RocksDB Festival

RF5_Team_Key_Value

reflected Q&A about key distribution(p5)

Supported by IITP, StarLab.

July 19, 2021

Minguk Choi, Jungwon Lee, Guangxun shin
koreachoi96@gmail.com, gardenlee960828@gmail.com, guangxun0621@naver.com

Docks

RocksDB Festival

- 1. Team
- 2. db_bench Experiment
  - ✓ 1. key_size
  - ✓ 2. value_size
  - ✓ 3. data_size
- 3. Topic: Key-value
- 4. Goals
  - ✓ 1. Latest research trends
    - Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook
  - ✓ 2. Research topic
  - ✓ 3. Final goal

1. Team

- Team: Docks

- Member
  - ✓ Minguk Choi 최민국 [Leader]
    - **koreachoi96@gmail.com**
    - **www.github.com/korea-choi**
  - ✓ Jungwon Lee 이정원
    - **gardenlee960828@gmail.com**
    - **www.github.com/gardenlee96**
  - ✓ Guangxun shin 좌오꾸와쉰
    - **guangxun0621@naver.com**
    - **www.github.com/GUANG32194441**

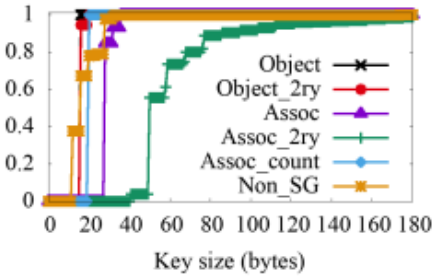
2. db_bench Experiment

- 0. Experiment subject
  - ✓ Check out read/write throughput when [key/value/data] size changes
    - Just simple experiment
    - Just to be familiar with rocksdb & db_bench

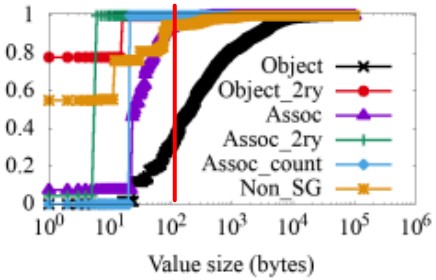
```
Initializing RocksDB Options from the specified file
Initializing RocksDB Options from command-line flags
RocksDB:    version 6.22
Date:       Sun Jul 18 16:50:59 2021
CPU:        4 * Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz
CPUCache:   3072 KB
Keys:       16 bytes each (+ 0 bytes user-defined timestamp)
Values:     100 bytes each (50 bytes after compression)
Entries:    10000000
Prefix:    0 bytes
Keys per prefix:    0
RawSize:    1106.3 MB (estimated)
FileSize:   629.4 MB (estimated)
Write rate: 0 bytes/second
Read rate: 0 ops/second
Compression: Snappy
Compression sampling rate: 0
Memtablerep: skip_list
Perf Level: 1
WARNING: Assertions are enabled; benchmarks unnecessarily slow
```

2. db_bench Experiment

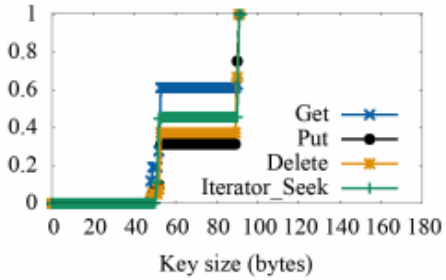
- 1. Value_size
  - ✓ Workload
    - Key_size = 16 Byte / Data_number = 10,000,000
    - Value_size = 100/200/400Byte
  - ✓ Real-World key/value distribution: [value size>100B] is not general



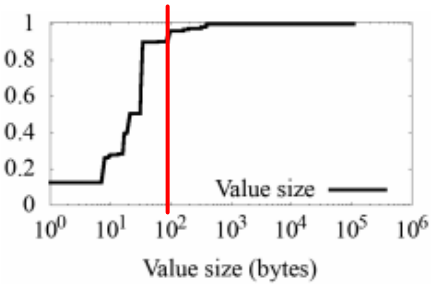
(a) UDB key size CDF



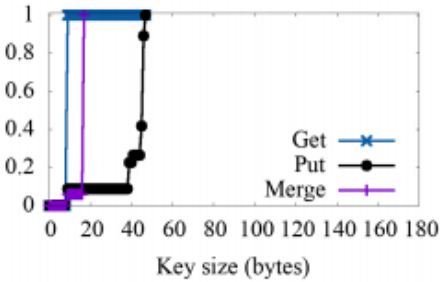
(b) UDB value size CDF



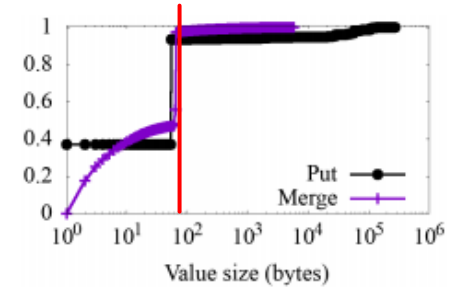
(c) ZippyDB key size CDF



(d) ZippyDB value size CDF



(e) UP2X key size CDF



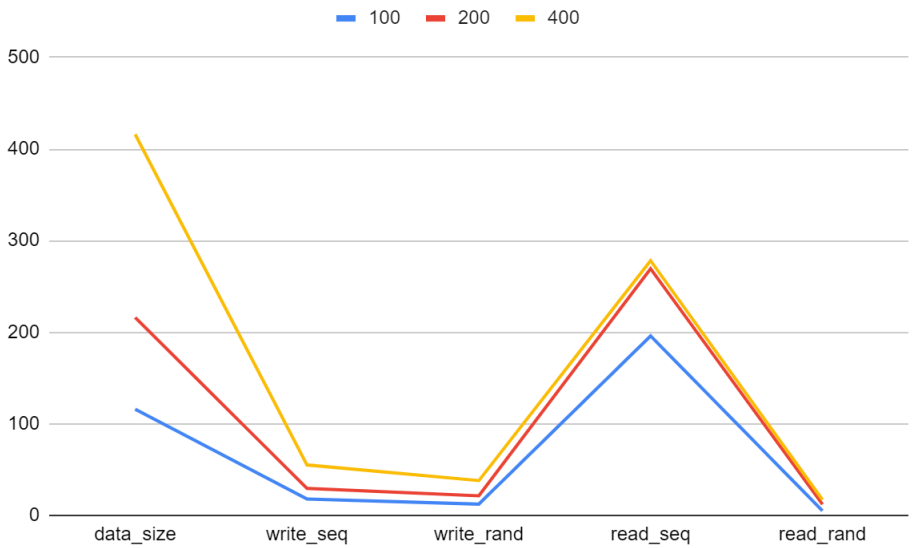
(f) UP2X value size CDF

2. db_bench Experiment

- 1. Value_size
  - ✓ Workload
    - Key_size = 16 Byte / Data_number = 10,000,000
    - Value_size = 100/200/400Byte
    - write_seq / write_rand / read_seq / read_rand

value_size (B)	data_size	write_seq	write_rand	read_seq	read_rand
100	116	17.8	12.3	195.9	5
200	216	29.4	21.3	269.2	12.1
400	416	55	38	278	17

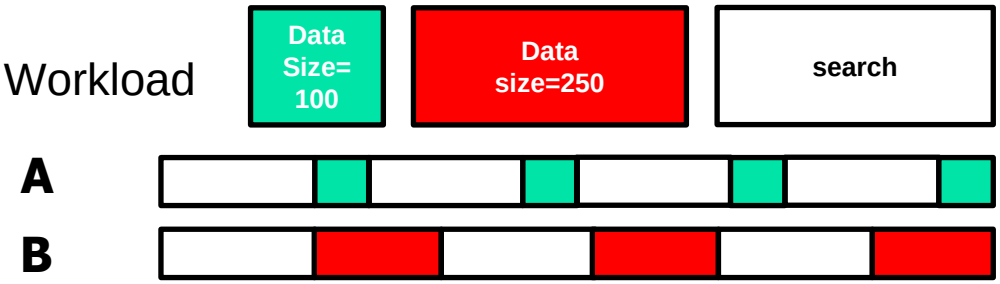
Ratio (/K16-V100)	data_size	write_seq	write_rand	read_seq	read_rand
K16-V100	1.00	1.00	1.00	1.00	1.00
K16-V200	1.86	1.65	1.73	1.37	2.42
K16-V400	3.59	3.09	3.09	1.42	3.40



  - ✓ value size ∝ throughput
    - the bigger value size, the better performance? Maybe Not.
    - More flush/compactions, longer total execution time

2. db_bench Experiment

- 1. Value_size
  - ✓ the bigger value size, the better performance? Maybe Not.
  - ✓ Why?
    - Throughput is rate of transferring data
  - ✓ EX.
    - Workload



    - Throughput = $\frac{read\_data}{time}$ -> Ta < Tb

      -> B's performance is better than A?

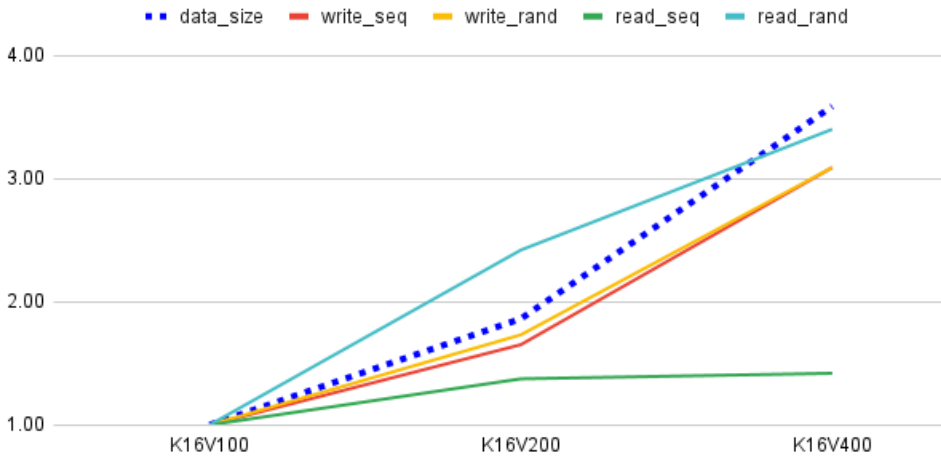
  - ✓ Data size ∝ throughput
  - ✓ If data size is different, how about comparing performance with…
    - Compaction number
    - Total execution time

2. db_bench Experiment

- 1. Value_size
  - ✓ Ratio
    - Ratio = [K16, V100/200/400] value / [K16, V100] value

value_size (B)	data_size	write_seq	write_rand	read_seq	read_rand
100	116	17.8	12.3	195.9	5
200	216	29.4	21.3	269.2	12.1
400	416	55	38	278	17

Ratio (/K16-V100)	data_size	write_seq	write_rand	read_seq	read_rand
K16-V100	1.00	1.00	1.00	1.00	1.00
K16-V200	1.86	1.65	1.73	1.37	2.42
K16-V400	3.59	3.09	3.09	1.42	3.40



Ratio = [K16, V100/200/400] value / [K16, V100] value

- ✓ data size ∝ throughput
- ✓ if data size is different,
  - maybe throughput would not be the appropriate metrics.

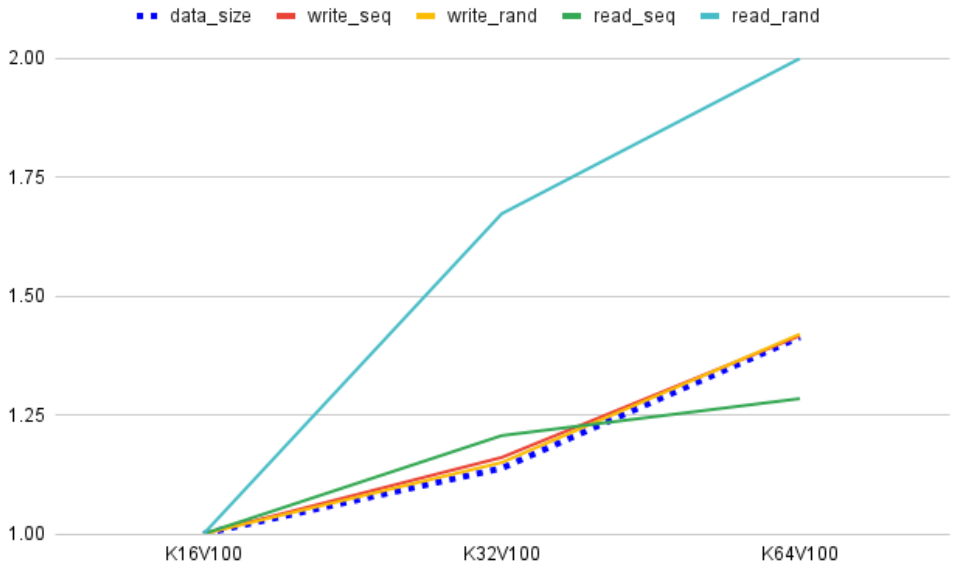
2. db_bench Experiment

- 2. Key_size
  - ✓ Workload
    - Key_size = 16/32/64 Byte
    - Value_size = 100 Byte / Data_number = 10,000,000
    - write_seq/write_rand/read_seq/read_rand
  - ✓ The bigger the key size, the bigger overhead

    -> But data size is much more critical

value_size	data_size	write_seq	write_rand	read_seq	read_rand
16 B	116	16.8	11.8	193.1	4.9
32 B	132	19.5	13.7	233	8.2
64 B	164	23.8	16.7	248	9.8

Ratio (/K16-V100)	data_size	write_seq	write_rand	read_seq	read_rand
K16-V100	1.00	1.00	1	1.00	1.00
K32-V100	1.14	1.16	1.15	1.21	1.67
K64-V100	1.41	1.42	1.42	1.28	2.00



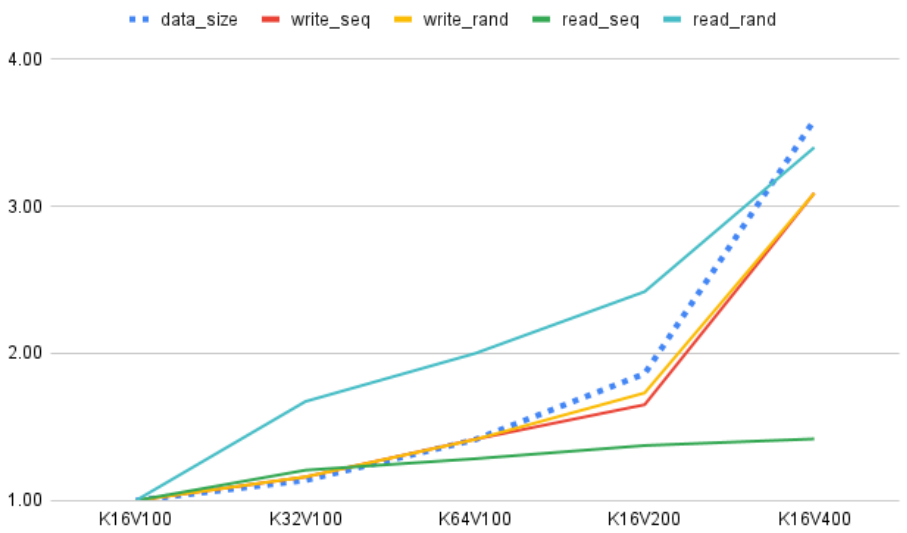
Ratio = [K16/32/64, V100] value / [K16, V100] value

2. db_bench Experiment

- 3. Data_size
  - ✓ Workload
    - Key_size = 16/32/64 Byte / Value_size = 100/200/400 Byte
    - Data_number = 10,000,000
    - write_seq/write_rand/read_seq/read_rand

data_size (B)	write_seq	write_rand	read_seq	read_rand
116	17.8	12.3	195.9	5
132	19.5	13.7	233	8.2
164	23.8	16.7	248	9.8
216	29.4	21.3	269.2	12.1
416	55	38	278	17

Ratio (/K16-V100)	data_size	write_seq	write_rand	read_seq	read_rand
K16-V100	1.00	1.00	1.00	1.00	1.00
K32-V100	1.14	1.16	1.16	1.21	1.67
K64-V100	1.41	1.42	1.42	1.28	2.00
K16-V200	1.86	1.65	1.73	1.37	2.42
K16-V400	3.59	3.09	3.09	1.42	3.40



Ratio = [K16/32/64, V100/200/400] value / [K16, V100] value

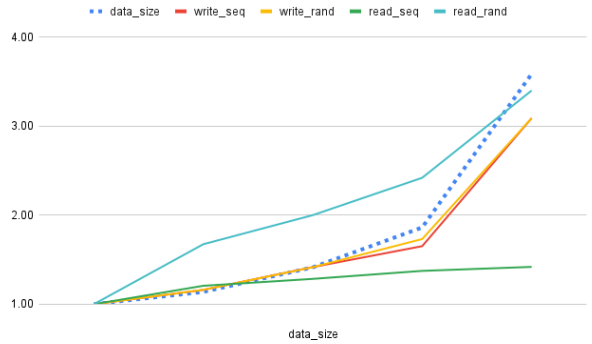
- ✓ Why read_ratio is not directly proportional?

2. db_bench Experiments

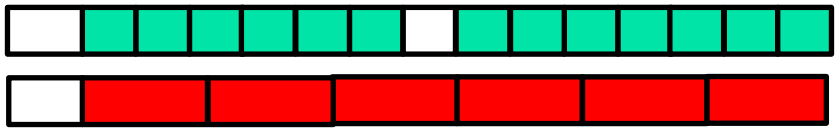
- 4. Why read_ratio is not directly proportional

✓ **Workload**

✓ **Throughput** = $\frac{read\_data}{time}$

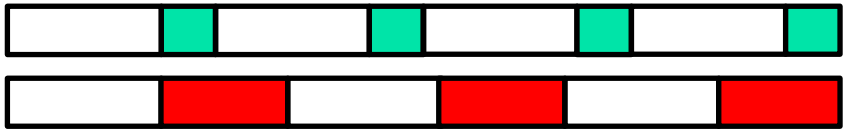


✓ **read_sequential: iterator**



```
void ReadSequential(ThreadState* thread, DB* db) {
Iterator* iter = db->NewIterator(options);
int64_t i = 0;
int64_t bytes = 0;
for (iter->SeekToFirst(); i < reads_ && iter->Valid(); iter->Next()) {
  bytes += iter->key().size() + iter->value().size();
  thread->stats.FinishedOps(nullptr, db, 1, kRead);
  ++i;
```

✓ **read_random: Get**



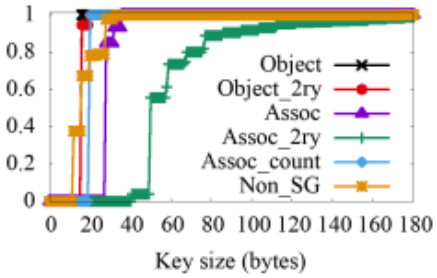
```
void ReadRandom(ThreadState* thread) {
if (FLAGS_num_column_families > 1) {
  s = db_with_cfh->db->Get(options, db_with_cfh->GetCfh(key_rand), key,
                           &pinnable_val, ts_ptr);
} else {
  s = db_with_cfh->db->Get(options,
                           db_with_cfh->db->DefaultColumnFamily(), key,
                           &pinnable_val, ts_ptr);
}
```

2. db_bench Experiments

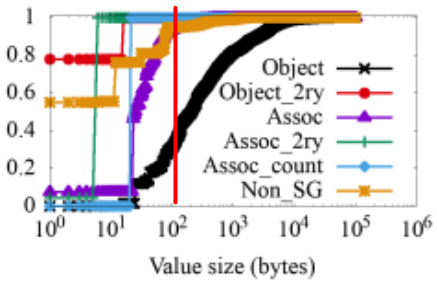
- 5. Summary
  - ✓ data size ∝ throughput
    - the higher throughput, the better performance? Maybe Not.
  - ✓ if data size is different,
    - throughput would may not be the appropriate metrics.
    - compare with Compaction number/Total execution time
  - ✓ Why read_ratio is not directly proportional?
    - read_sequential: Iterator
    - read_random: Get

Experiments Q & A

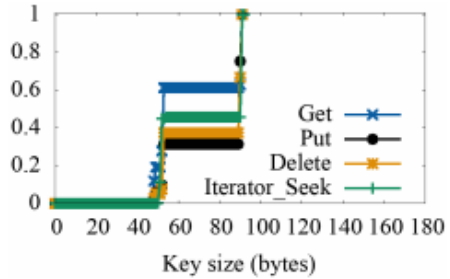
- ✓ Real-World key/value distribution
  -> [value size>100B] is not general



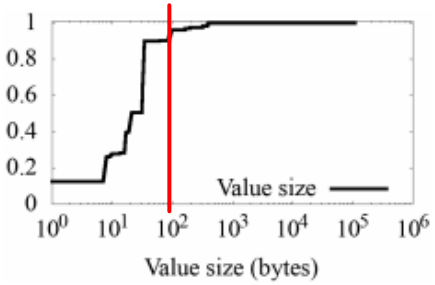
(a) UDB key size CDF



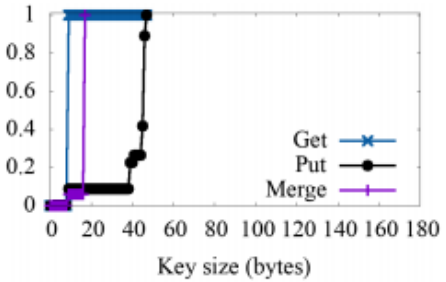
(b) UDB value size CDF



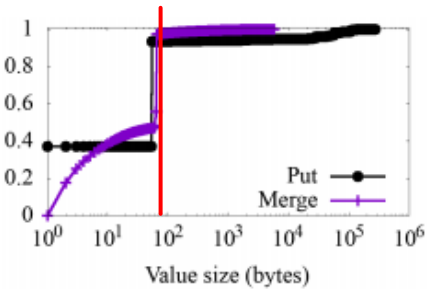
(c) ZippyDB key size CDF



(d) ZippyDB value size CDF



(e) UP2X key size CDF



(f) UP2X value size CDF

<source>

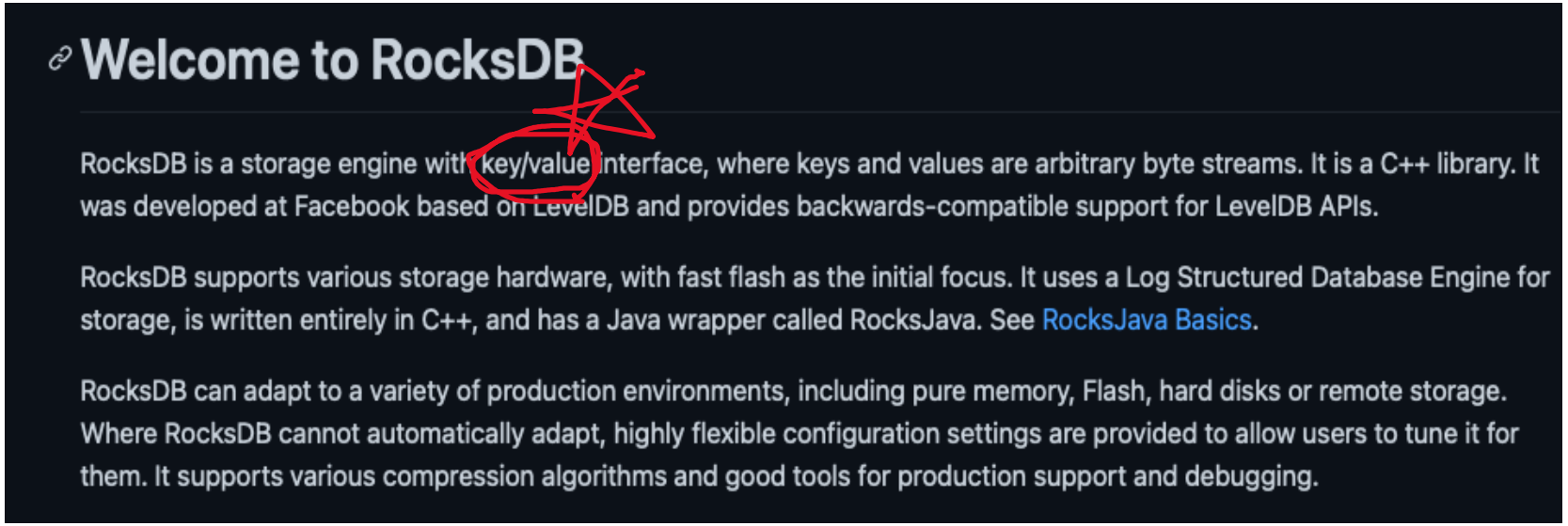
Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook

Zhichao Cao, *University of Minnesota, Twin Cities, and Facebook;*
Siying Dong and Sagar Vemuri, *Facebook;*
David H.C. Du, *University of Minnesota, Twin Cities*

https://www.usenix.org/conference/fast20/presentation/cao-zhichao

3. Topic: Key/Value

- RocksDB is Key/Value DB



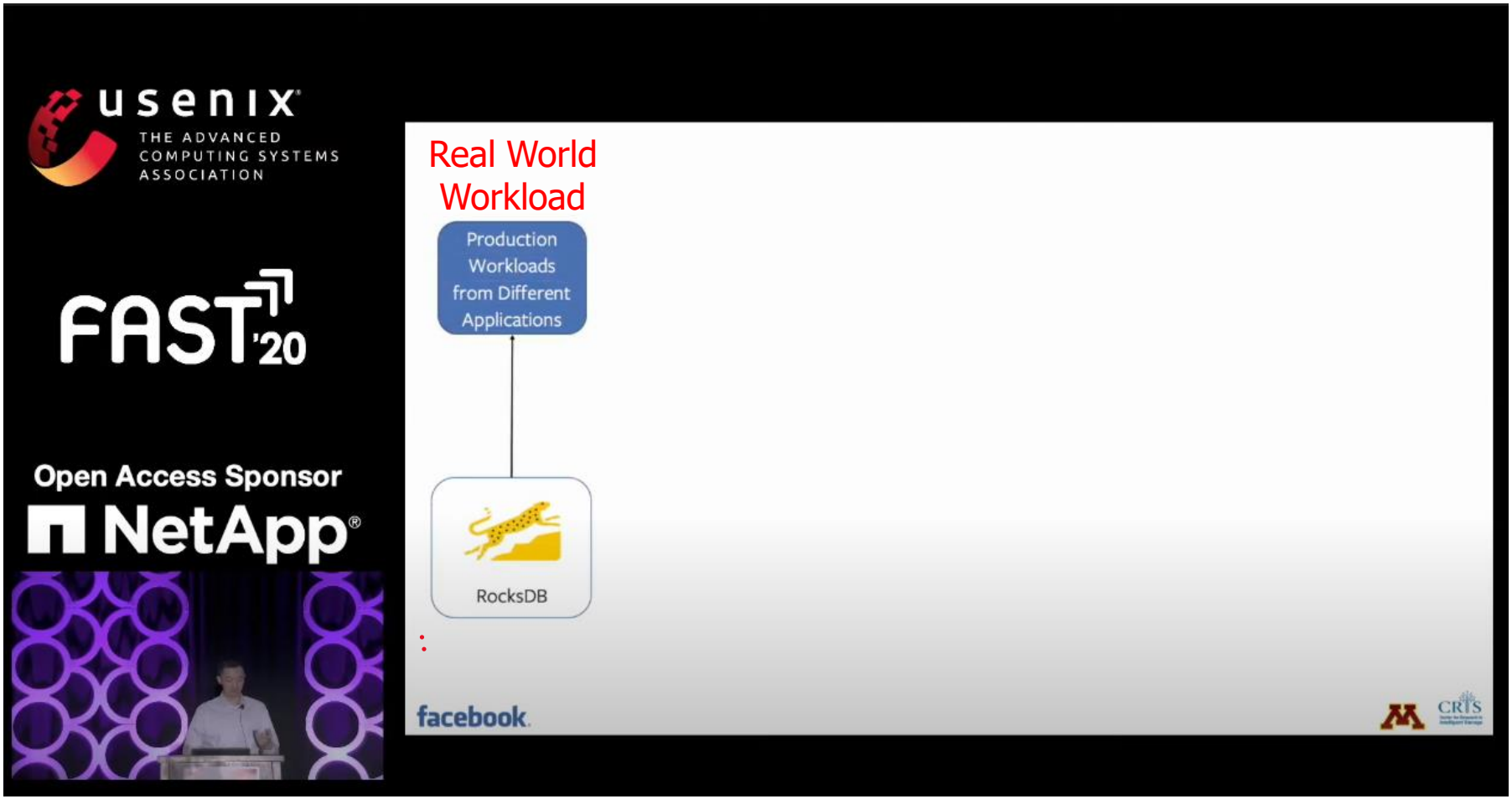
4. Goal

- 1. Latest research trends: Key Trace

Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook

Zhichao Cao, *University of Minnesota, Twin Cities, and Facebook;*
Siying Dong and Sagar Vemuri, *Facebook;*
David H.C. Du, *University of Minnesota, Twin Cities*

https://www.usenix.org/conference/fast20/presentation/cao-zhichao



Source: FAST '20 - Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook (https://youtu.be/MZTSjBERXVc)

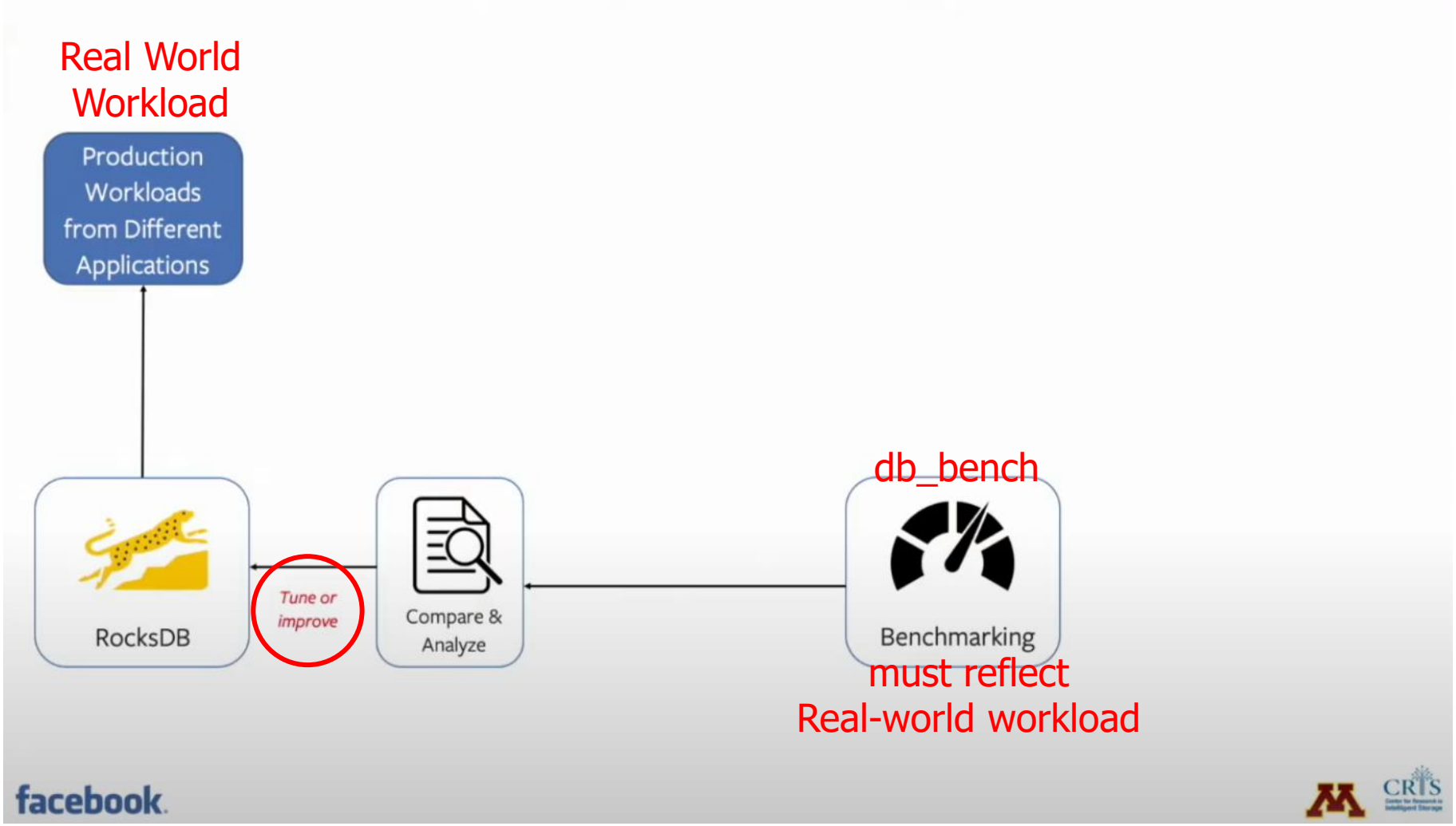
4. Goal

- 1. Latest research trends: Key Trace

Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook

Zhichao Cao, *University of Minnesota, Twin Cities, and Facebook;*
Siying Dong and Sagar Vemuri, *Facebook;*
David H.C. Du, *University of Minnesota, Twin Cities*

https://www.usenix.org/conference/fast20/presentation/cao-zhichao



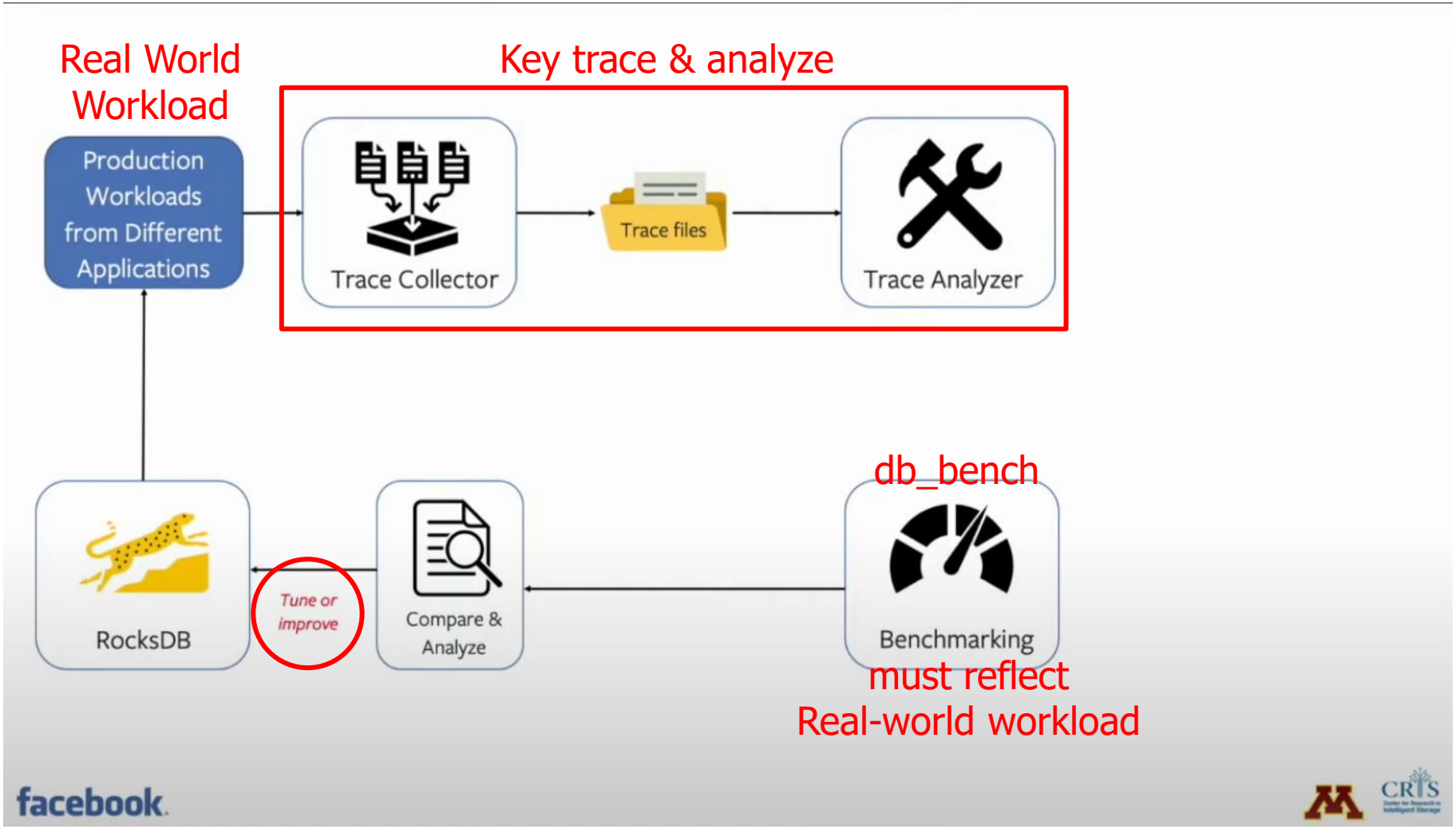
4. Goal

- 1. Latest research trends: Key Trace

Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook

Zhichao Cao, *University of Minnesota, Twin Cities, and Facebook;*
Siying Dong and Sagar Vemuri, *Facebook;*
David H.C. Du, *University of Minnesota, Twin Cities*

https://www.usenix.org/conference/fast20/presentation/cao-zhichao



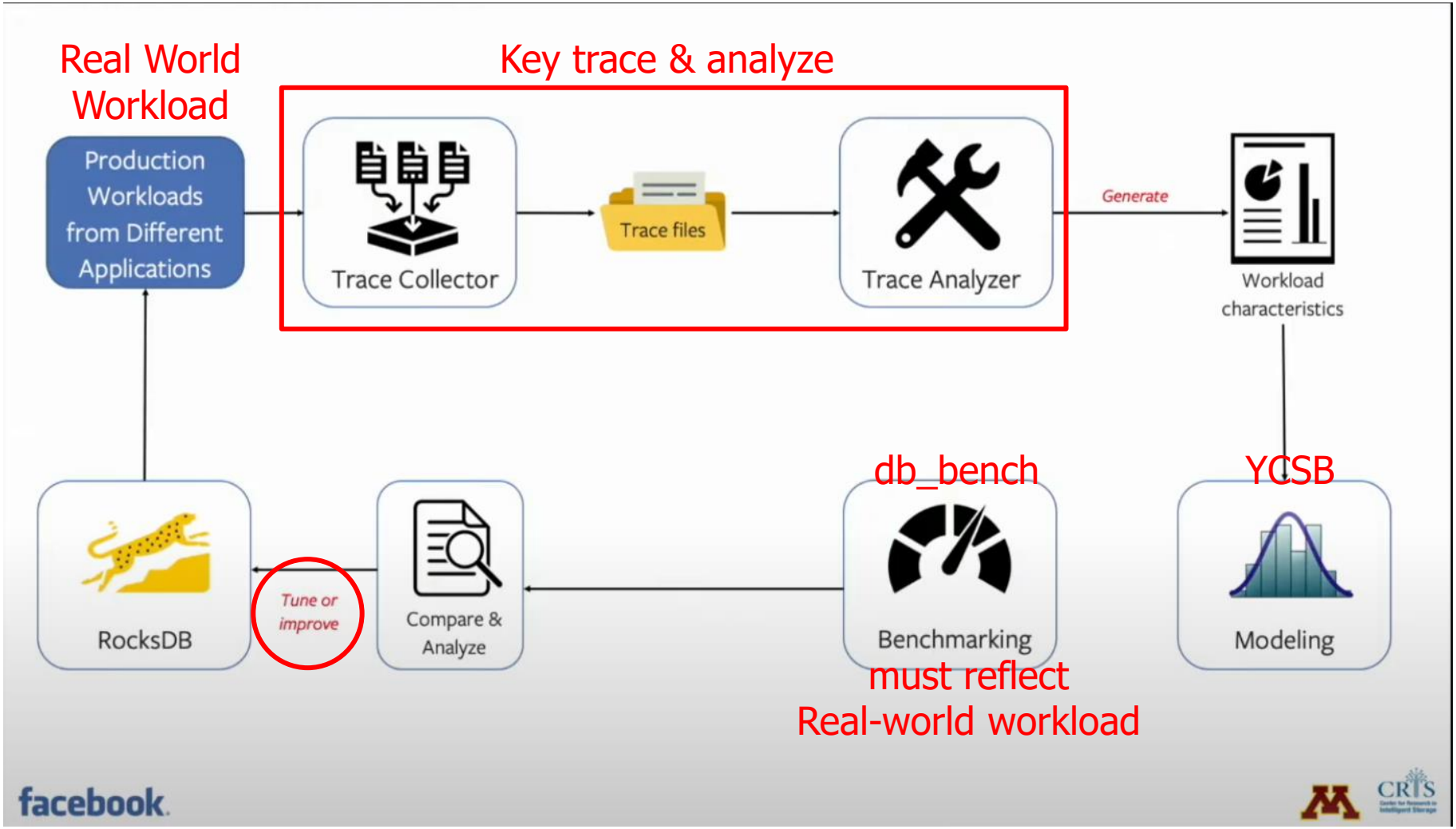
4. Goal

- 1. Latest research trends: Key Trace

Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook

Zhichao Cao, *University of Minnesota, Twin Cities, and Facebook;*
Siying Dong and Sagar Vemuri, *Facebook;*
David H.C. Du, *University of Minnesota, Twin Cities*

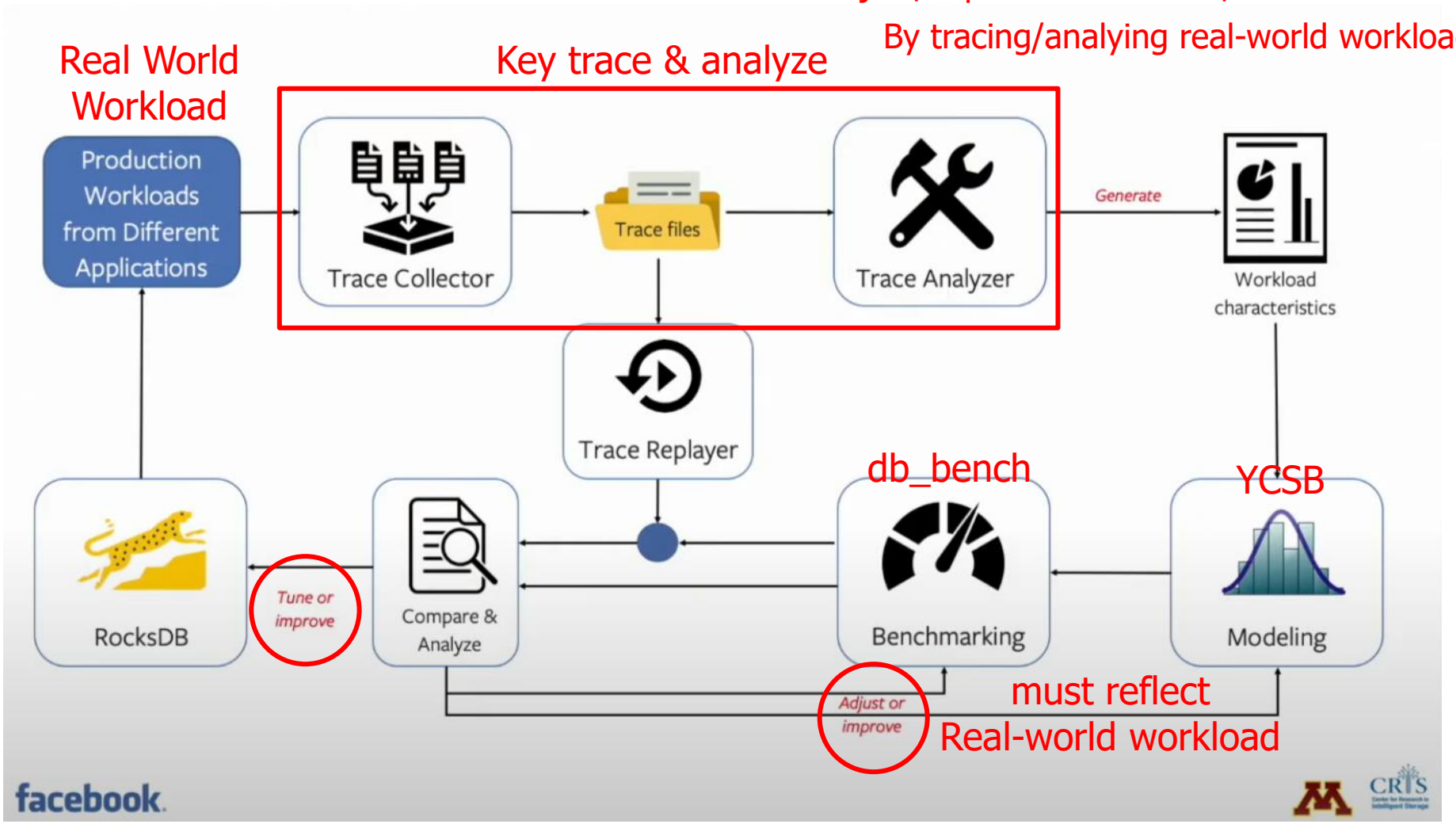
https://www.usenix.org/conference/fast20/presentation/cao-zhichao



4. Goal

- 1. Latest research trends: Key Trace

For RocksDB in real-world
Adjust/improve benchmark/virtual workload
By tracing/analying real-world workload



4. Goal

- 2. Research topic: One(Two) step forward
  - ✓ Analyze new real-world workload
    - Getting new real-world workload is much harder than analyzing it
  - ✓ Analyze existing virtual workload
    - It's already analyzed (maybe)
  - ✓ Generate/Adjust virtual workload
    - Based on open-source real-world workload
  - ✓ Adjust/Improve rocksdb/db_bench
    - Based on open-source real-world workload
  - ✓ Other open issues/questions
- 3. Final Goal: KSC 2021 submission

Q & A