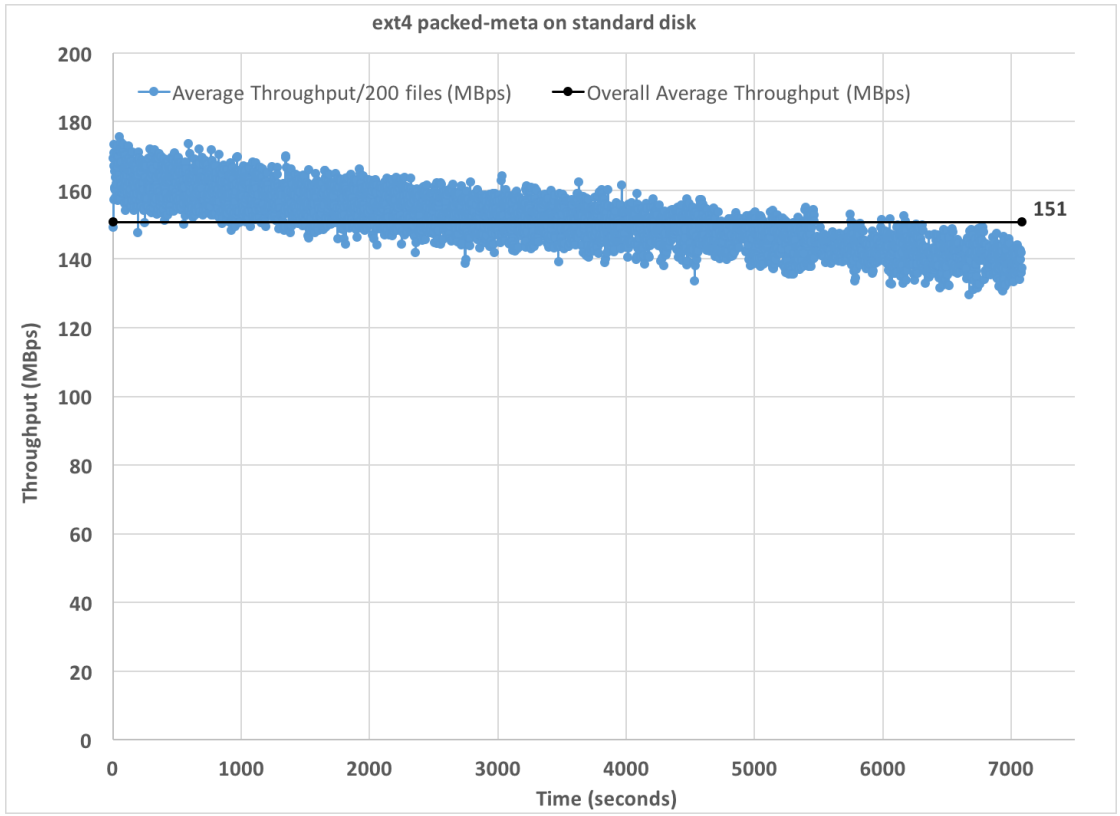


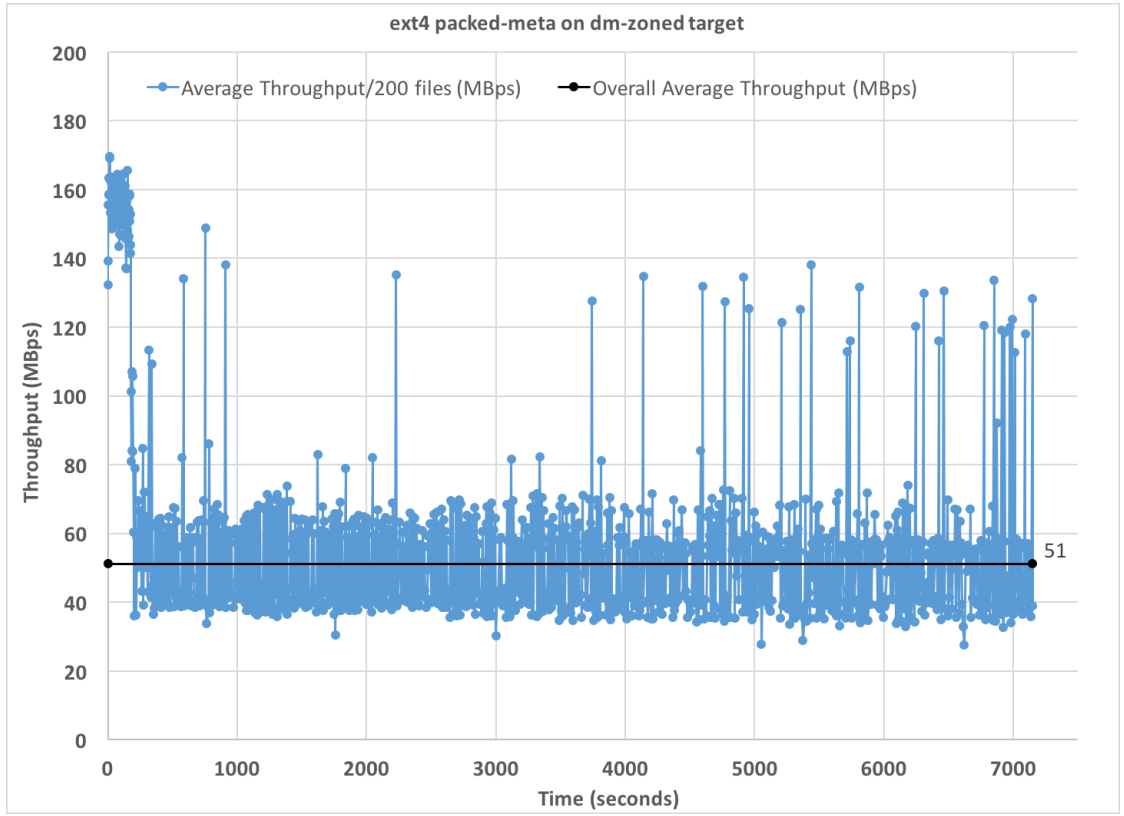


Sustained Write Workload

ext4 with packed metadata

- Better on standard disk, no significant improvements on dm-zoned
  - Concurrent writes by different processes do not generate a sequential write sequence per zone

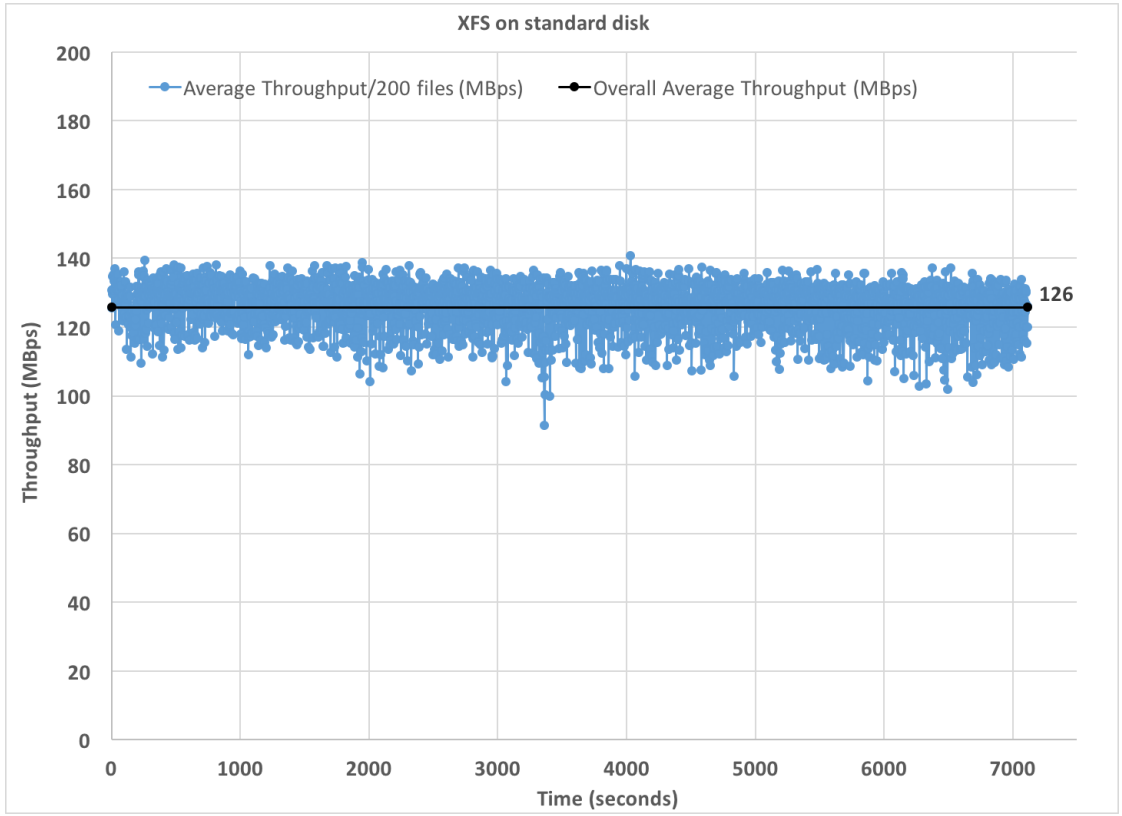


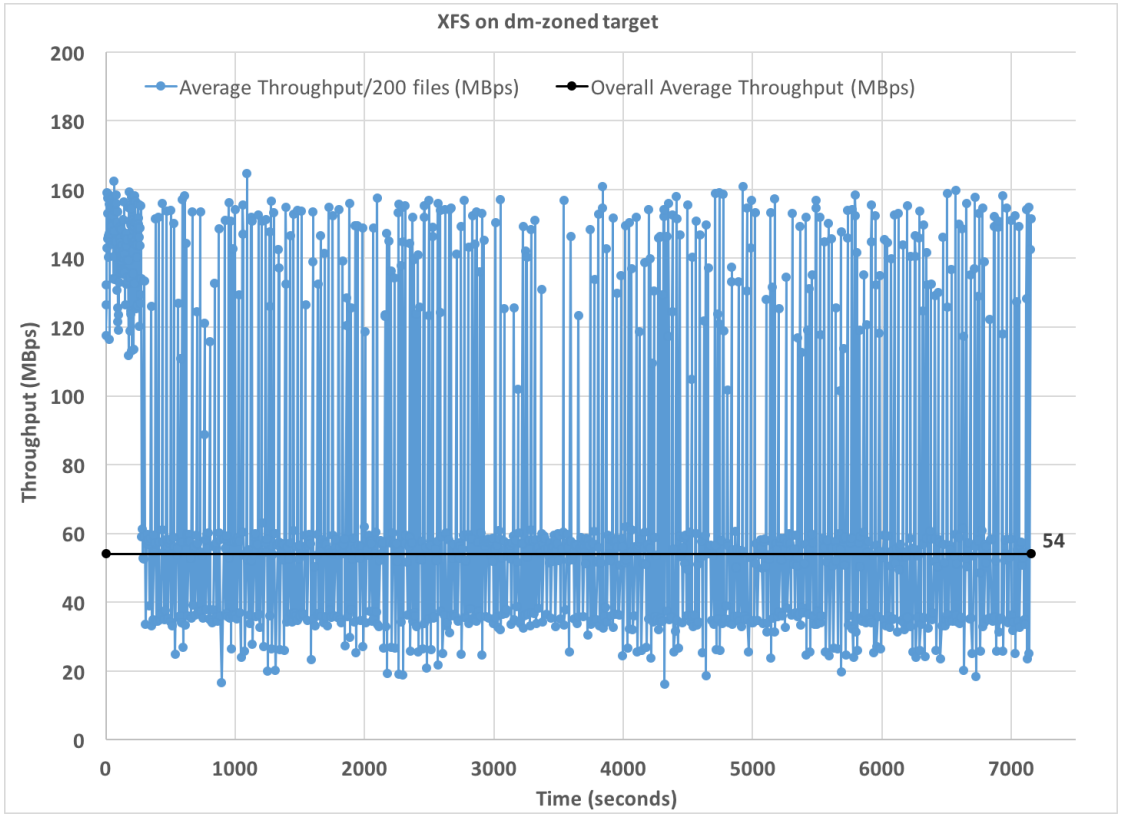


Sustained Write Workload

XFS

- Initial high performance on dm-zoned better (longer) than ext4
  - But overall no significant improvements, similar average

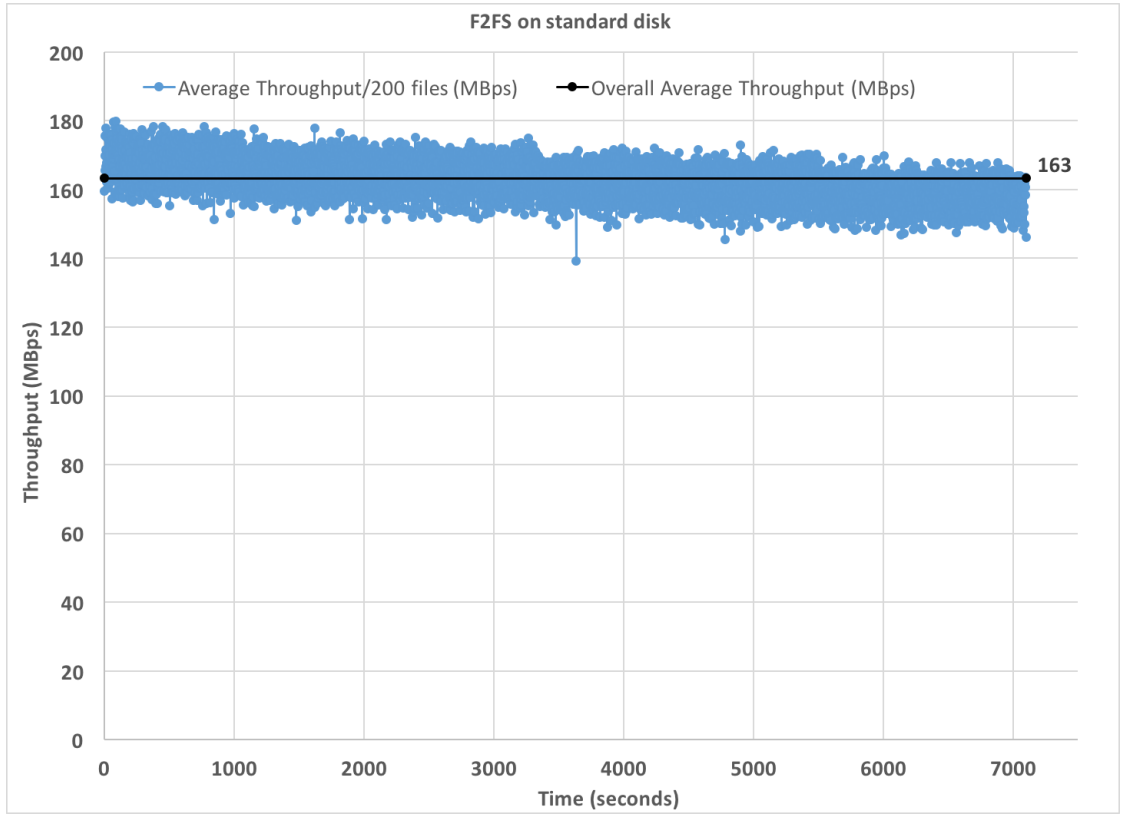


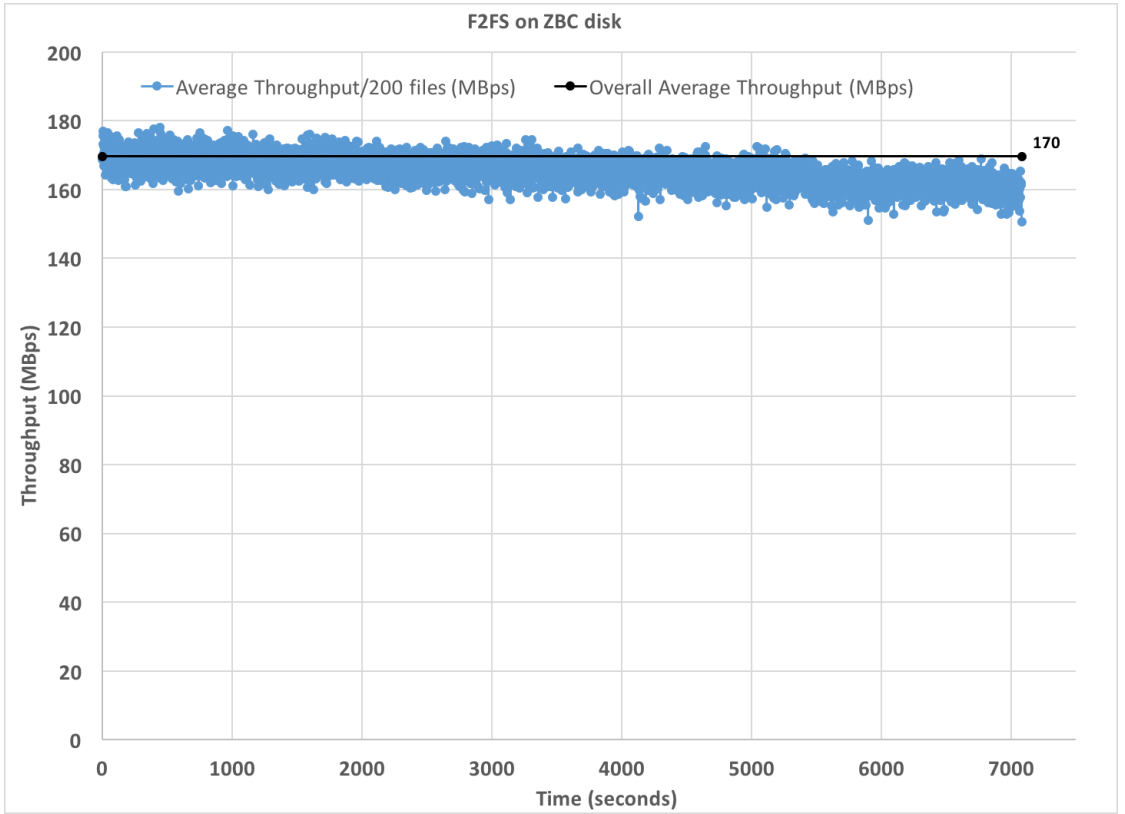


Sustained Write Workload

f2fs

- SMR does not matter !
  - Same performance

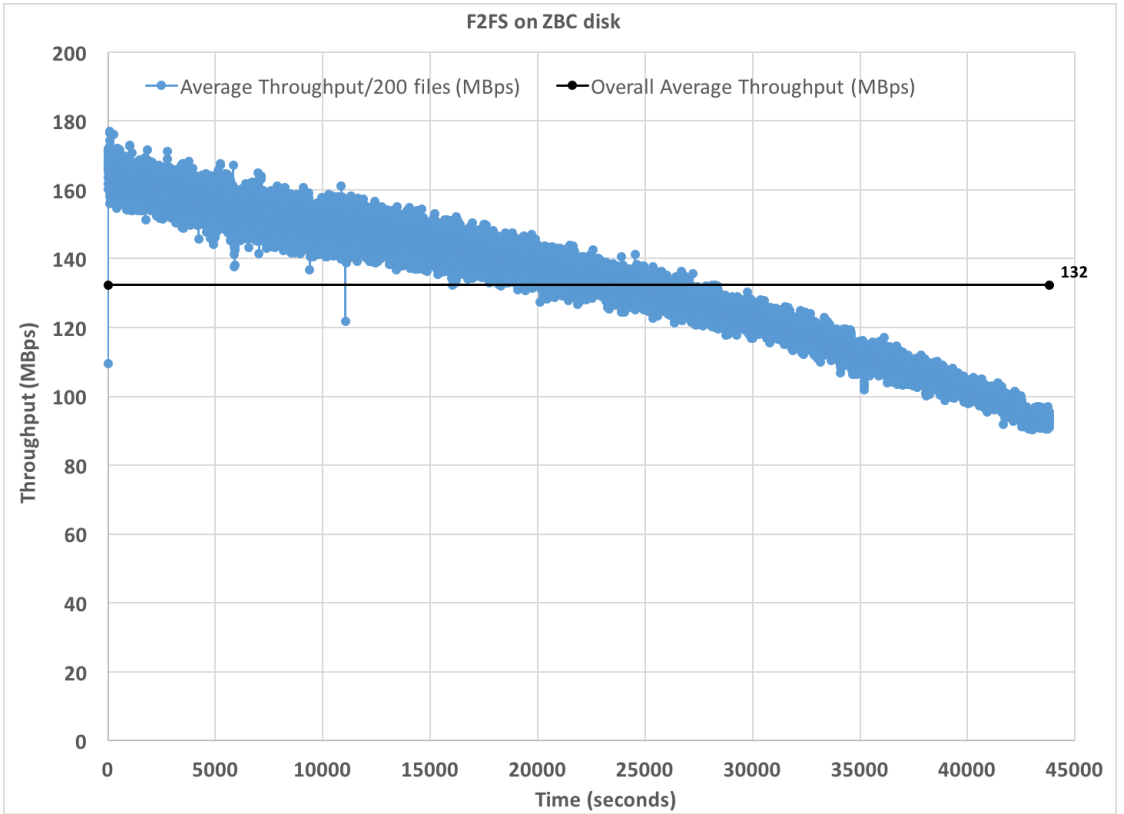




Sustained Write Workload

f2fs

- Performance is sustained until disk full
  - Only OD-ID performance change

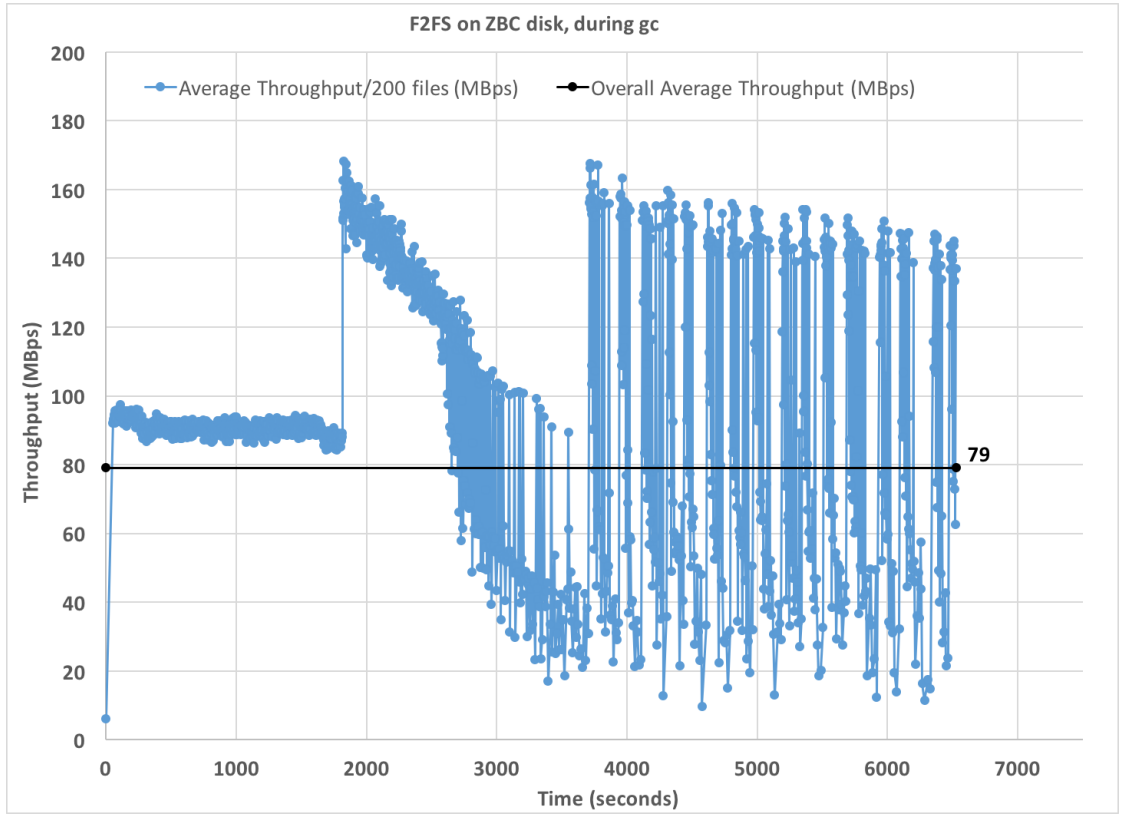


Sustained Write Workload

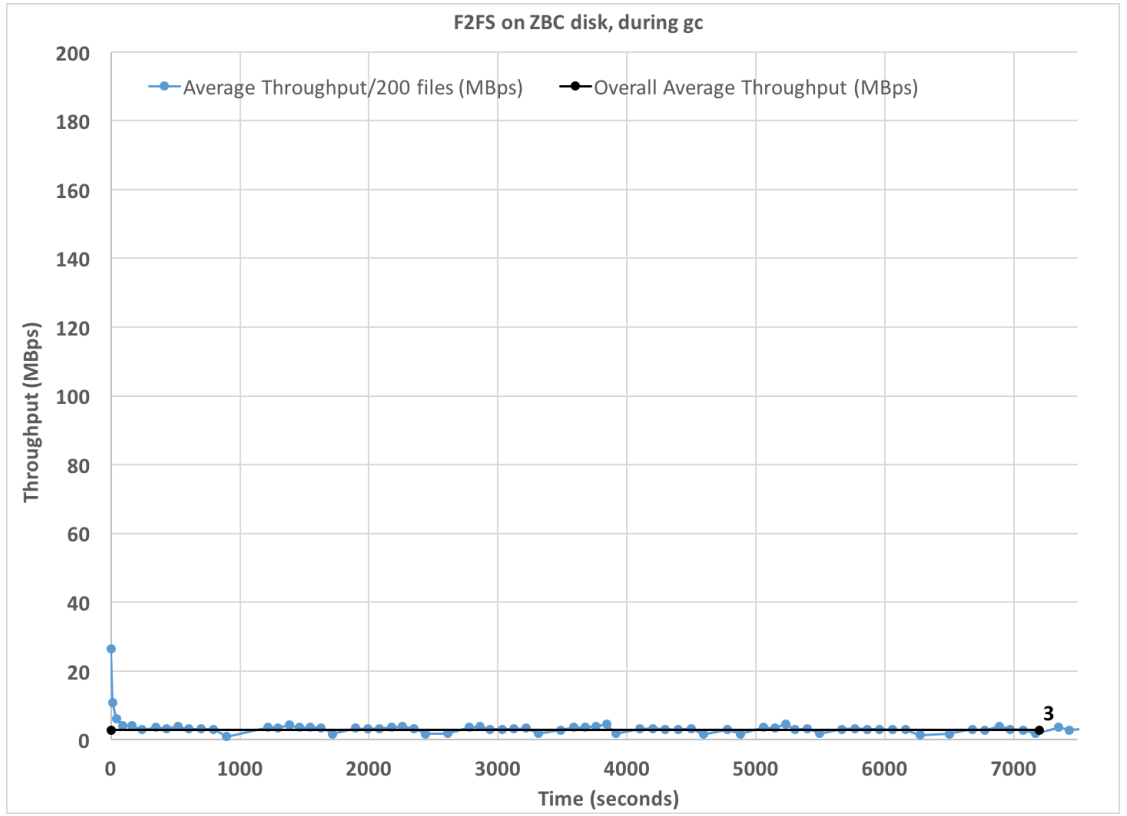
f2fs under GC

- GC Can be very costly
  - dm-zoned doing better

Write after random directory deletion



Write after random file deletion



Conclusions

SMR constraints can be dealt with

- ZBC kernel support is simple
  - No "intelligence" to sequentialize writes
  - But flexible for file systems and device mappers
- File system approach is better
  - More information to work with compared to device mappers
  - Better performance
    - E.g. F2FS
  - More tuning necessary
- More work coming online in the next few month
  - BtrFS, XFS
  - dm-zoned (with compression ?)