WAL

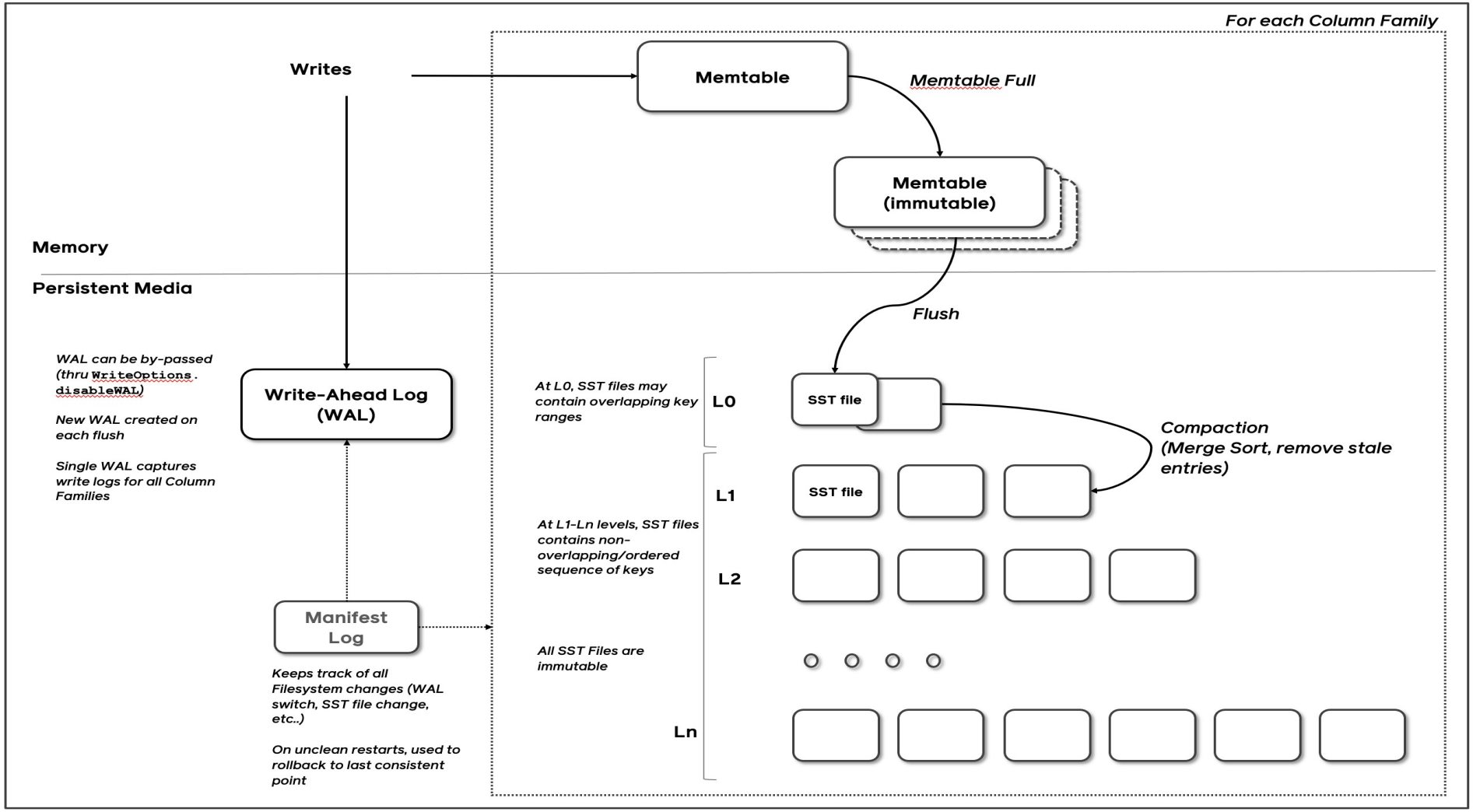
Suhwan Shin, Isu Kim, Seyeon Park

E-Mail: tlstnghks77@dankook.ac.kr

Content

- WAL
- Experiment
  - Disable_wal
  - Max_total_wal_size
  - Manual_wal_flush

WAL



https://github-wiki-see.page/m/facebook/rocksdb/wiki/RocksDB-Overview

Before we start experiment…

- LevelDB has no option of WAL for db_bench
- Need to use RocksDB
- How big is RocksDB?

```
gooday2die@flagship:~/projects/School/2022_0.5/LevelDB/new/rocksdb_release$ cloc . | grep -e "C++" -e "Header" -e "C"
     195 files ignored.
C++                             671          52413          48968         345453
C/C++ Header                    538          21177          42280          90995
CMake                            77            476            328           5207
C                                 4            691            194           3585
gooday2die@flagship:~/projects/School/2022_0.5/LevelDB/new/rocksdb_release$
gooday2die@flagship:~/projects/School/2022_0.5/LevelDB/levelsDB/leveldb_debug$ cloc . | grep -e "C++" -e "Header" -e "C"
     265 files ignored.
C++                             239          12987          13973          63411
C/C++ Header                    127           5309          11754          19115
CMake                           116            775            882           4026
C                                 2            175             73            939
gooday2die@flagship:~/projects/School/2022_0.5/LevelDB/levelsDB/leveldb_debug$
```

Option : Disable_wal

Isu Kim

Disable_wal : Hypothesis

- Disabling WAL will affect db_bench performance
- WAL will use IO, thus will be slower if enabled
- However, SAF and WAF will be the same

=> Disabling WAL will have better results in terms of throughput and latency

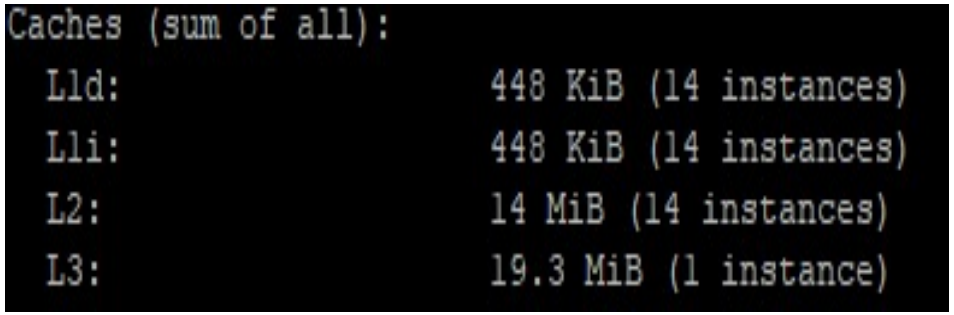
Disable_wal : Design

- Independent Variable: --disable_wal (true or false)
- Dependent Variable: SAF, WAF, Latency, Throughput
- Controlled Variable:
  1. --benchmarks="fillseq,stats,levelstats,compact,stats,levelstats"
  2. --num=10000000 (default)
  3. Compiled option :

```
$ cmake -DCMAKE_BUILD_TYPE=Release -DFAIL_ON_WARNINGS=OFF -DWITH_SNAPPY=ON.. & cmake --build . -j 20
```

Disable_wal : Environment

- Server Spec
  - OS: Ubuntu 22.04 LTS (Not VM)
  - CPU: Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz
  - SSD: Samsung SSD 860 2TB



- Test Environment
  - Python (IPython Script) using subprocess
  - Automatically runs fillseq, fillrandom, readrandom for designated times
  - Parses output from db_bench and generates a simple pandas.DataFrame
  - Does not use thread since it might affect output results.
  - All other processes were turned off as much as possible

Disable_wal : Result

- Results of fillseq, fillrandom, readrandom for 10 times each.

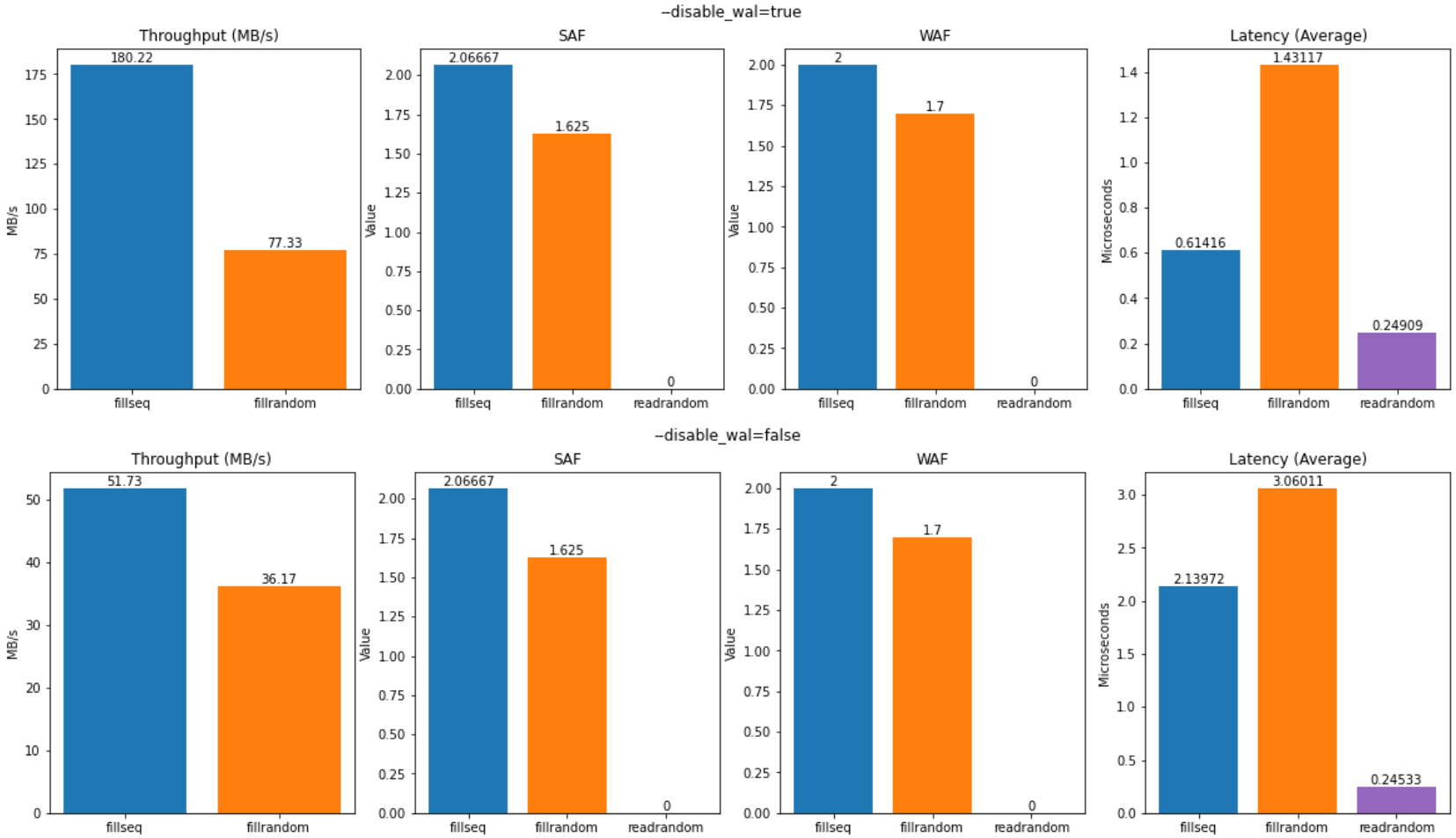
```
In [12]: result
```

Out [12]:

	Type	Throughput(MB/s)	Throughput(Micros/op)	Latency (Average)	Latency (Standard Deviation)	Latency (Min)	Latency (Max)	Latency (Median)	WAF	SAF
0	fillrandom	77.3	1.431	1.4313	8.75	0.0	8713.0	0.7977	1.7	1.625000
1	fillseq	179.1	0.618	0.6178	8.78	0.0	8756.0	0.5187	2.0	2.066667
2	readrandom	-1.0	0.264	0.2643	0.55	0.0	266.0	0.5002	0.0	0.000000
3	fillrandom	80.5	1.374	1.3742	8.75	0.0	8723.0	0.7465	1.7	1.625000
4	fillseq	178.4	0.620	0.6201	8.58	0.0	8554.0	0.5181	2.0	2.066667
5	readrandom	-1.0	0.243	0.2430	0.54	0.0	281.0	0.5001	0.0	0.000000
6	fillrandom	78.0	1.418	1.4183	8.88	0.0	8853.0	0.7859	1.7	1.625000

Disable_wal : Result

- Average of 10 db_bench results



Disable_wal : Conclusion & Future Study

- Enabling WAL will use IO, thus in terms of throughput and latency, it is slower than disabling it.

- Future experiment ideas:
  - Use uftrace to analyze more about internal operations.
  - Disable WAL and shutdown abruptly.
  - Enable WAL and shutdown abruptly.
  - Find reasons for why SAF and WAF stays the same.
  - Test with wal_bytes_per_sync.

Option : Max_total_wal_size

Seyeon Park

Max_total_wal_size : Hypothesis

- If there is no limit about the wal size, it cause slow deletion of wal in ac cor-dance with no flush for a while
- In case of setting option ”max_total_wal_size”, if data satisfy the size, it’s going to be triggered
- Then, will the performance deteriorate because the smaller the wal, the more often it flushes?

Max_total_wal_size : Design

- Independent Variable: --max_total_wal_size = [int value]
- Dependent Variable: SAF, WAF, Latency, Throughput

- ./db_bench --benchmarks="fillseq,stats,levelstats,compact,stats,levelstats, fillrandom,stats,levelstats,compact,stats,levelstats" --max_total_wal_size= [0,1,10,100,1000,10000,100000,1000000,10000000, 100000000]

- --num=10000000 (default)

Max_total_wal_size : Environment

- Server Spec
  - OS: macOS Monterey
  - Processor: 2.3 GHz 8코어 Intel Core i9
  - SSD: APPLE SSD AP1024N 1TB

Max_total_wal_size : Result

- I'd tried num, key_size, WALRecoveryMode…
- I think there is no significant result..

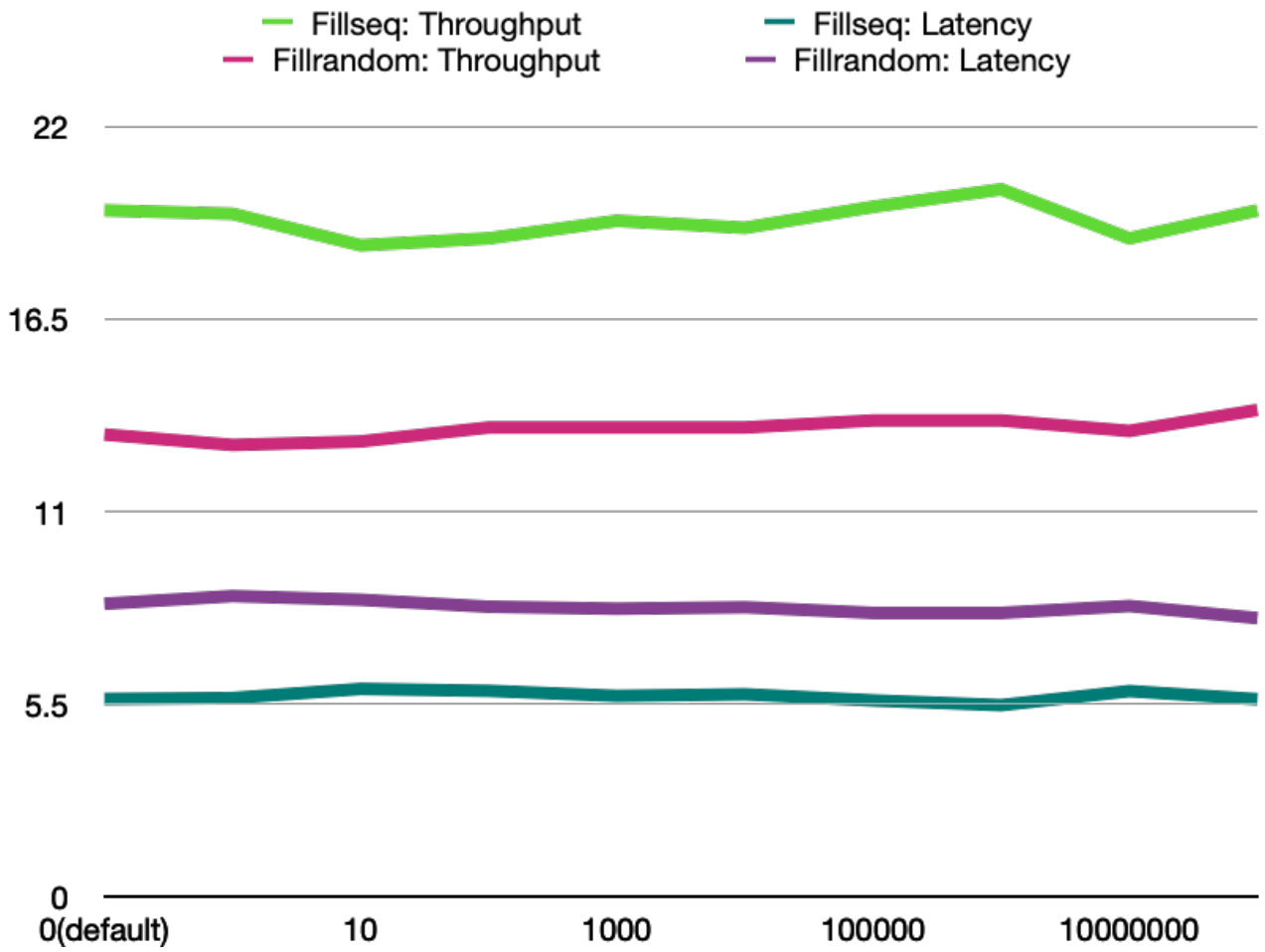
fillseq

WAL_size	0(default)	1	10	100	1000	10000	100000	1000000	10000000	100000000
Throughput	19.6	19.5	18.6	18.8	19.3	19.1	19.7	20.2	18.8	19.6
Latency	5.648	5.676	5.937	5.881	5.736	5.783	5.607	5.472	5.874	5.642

fillrandom

WALsize	0(default)	1	10	100	1000	10000	100000	1000000	10000000	100000000
Throughput	13.2	12.9	13	13.4	13.4	13.4	13.6	13.6	13.3	13.9
Latency	8.366	8.586	8.480	8.282	8.225	8.267	8.105	8.105	8.303	7.957

Max_total_wal_size : Result



Max_total_wal_size : Conclusion

- In fact, this option is valid when the value is 0 (default)
- [sum of all write_buffer_size * max_write_buffer_number] * 4

- For example, with 15 column families,
- write_buffer_size = 128 MB
- max_write_buffer_number = 6
- max_total_wal_size
  - [15 * 128MB * 6] * 4 = 45GB

```
// Once write-ahead logs exceed this size, we will start forcing the flush of
// column families whose memtables are backed by the oldest live WAL file
// (i.e. the ones that are causing all the space amplification). If set to 0
// (default), we will dynamically choose the WAL size limit to be
// [sum of all write_buffer_size * max_write_buffer_number] * 4
//
// For example, with 15 column families, each with
// write_buffer_size = 128 MB
// max_write_buffer_number = 6
// max_total_wal_size will be calculated to be [15 * 128MB * 6] * 4 = 45GB
//
// The RocksDB wiki has some discussion about how the WAL interacts
// with memtables and flushing of column families.
// https://github.com/facebook/rocksdb/wiki/Column-Families
//
// This option takes effect only when there are more than one column
// family as otherwise the wal size is dictated by the write_buffer_size.
//
// Default: 0
//
// Dynamically changeable through SetDBOptions() API.
uint64_t max_total_wal_size = 0;
```

Max_total_wal_size : Conclusion

- And It takes effect only when there are more than two column family.

- Therefore, have to experiment with more than two column family again.

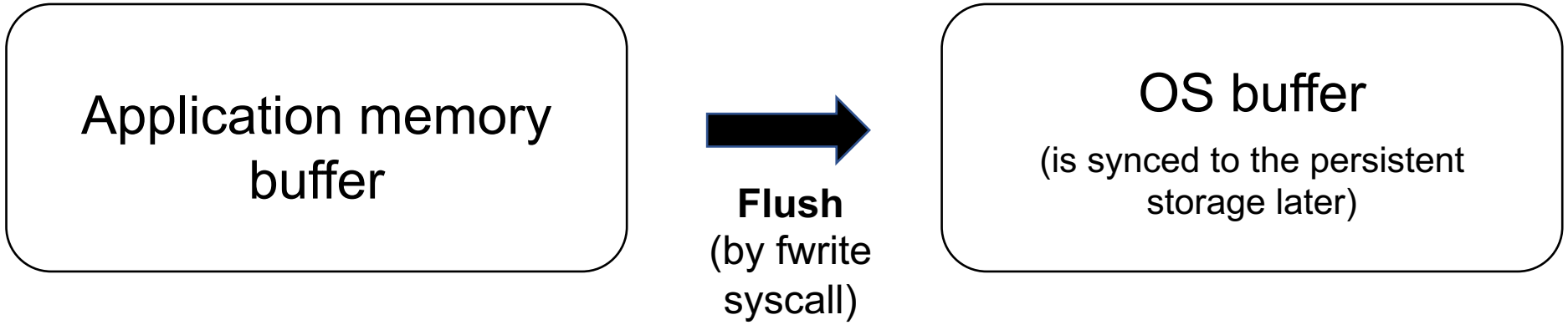
Option : Manual_wal_flush

Suhwan Shin

WAL_flush

- Q: When will the WAL be written?
- A: DB::put or DB::write.

- WAL is first written to the application memory buffer.
- And flushes the WAL from application memory to OS buffer.



Manual_wal_flush : Hypothesis

- Many flush = Overhead ⬆

  => Affect the performance

- Test – Option: manual_wal_flush
  - Manual_wal_flush can be changed flush from automatically to manually after an explicit call to ::FlushWAL.

➢ Tradeoff between reliability and write latency.

Manual_wal_flush : Design

- Independent Variable: --manual_wal_flush (true or false)
- Dependent Variable: Latency, Throughput
- Controlled Variable : Sync mode (Sync or Async)
  - --sync=false(default) : Async
  - --sync=true : Sync

Manual_wal_flush : Environment

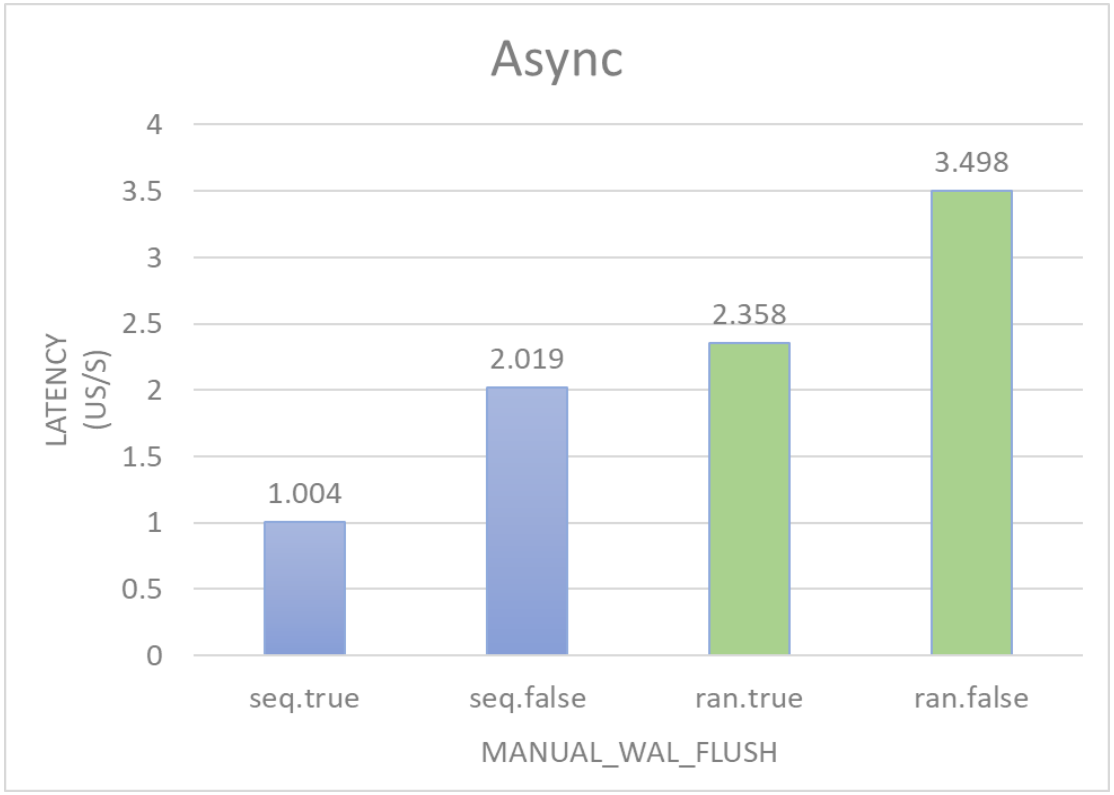
```
RocksDB:    version 7.6.0
Date:       Mon Jul 25 15:57:36 2022
CPU:        16 * Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz
CPUCache:   16384 KB
Keys:       16 bytes each (+ 0 bytes user-defined timestamp)
Values:     100 bytes each (50 bytes after compression)
Entries:    1000000
Prefix:    0 bytes
Keys per prefix:    0
RawSize:    110.6 MB (estimated)
FileSize:   62.9 MB (estimated)
Write rate: 0 bytes/second
Read rate: 0 ops/second
Compression: Snappy
Compression sampling rate: 0
Memtablerep: SkipListFactory
Perf Level: 1
```

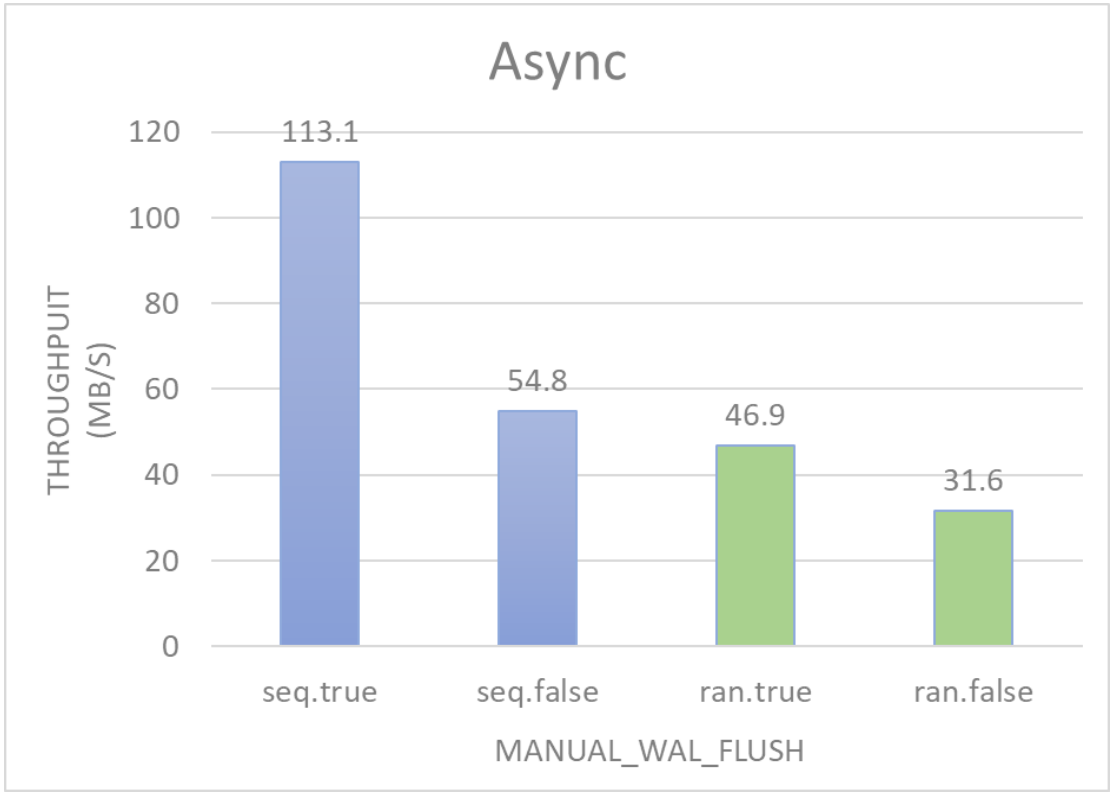
Manual_wal_flush : Result

	Async				Sync			
Benchmark /Manual	Seq/True	Seq/False	Ran/True	Ran/False	Seq/True	Seq/False	Ran/True	Ran/False
Latency (micros/op)	1.004	2.019	2.358	3.498	3029	3041	3047	3035
Throughput (MB/s)	113.1	54.8	46.9	31.6	0	0	0	0

Manual_wal_flush : Result

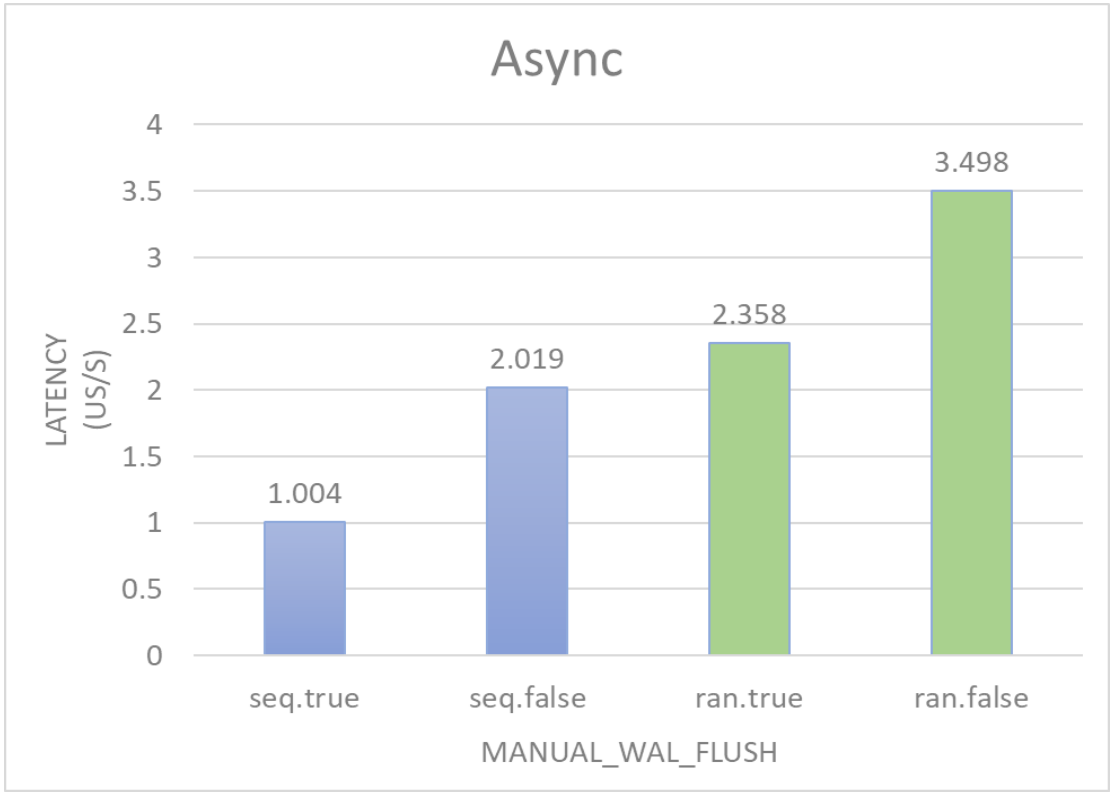
- Async (Latency / Throughput)
  - manual_wal_flush (True/False)
  - fillseq / fillrandom

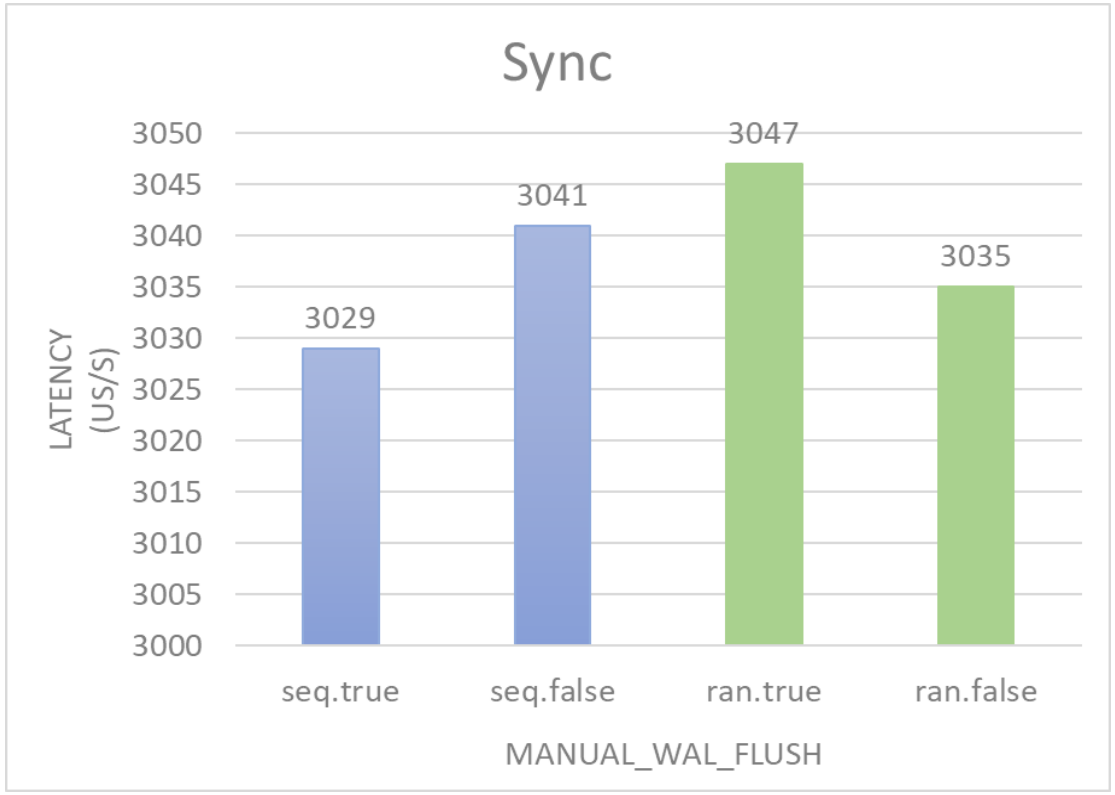




Manual_wal_flush : Result

- Sync (Latency)
  - manual_wal_flush (True/False)
  - fillseq / fillrandom





Manual_wal_flush : Conclusion

- Async mode

➢ An ideal result :

  - fillseq > fillrandom
  - Manual > Automatic

  \+ new fact : Benchmark affects more than Option:manual_wal_flush

- Sync mode

➢ An unideal result :

  - (fillseq > fillrandom) & (fillseq < fillrandom)
  - (Manual > Automatic) & (Manual < Automatic)

Manual_wal_flush : Future Study

- Why Sync's result is informal?

- Manifest

Question