IO-uring speed the RocksDB & TiKV

Why did the **code** always 

Git Repo: https://github.com/PingCAP-Hackthon2019-Team17

Overview

- Background
  - libaio **VS** liburing
- What have we done
  - Case#1: What can our TiKV benefit from io_uring ?
  - Case#2: RocksDB can benefit more from io_uring
  - Case#3: Rewrite the RocksDB compaction by using io_uring
- Future work

IO API history in Linux

- ➔ read(2) / write(2)
- ➔ pread(2) / pwrite(2) offset
- ➔ preadv(2) / pwritev(2) vector-based
- ➔ preadv2(2) / pwritev2(2) modifier flags
- ➔ aio_read(3) / aio_write(3) limited async IO interfaces
- ➔ io-uring since Linux Kernel 5.1

libaio **vs** liburing

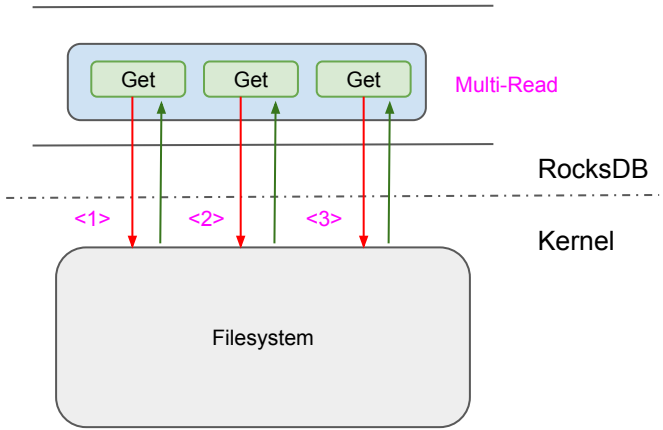
- libaio
  - limitation: only supports async IO for O_DIRECT (or un-buffered) accesses
  - Some internal implementations is still blocking ?
    - meta-data perform blocking IO
    - block waiting for the available request slots in storage device if no available now.
  - Overhead: need extra bytes copy
    - IO submission need 64+8 bytes
    - IO completion need 32 bytes.

libaio **vs** liburing

- libaio
  - limitation: only supports async IO for O_DIRECT (or un-buffered) accesses
  - Some internal implementations is still blocking ?
    - meta-data perform blocking IO
    - block waiting for the available request slots in storage device if no available now.
  - Overhead: need extra bytes copy
    - IO submission need 64+8 bytes
    - IO completion need 32 bytes.
- liburing
  - **Fixed all above problem**
  - Better performance & scalability

Case#1: What we TiKV can benefit from io_uring ?

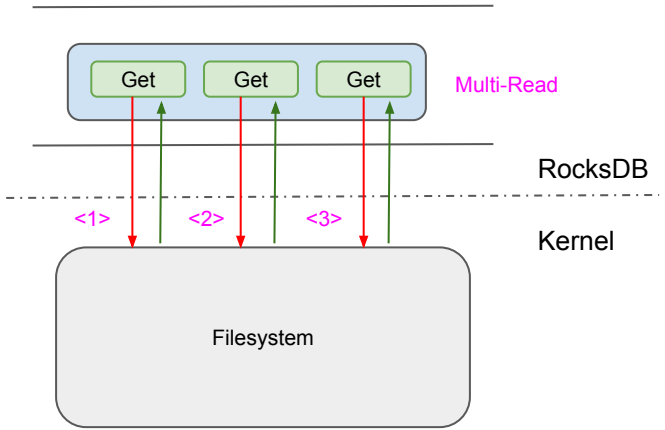
- Facebook rewrite the MultiRead by using io_uring
  - https://github.com/facebook/rocksdb/pull/5881/files



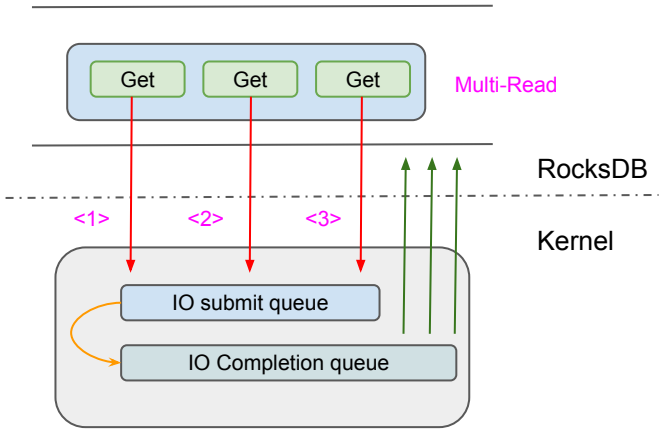
Before io_uring

Case#1: What we TiKV can benefit from io_uring ?

- Facebook rewrite the MultiRead by using io_uring
  - https://github.com/facebook/rocksdb/pull/5881/files

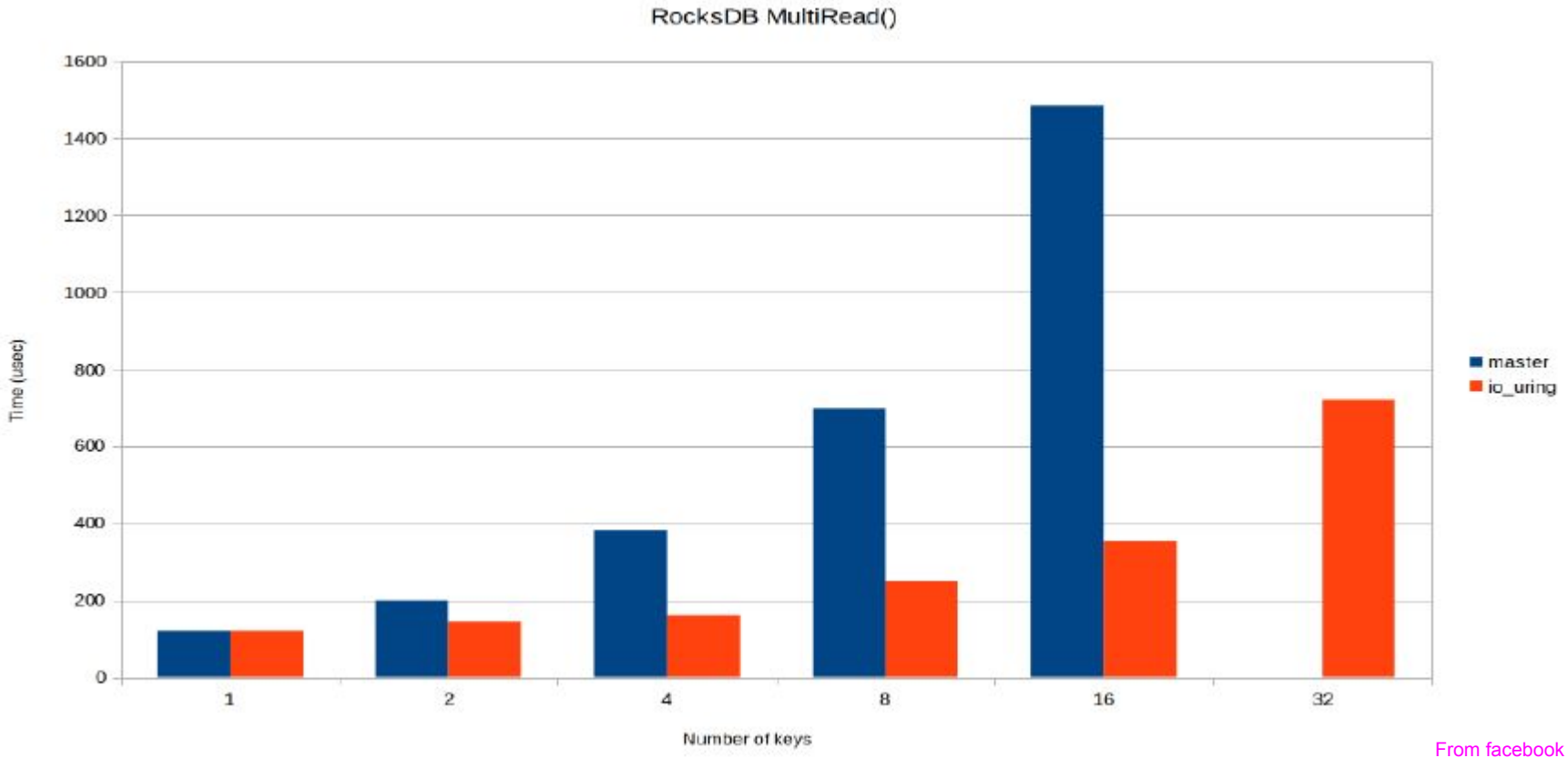


Before io_uring



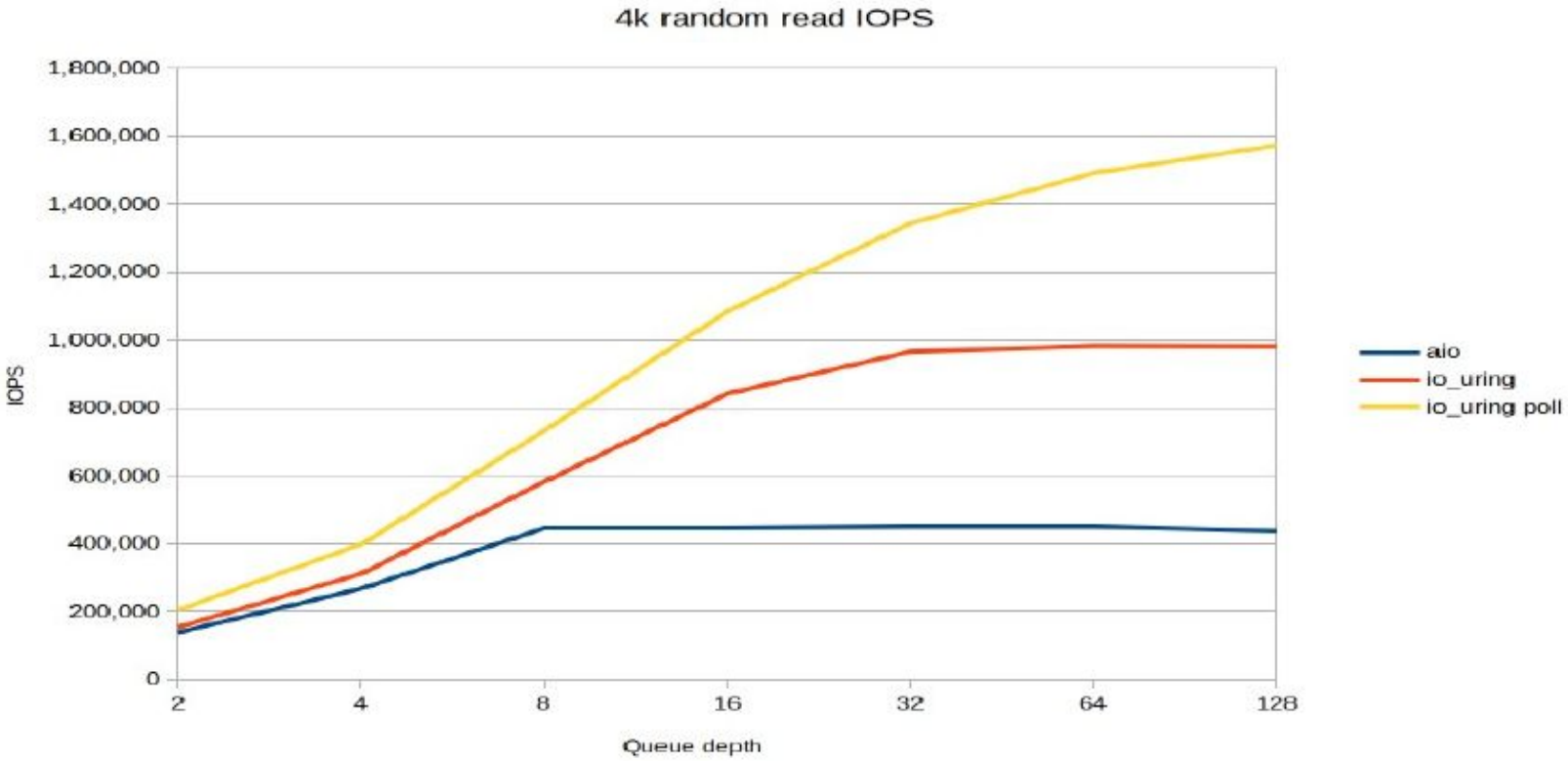
After io_uring

RocksDB: Multi-Reads optimized by io_uring (1)



From facebook team

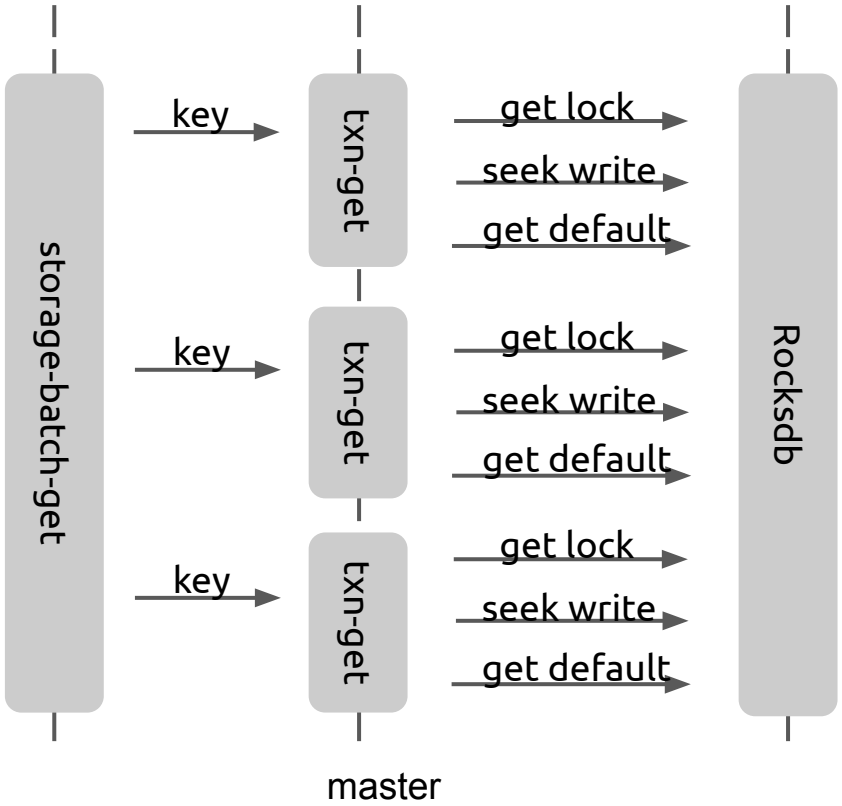
RocksDB: Multi-Reads optimized by io_uring (2)

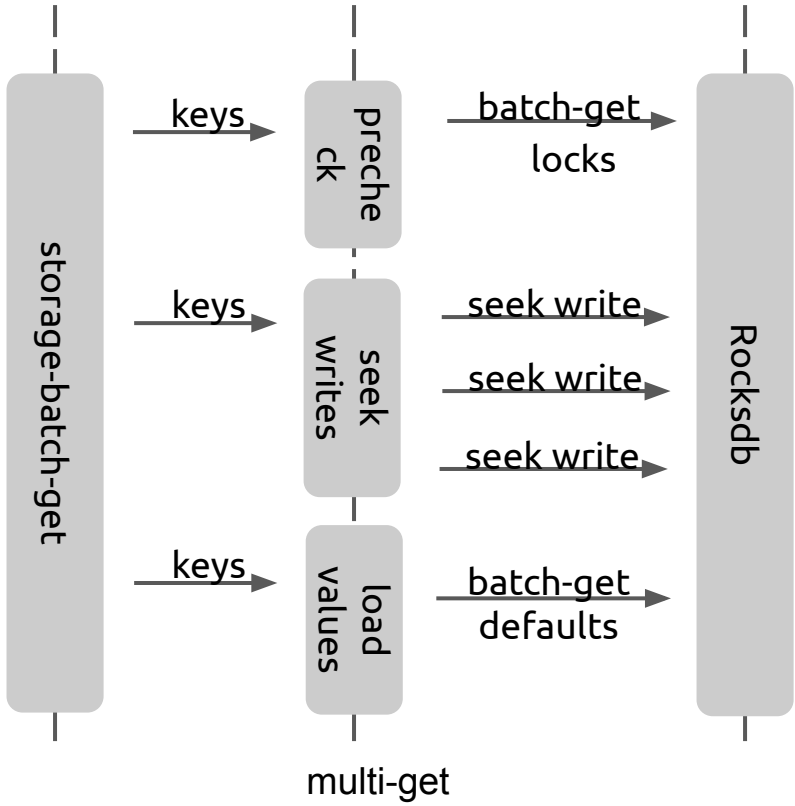


From facebook team

For TiKV ?

select * from table where (a, b, c) in ((1,2,3),(2,4,5));



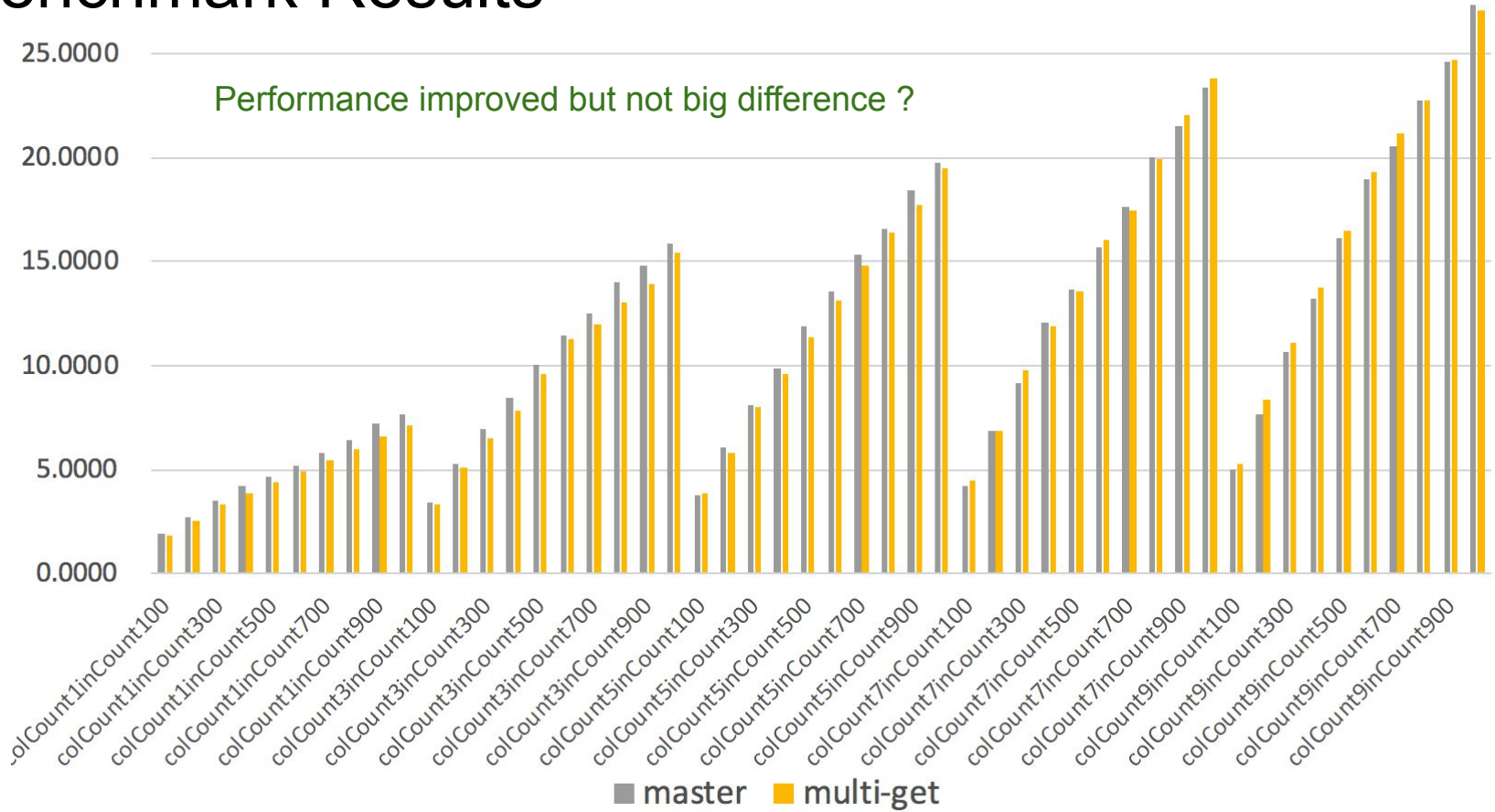


Let’s benchmark the TiKV 

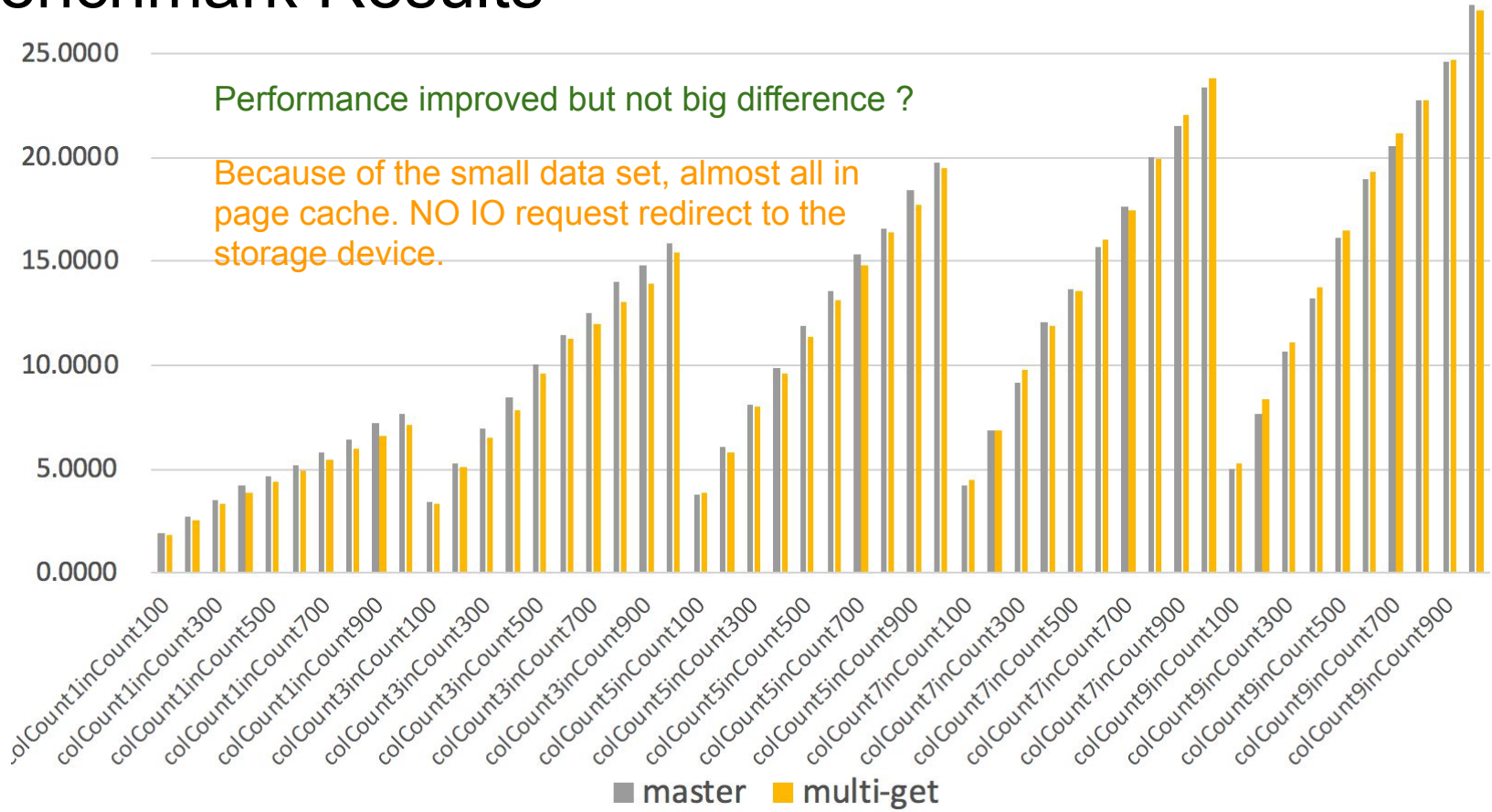
- Prepare
  - Set the Rocksdb config (Multi-Reads only supported in one SST now):
    - Disable the block cache.
    - write-buffer-size=500MB
    - target-file-size-base=500MB

    } Ensure that only one SST in the RocksDB
  - Load a small (50MB) data set
    - Flush the memstore to make it to be a SST.
- Benchmark running
  - Run the SQL few minutes.
    - Such as: select * from table where (a, b, c) in ((1,2,3),(2,4,5));

Benchmark Results

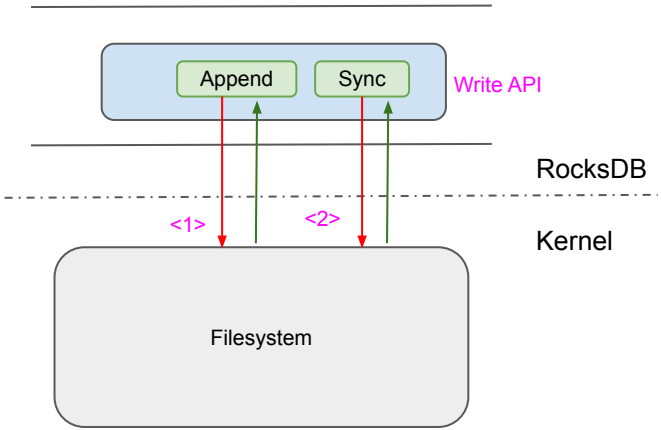


Benchmark Results

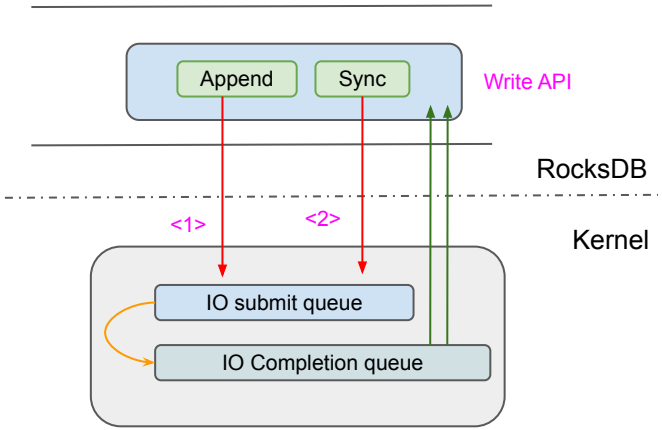


Case#2: RocksDB can benefit more from io_uring ?

- Rewrite the write+sync WAL in RocksDB by using io_uring
  - https://github.com/PingCAP-Hackthon2019-Team17/rocksdb/pull/1



Before io_uring

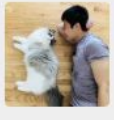


After io_uring

RocksDB Performance Improvement

吴毅

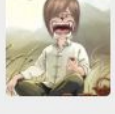
sync write:
uring: fillrandom : 96.676 micros/op 10343 ops/sec; 1.1 MB/s
master: fillrandom : 99.899 micros/op 10010 ops/sec; 1.1 MB/s



吴毅

no-sync write:
uring: fillrandom : 2.143 micros/op 466612 ops/sec; 51.6 MB/s
master: fillrandom : 2.209 micros/op 452624 ops/sec; 50.1 MB/s

这个看着更好一点。



RocksDB Performance Improvement

吴毅

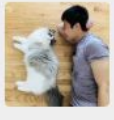
Write key-value with a fsync in RocksDB

sync write:

uring: fillrandom : 96.676 micros/op 10343 ops/sec; 1.1 MB/s

master: fillrandom : 99.899 micros/op 10010 ops/sec; 1.1 MB/s

ops/sec: +3.3%



吴毅

Write key-value without a fsync in RocksDB

no-sync write:

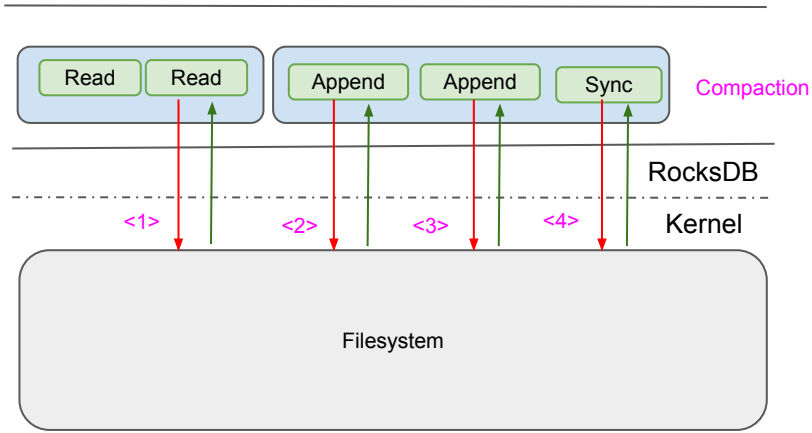
uring: fillrandom : 2.143 micros/op 466612 ops/sec; 51.6 MB/s

master: fillrandom : 2.209 micros/op 452624 ops/sec; 50.1 MB/s

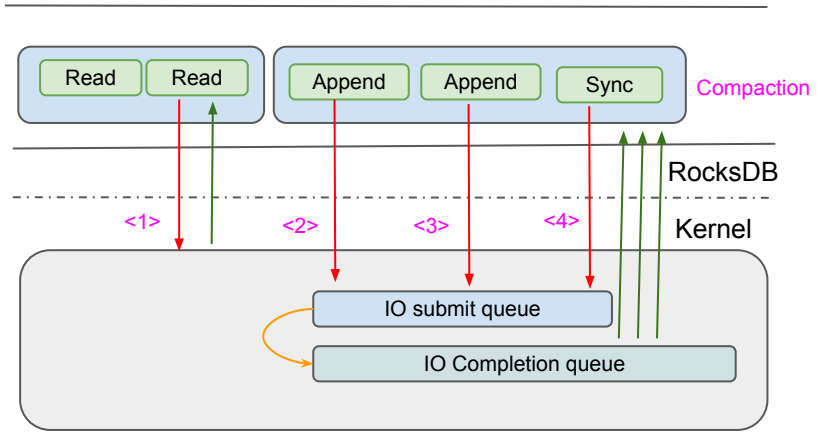
ops/sec: +3.1%

这个看着更好一点。

Case#3: Rewrite the compaction



Before io_uring



After io_uring

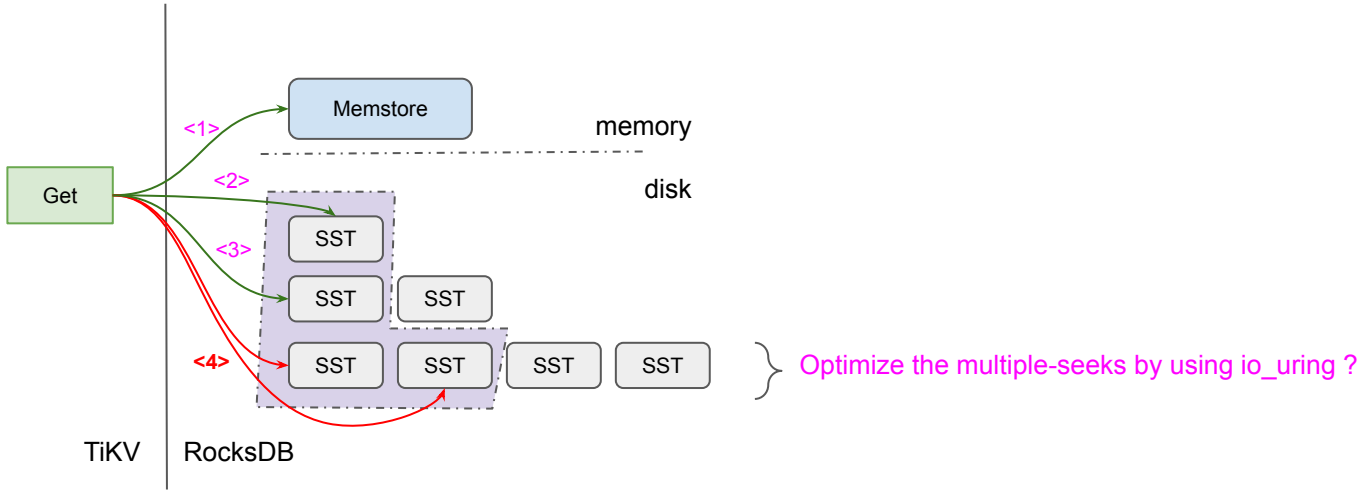
Case#3: Rewrite the compaction by io_uring

	master	uring
compaction time :	5163ms	4942ms
compaction cpu time:	3567ms	3484ms
file write time :	604ms	309ms
range sync time :	94ms	1ms
fsync time :	195ms	188ms

File write time decreased ~50%

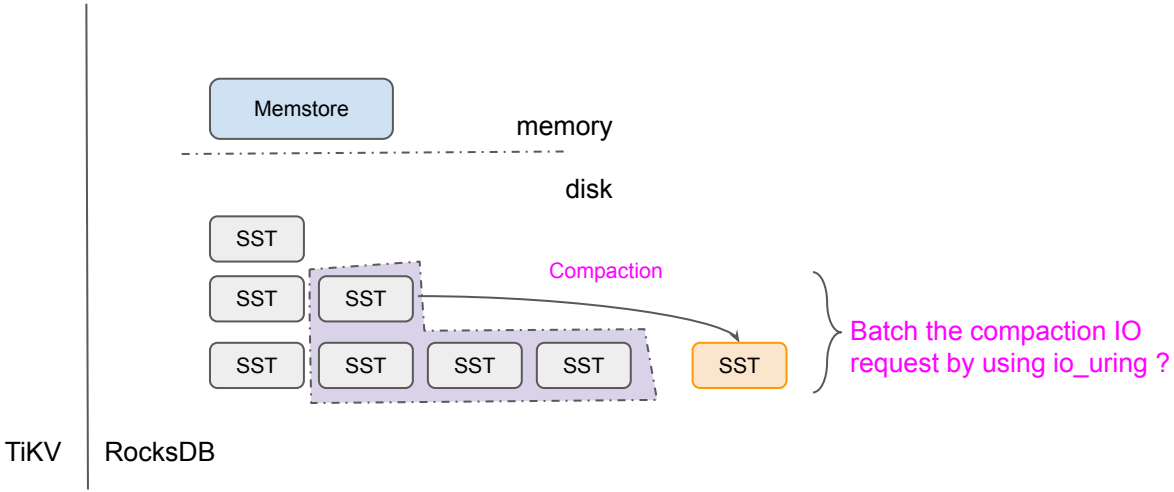
Conclusion & Future work

- One RPC to TiKV which would produce multiple IO requests to Filesystem
  - Example#1: One Get with multiple disk seek & read ?



Conclusion & Future work

- One RPC to TiKV which would produce multiple IO requests to FS
  - Example#2: batch the compaction IO request by using io_uring ?



Conclusion & Future work

- One RPC to TiKV which would produce multiple IO requests to FS
  - More example ….

Reference

1. https://github.com/PingCAP-Hackthon2019-Team17
2. https://github.com/facebook/rocksdb/pull/5881/files
3. https://www.slideshare.net/ennael/kernel-recipes-2019-faster-io-through-iouring
4. http://git.kernel.dk/cgit/liburing/tree/