**CSC494.A2**

## 1.1 ext4 file system

- *Q1: List the 3 different logging modes in ext4. How are they different from each other? (Hint: look at the "data" option in mount command for ext4.)* (https://bit.ly/2F4aFaJ)

| | | |
|---|---|---|
| data=journal | | All data are committed **into the journal prior to being written into the main file system**. Enabling this mode will disable delayed allocation and O_DIRECT support. |
| data=ordered | (*) | All data are forced directly **out to the main file system prior to its metadata** being committed to the journal. |
| data=writeback | | **Data ordering is not preserved**, data **may be written into the main file system after its metadata** has been committed to the journal. |

- *Q2: Which mode is the most efficient (high-performance) mode? Why? Is it reliable?* (https://ibm.co/2Dl6l5d)
  Efficiency: data=writeback > data=ordered > data=journal
  data=writeback mode is the most efficient mode under most conditions, since it doesn't do any form of data journaling at all and there is no data ordering preserved. But it is **not reliable**, since this could allow recently modified files to become corrupted in the event of an unexpected reboot.

- *Q3: Which mode is the most reliable mode? Why? Is it efficient?* (https://ibm.co/2Dl6l5d)
  data=journal mode is the most reliable mode. Since all new data is written to the journal first, and then to its final location. In the event of a crash, the journal can be replayed, bringing both data and metadata into a consistent state. It is **not efficient**, since data gets written to disk twice rather than once.

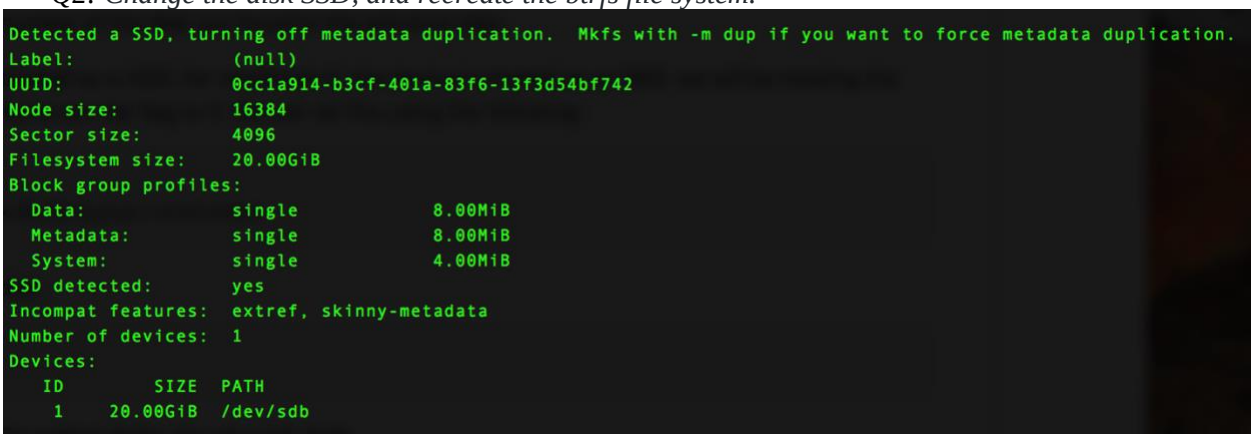
## 1.2 Btrfs file system

- *Q1: Create a btrfs File system on your workload device. Read the log that is created. In particular, look at "Metadata" parameter in "Block group Profiles".*

```
root@yuhanshao-VirtualBox:/home/yuhanshao# mkfs.btrfs -f /dev/sdb
btrfs-progs v4.15.1
See http://btrfs.wiki.kernel.org for more information.

Label:              (null)
UUID:               8799fe6b-d0f8-4b73-a6ea-54c89e15afa6
Node size:          16384
Sector size:        4096
Filesystem size:    20.00GiB
Block group profiles:
  Data:             single            8.00MiB
  Metadata:         DUP               1.00GiB
  System:           DUP               8.00MiB
SSD detected:       no
Incompat features:  extref, skinny-metadata
Number of devices:  1
Devices:
   ID        SIZE  PATH
    1    20.00GiB  /dev/sdb
```

- *Q2: Change the disk SSD, and recreate the btrfs file system.*



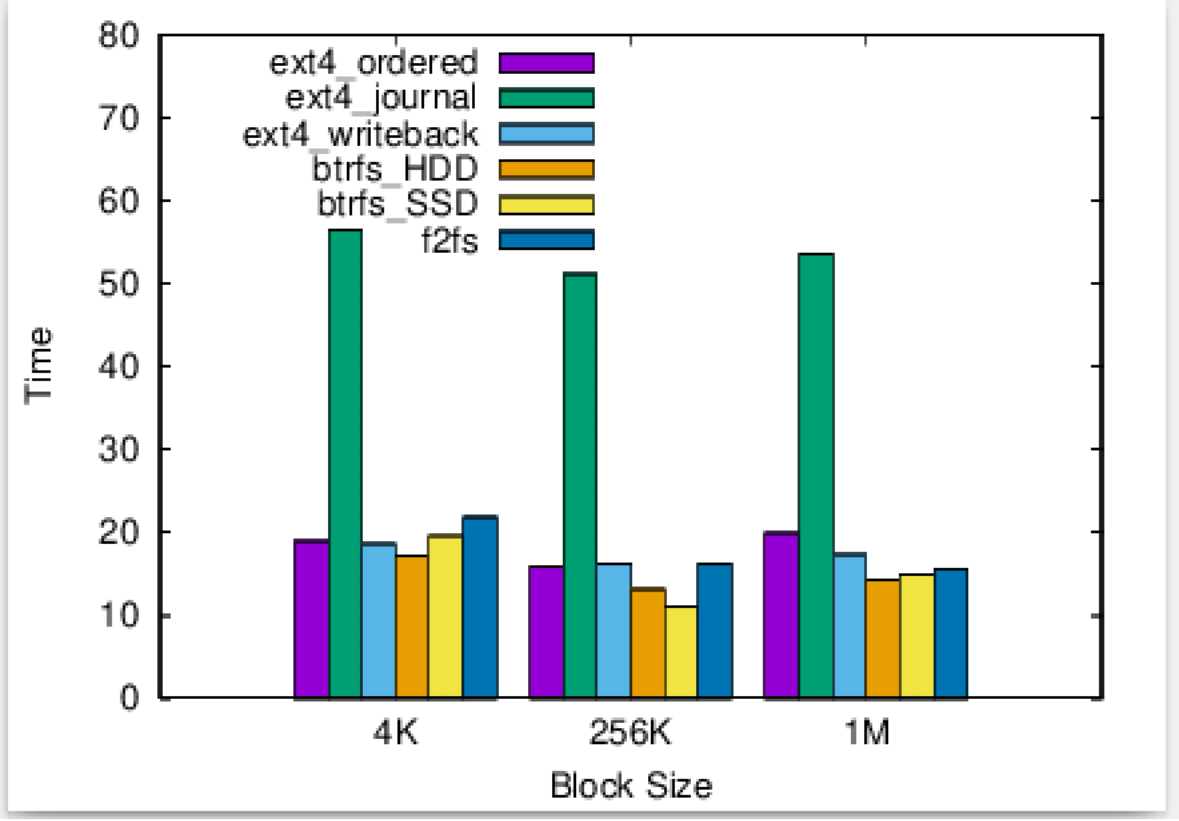

- *Q3: look at the "Metadata" parameter in "Block Group Profiles". Do you see a difference? What could be the reason for this difference between SSD and HDD mode? If you don't see a difference, please email TA/mailing list.*
  Yes, the "Metadata" parameter for HDD is larger than the "Metadata" parameter for SSD. The reason is probably that HDD need extra space for the metadata, but the metadata for SSD is already there, and we do not need to allocate extra space for the metadata.

■ **1.3 F2FS file system** (https://bit.ly/2yRWuQs)

- *Q1: Mention at least 3 reasons (and elaborate) why F2FS is better than ext4 and Btrfs.*
  1. Multi-head logging (cleaning cost reduction)
  2. Adaptive logging (graceful performance degradation in aged condition)
  3. Roll-forward recovery (fsync acceleration)
- *Q2: What type of workloads do you think would lead to performance drop in F2FS?* (https://bit.ly/2QiRona)
  "The file system builds on append-only logging and its key design decisions were made with the characteristics of flash storage in mind." Therefore, the performance is better on the sequential access workloads and the performance drop when the workloads is random.

■ **2.1 dd**

- *Q1: Use the dd command to write data to disk. fill the entire disk (do not use the count parameter of dd, it would continue writing until there is no more space left on device). Vary block size (bs) parameter and plot the amount of time it takes for 3 different block sizes - 4K, 256K, 1M. Plot graph showing the time taken to fill the disk v/s block size for all 6 file system configurations.*




- *Q2: Which block size takes least time to fill the disk across all 6 configurations (3 - ext4, 2 - btrfs and default f2fs mode). Plot data to justify your answer.*

  [redacted] 4K > 1M > 256K
- *Q3: Which file system mode of operation (for ext4, btrfs) is the fastest? Plot data to justify your answer.*

  For ext4: for block size 4K and 1M writeback is faster, for block size 256K ordered is slight faster
  For btrfs: for block size 4K and 1M HDD is faster, for block size 256K SDD is slight faster
- *Q4: Which file system is the fastest? Plot data to justify your answer.*

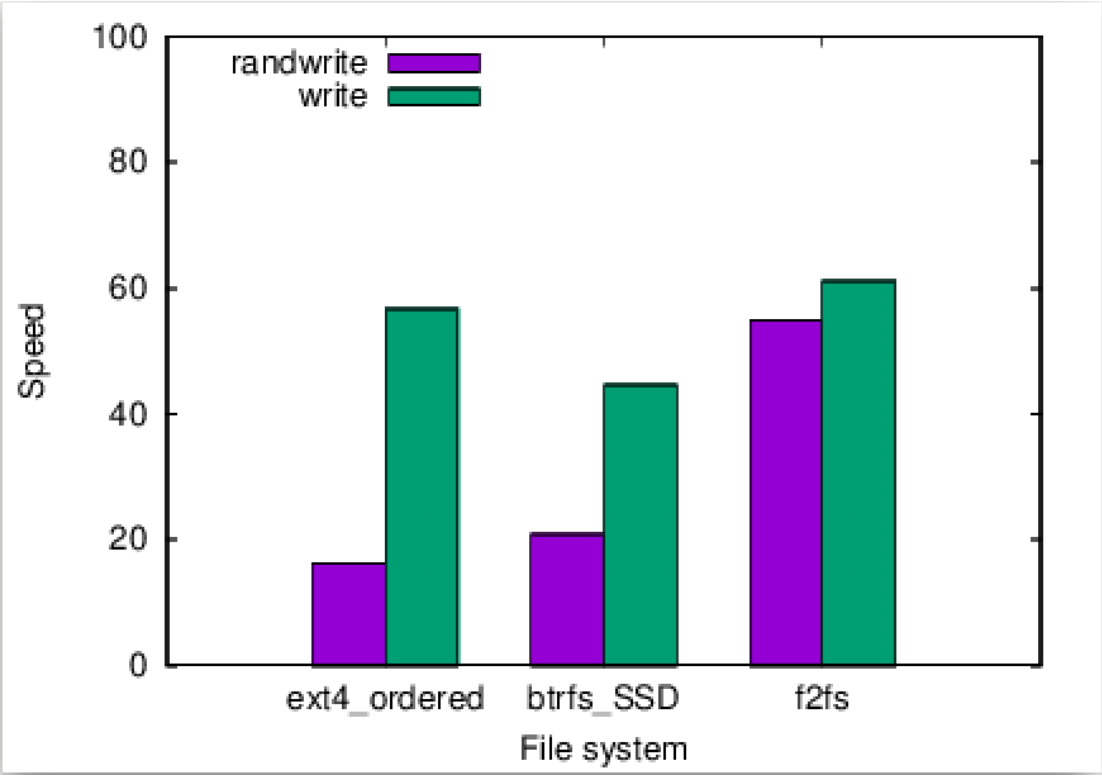
  **btrfs**

■ **2.2 fio (https://linux.die.net/man/1/fio)**

- Q1: run fio on your device with 4 threads for 5 minutes. Show the effect of varying the following parameters on ext4(data=ordered mode), btrfs(SSD mode) and default F2FS mode:

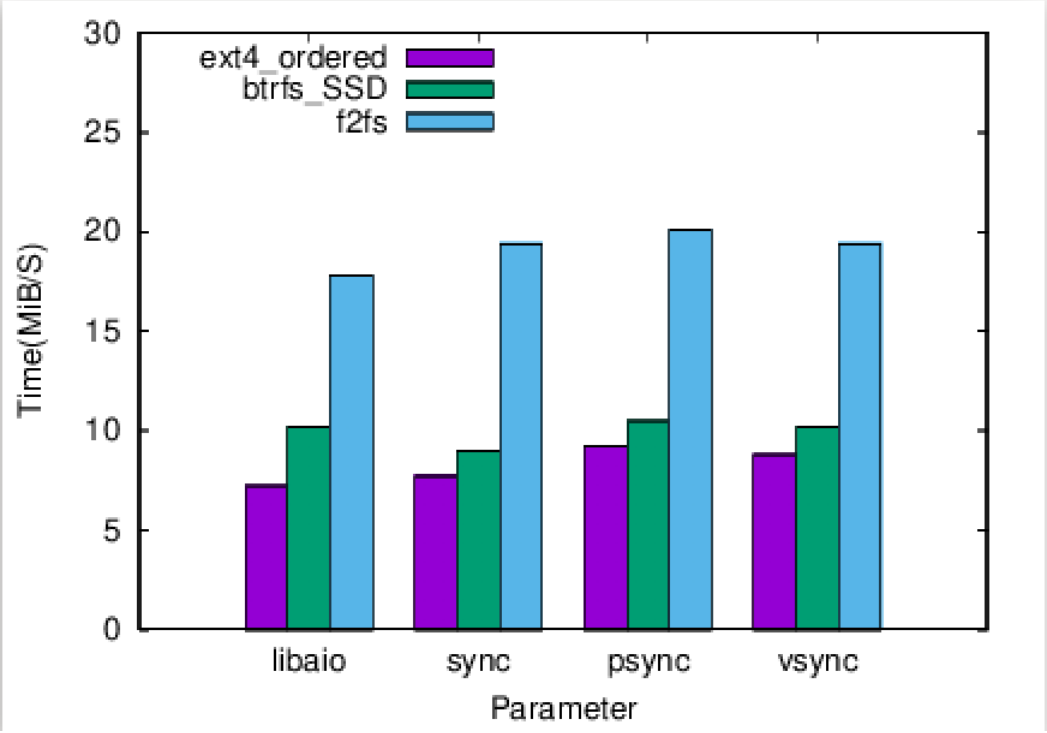
→ Since vary of the 'readwrite' with different 'io-throughput' (randwrite with 2048MiB; write with 2048MiB; randrw with 1024 MiB), draw the graph of this parameter with 'randwrite' and 'write'.

→ **write:**



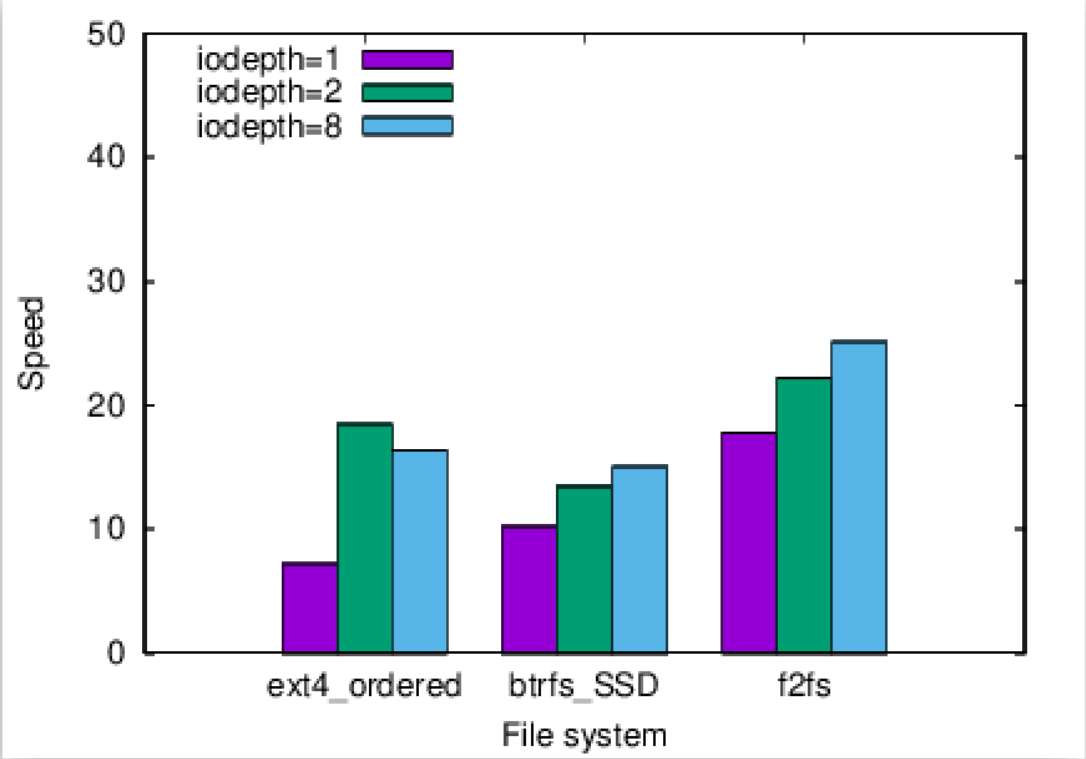

**Observation:** 'rw=write' is sequential write, 'rw=randwrite' is random write. For both write, f2fs is the fastest.

→ **ioengine:**



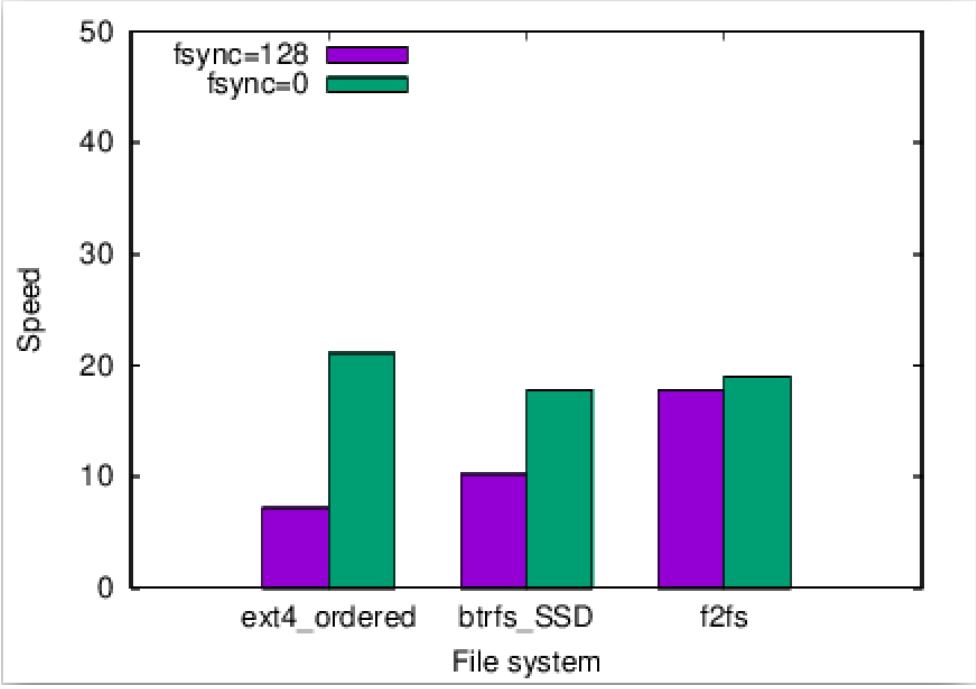

**Observation:** No matter how to vary the parameter ioengine, f2fs is the fastest.

→ **iodepth:**



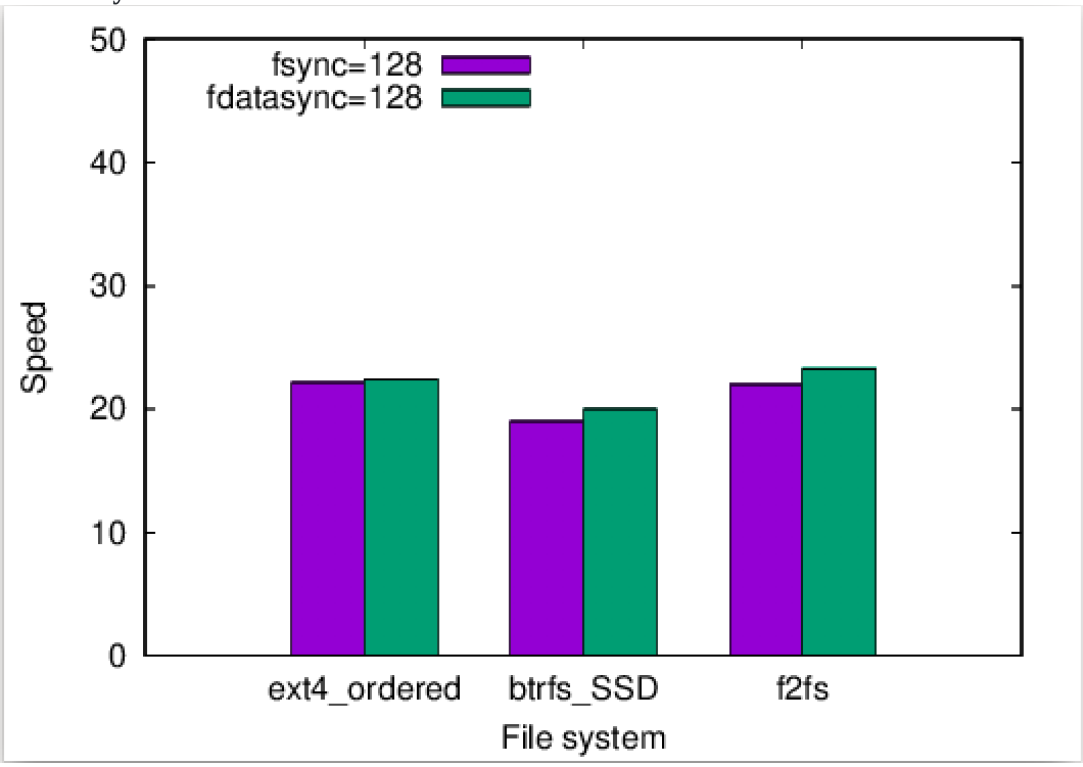

**Observation:** For both iodepth=1 and iodepth=2, f2fs is the fastest. When iodepth=1, btfrs(SSD) is better than ext4, but when iodepth=2, ext4 is better than btrfs(SSD).

→ **fsync:**




**Observation:** For both fsync=128 and fsync=0, f2fs is the fastest. When fsync=128, btfrs(SSD) is better than ext4, but when fsync=0, ext4 is better than btrfs(SSD).

→ **fdatasync:**




**Observation:** in man page '**fdatasync**=*int*, Like **fsync**, but uses **fdatasync**(2) instead to only sync the data parts of the file. Default: 0.' When 'fsync=128' btrfs (SSD) is slight faster than ext4, but when 'fdatasync=128', ext4 (ordered) is much faster than btrf (SSD).

**Default command: (used to compare)**

sudo fio --*name*=randrw --**ioengine**=libaio --**iodepth**=1 --**rw**=**randrw** --bs=4k --direct=1 --**fsync**=128 --size=512M --numjobs=4 --runtime=300 --directory=/mnt

| | | **ext4 (ordered)** | **btrfs (SSD)** | **f2fs** |
|---|---|---|---|---|
| **readwrite** | **randwrite** io=2048MiB (2147MB) | bw=16.2MiB/s (16.9MB/s) | bw=20.9MiB/s (21.9MB/s) | bw=54.9MiB/s (57.6MB/s), |
| | **write** io=2048MiB (2147MB) | bw=56.8MiB/s (59.5MB/s) | bw=44.7MiB/s (46.9MB/s) | bw=61.2MiB/s (64.2MB/s) |
| | **randrw** io=1024MiB (1074MB) | bw=7.240MiB/s (7.413MB/s) | bw=10.2MiB/s (10.7MB/s) | bw=17.8MiB/s (18.7MB/s) |
| **ioengine** (rw=randrw) | **libaio** io=1024MiB (1074MB) | bw=7.240MiB/s (7.413kB/s) | bw=10.2MiB/s (10.7MB/s) | bw=17.8MiB/s (18.7MB/s) |
| | **sync** io=1024MiB (1074MB) | bw=7.738MiB/s (7.924MB/s) | bw=8.956MiB/s (9.171MB/s) | bw=19.4MiB/s (20.4MB/s) |
| | **psync** io=1024MiB (1074MB) | bw=9.206MiB/s (9.427MB/s) | bw=10.5MiB/s (10.0MB/s) | bw=20.1MiB/s (21.0MB/s) |
| | **vsync** io=1024MiB (1074MB) | bw=8.777MiB/s (8.988MB/s) | bw=10.2MiB/s (10.7MB/s) | bw=19.4MiB/s (20.3MB/s) |

| | | | | |
|---|---|---|---|---|
| **iodepth** (rw=randrw, ioengine=libaio) | **iodepth=1** io=1024MiB (1074MB) | bw=7.240MiB/s (7.413MB/s) | bw=10.2MiB/s (10.7MB/s) | bw=17.8MiB/s (18.7MB/s) |
| | **iodepth=2** io=1024MiB (1074MB) | bw=18.5MiB/s (19.4MB/s) | bw=13.5MiB/s (14.1MB/s) | bw=22.2MiB/s (23.2MB/s) |
| | **iodepth=8** io=1024MiB (1074MB) | bw=16.4MiB/s (17.2MB/s) | bw=15.1MiB/s (15.9MB/s) | bw=25.1MiB/s (26.3MB/s) |
| **fsync** (rw=randrw, ioengine=libaio, iodepth=1) | **fsync=128** io=1024MiB (1074MB) | bw=7.240MiB/s (7.413MB/s) | bw=10.2MiB/s (10.7MB/s) | bw=17.8MiB/s (18.7MB/s) |
| | **fsync=0** io=1024MiB (1074MB) | bw=21.1MiB/s (22.1MB/s) | bw=17.8MiB/s (18.7MB/s) | bw=19.0MiB/s (20.9MB/s) |
| **fdatasync** (fsync=128 / fdatasync=128) | **fsync=128** io=1024MiB (1074MB) | bw=22.2MiB/s (23.3MB/s) | bw=19.0MiB/s (20.9MB/s) | bw=22.0MiB/s (24.1MB/s) |
| | **Fdatasync=128** io=1024MiB (1074MB) | bw=22.4MiB/s (23.5MB/s) | bw=20.0MiB/s (21.0MB/s) | bw=23.3MiB/s (24.4MB/s) |

■ **BONUS:**

According to the man page, possible parameters effecto run on the file system:

**randrepeat**=*bool*

Seed the random number generator in a predictable way so results are repeatable across runs. Default: true.

**use_os_rand**=*bool*

Fio can either use the random generator supplied by the OS to generator random offsets, or it can use it's own internal generator (based on Tausworthe). Default is to use the internal generator, which is often of better quality and faster. Default: false.

**fallocate**=*str*

Whether pre-allocation is performed when laying down files. Accepted values are:

**none** Do not pre-allocate space.
**posix** Pre-allocate via posix_fallocate().
**keep** Pre-allocate via fallocate() with FALLOC_FL_KEEP_SIZE set.
**0** Backward-compatible alias for 'none'.
**1** Backward-compatible alias for 'posix'.

May not be available on all supported platforms. 'keep' is only available on Linux. If using ZFS\ on Solaris this must be set to 'none' because ZFS doesn't support it. Default: 'posix'.

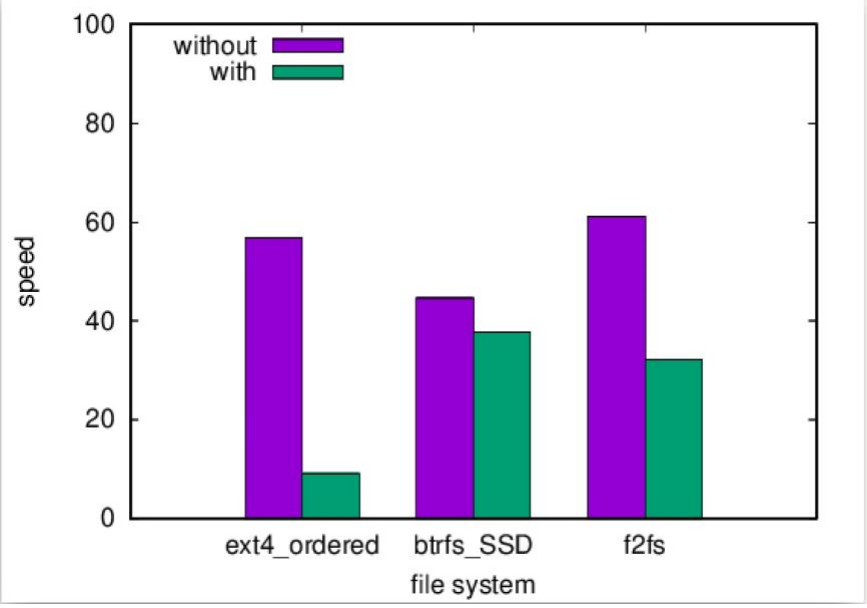
- **2.3 blktrace** (https://bit.ly/2qnHEN8 + 'man blkparse')

- *Q1: identify what each column stands for in the blktrace. Look at column 6. We are only interested in values with "C" explain why. Look at column 7. what do W and WS stand for? explain. Look at column 9. we see writes from not only dd but also by jbd2. what is jbd2?*
  → The 1st column shows the device (major:minor) tuple.
  → The 2nd column shows the CPU number.
  → The 3rd column shows the sequence number.
  → The 4th column is the time stamp.
  → The 5th column is the PID of the process issuing the I/O request.
  → The 6th column shows the event type. 'C'ompleted block traces, 'I'mcompleted block traces and 'D'river ["ACTION IDENTIFIERS" section]. We are interested in 'C' since we are only interested in the blocks that completely traced.
  → The 7th column "W"ritten both asynchronously or "W"ritten "S"ynchronously
  → The 8th column is the **block number** + **the number of blocks requested**.
  → The 9th column [ ] brackets is the process name of the process issuing the request.

- *Q2: you wrote 10 blocks each 4KB in size. the 8th column shows a number + 80. what does the number stand for? What does the 80 stands for? why is it 10 and not 80? (Hint: 1 sector = 512 bytes).*
  The number before '+' in the 8th column is the block number. 80 stands for the number of sectors requested. Need write 10 blocks each 4KB in size, then we need to write 40KB, each sector is 512 bytes, then we need number of sectors: 40KB / 512 bytes = 80.

- Q3: re-run fio on your device for 5 mins, with 2 of the best configurations (**maximum throughput**) for all 3 file system modes that you ran in **2.2.** This time, enable blktrace in live mode. Report the throughput and compare with the previous non-blocktrace fio run in a graph.

| | ext4(ordered) | btrfs(SSD) | f2fs |
|---|---|---|---|
| write without blktrace | 56.8MiB/s | 44.7MiB/s | 61.2MiB/s |
| write with blktrace | 9.109MiB/s | 37.735MiB/s | 32.187MiB/s |



- Q4: report the size of the blktrace file captured, by only looking at "C"ompleted block traces. which are "W"ritten both asynchronously or "W"ritten "S"ynchronously. report your log file size for each of the 6 configurations (3+2+1) runs for Q3.

| | |
|---|---|
| btrfsSSD.txt | 1441444 |
| btrfsHDD.txt | 1292344 |
| f2fs.txt | 1370374 |
| journal.txt | 5312964 |
| ordered.txt | 1419632 |
| writeback.txt | 1419749 |

- Q5: finally run fio for 20 minutes for any fio configuration in default file system mode. (btrfs)

■ Without blktrace:

sudo fio --name=randwrite --ioengine=libaio --iodepth=1 --rw=randwrite --bs=4k --direct=1 --fsync=128 --size=512M --numjobs=4 --runtime=1200 --directory=/mnt

WRITE: bw=81.9MiB/s (85.8MB/s), 20.5MiB/s-22.3MiB/s (21.5MB/s-23.4MB/s), io=2048MiB (2147MB), run=22975-25021msec

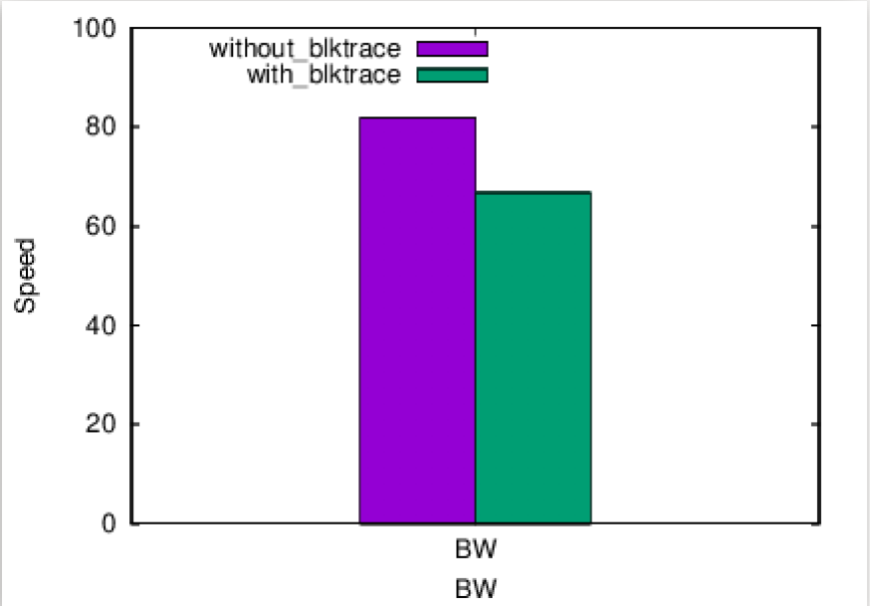
■ With blktrace:

sudo blktrace -d /dev/sdb -w 30 -o - | blkparse -a fs -i –

sudo fio --*name*=write --**ioengine**=libaio --**iodepth**=1 --**rw**=**write** --bs=4k --direct=1 --**fsync**=128 --size=512M --numjobs=4 --runtime=300 --directory=/mnt

WRITE: bw=66.8MiB/s (70.1MB/s), 16.7MiB/s-17.1MiB/s (17.5MB/s-17.0MB/s), io=2048MiB (2147MB), run=29888-30653msec

Block size: 34853231




decreased 18.43%MiB/s