

# RocksDB Festival

RF5\_Team\_Key\_Value

revised opinion about key distribution (p2)

Supported by IITP, StarLab.

July 19, 2021

Minguk Choi, Jungwon Lee, Guangxun shin

[koreachoi96@gmail.com](mailto:koreachoi96@gmail.com), [gardenlee960828@gmail.com](mailto:gardenlee960828@gmail.com), [guangxun0621@naver.com](mailto:guangxun0621@naver.com)

Docks

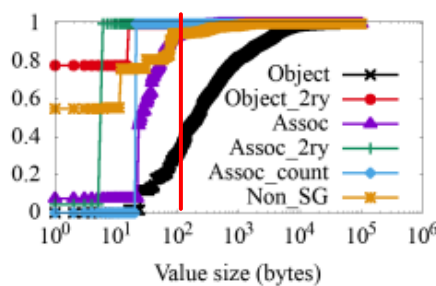
# Experiments Q & A



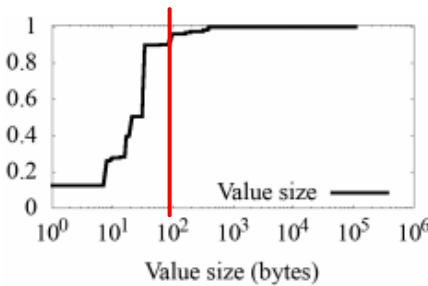
- ✓ Real-World key/value distribution  
value over 100B/1KB is ~~not general~~ -> it depends on workload

- Facebook: [value > 100B] is **not general**

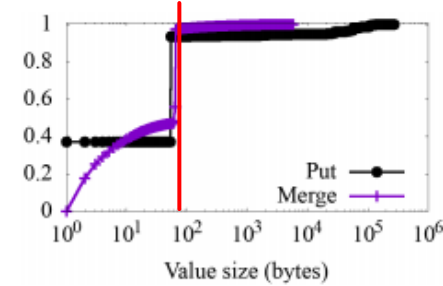
Cao, Zhichao, et al. "Characterizing, modeling, and benchmarking rocksdb key-value workloads at facebook." *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20)*. 2020.



(b) UDB value size CDF



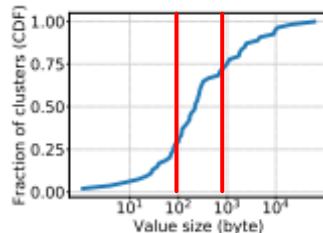
(d) ZippyDB value size CDF



(f) UP2X value size CDF

- Twitter: [value > 100B] is **general**, Also [value > 1KB] is **general**

Source: Yang, Juncheng, Yao Yue, and K. V. Rashmi. "A large scale analysis of hundreds of in-memory cache clusters at Twitter." *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*. 2020.



(b) Value size

- 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](mailto:koreachoi96@gmail.com)
  - [www.github.com/korea-choi](https://www.github.com/korea-choi)
- ✓ Jungwon Lee 이정원
  - [gardenlee960828@gmail.com](mailto:gardenlee960828@gmail.com)
  - [www.github.com/gardenlee96](https://www.github.com/gardenlee96)
- ✓ Guangxun shin 좌오꾸와썬
  - [guangxun0621@naver.com](mailto:guangxun0621@naver.com)
  - [www.github.com/GUANG32194441](https://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

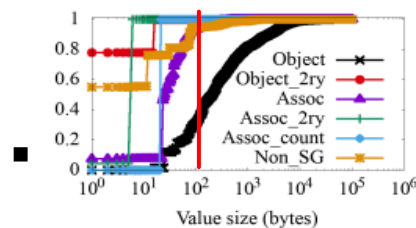
```
Initializing RocksDB Options from the specified file
Initializing RocksDB Options from command-line flags
RocksDB: version 6.22
Date: Sun Jul 18 16:50:59 2021
CPU: 4 * Intel(R) Core(TM) i3-2100 CPU @ 3.10GHz
CPUCache: 3072 KB
Keys: 16 bytes each (+ 0 bytes user-defined timestamp)
Values: 100 bytes 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

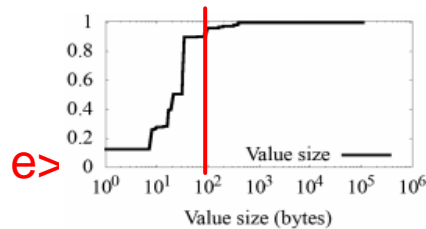
### 1. Value\_size

#### ✓ Workload

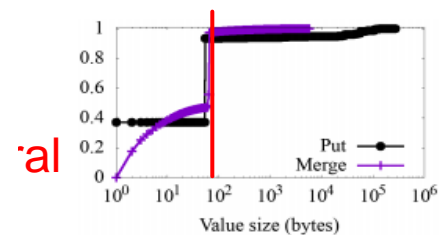
- Key\_size = 16 Byte / Data\_number = 10,000,000
- Value\_size = 100/200/400Byte



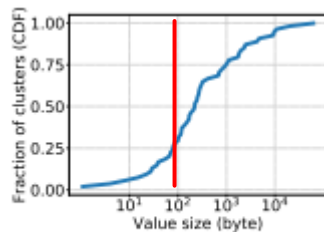
(b) UDB value size CDF



(d) ZippyDB value size CDF



(f) UP2X value size CDF



(b) Value size

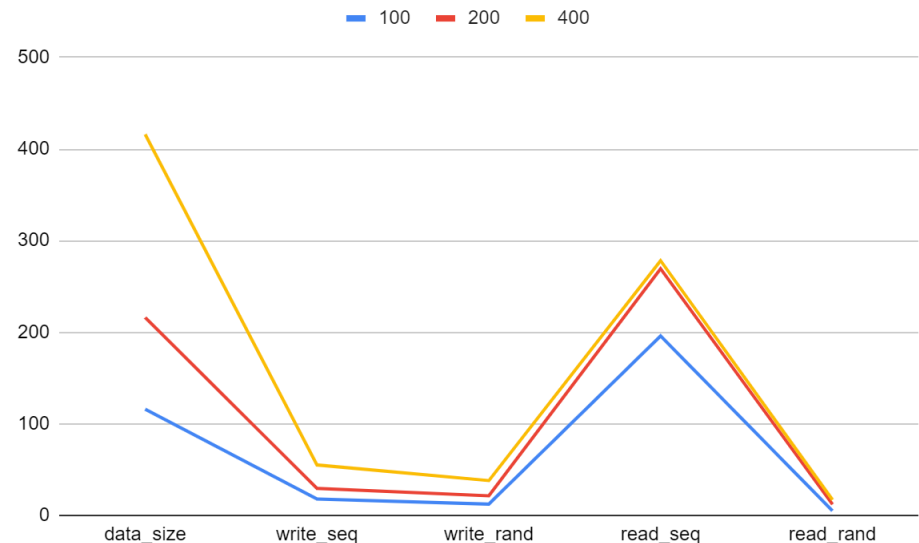
## 2. db\_bench Experiment

### ■ 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)					
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 $\propto$ throughput

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

## 2. db\_bench Experiment

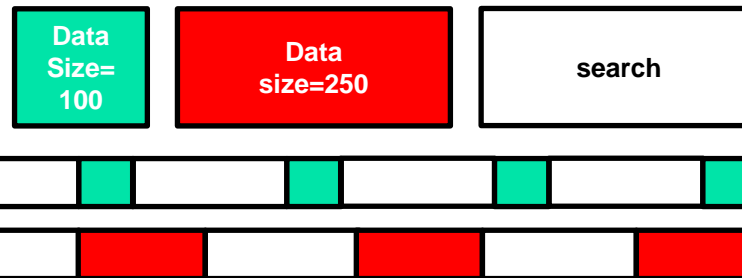
### ■ 1. Value\_size

- ✓ the bigger **value size**, the better **performance**? **Maybe Not.**
- ✓ Why?

- Throughput is rate of **transferring data**

✓ EX.

- Workload



- Throughput =  $\frac{read\_data}{time}$  ->  $T_a < T_b$   
-> B's performance is better than A?

- ✓ Data size  $\propto$  throughput
- ✓ If data size is different, how about comparing performance with...
  - **Compaction number**
  - **Total execution time**



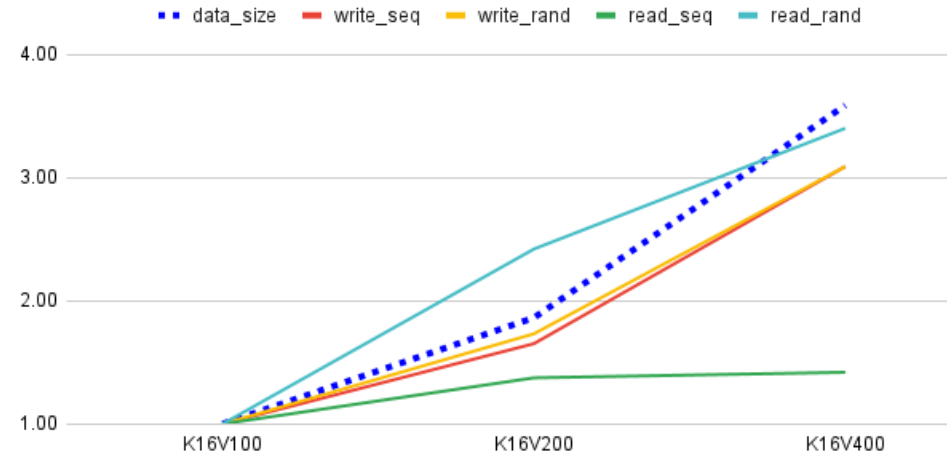
## 2. db\_bench Experiment

### ■ 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)					
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  $\propto$  throughput
- ✓ if data size is different,
  - maybe throughput would **not be the appropriate metrics.**

## 2. db\_bench Experiment

### ■ 2. Key\_size

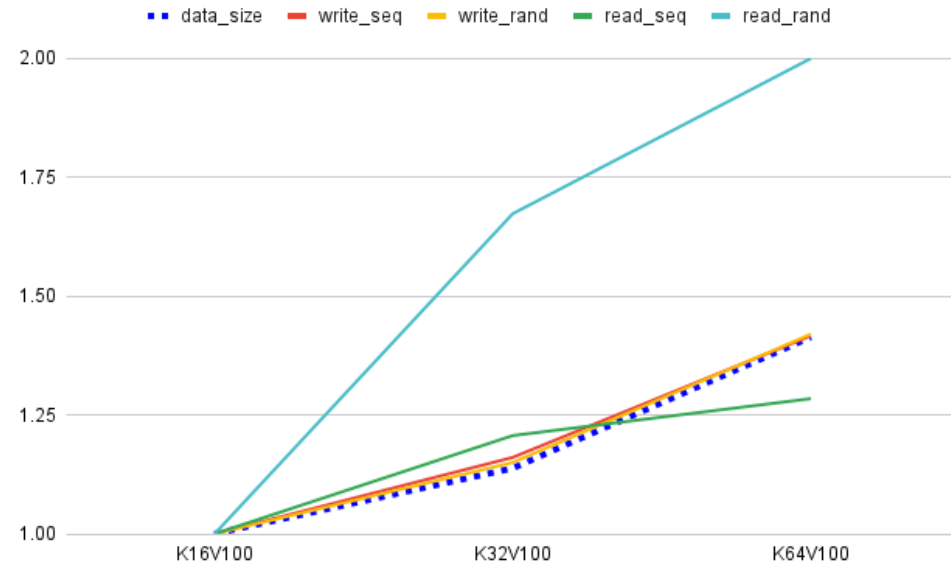
#### ✓ Workload

- 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

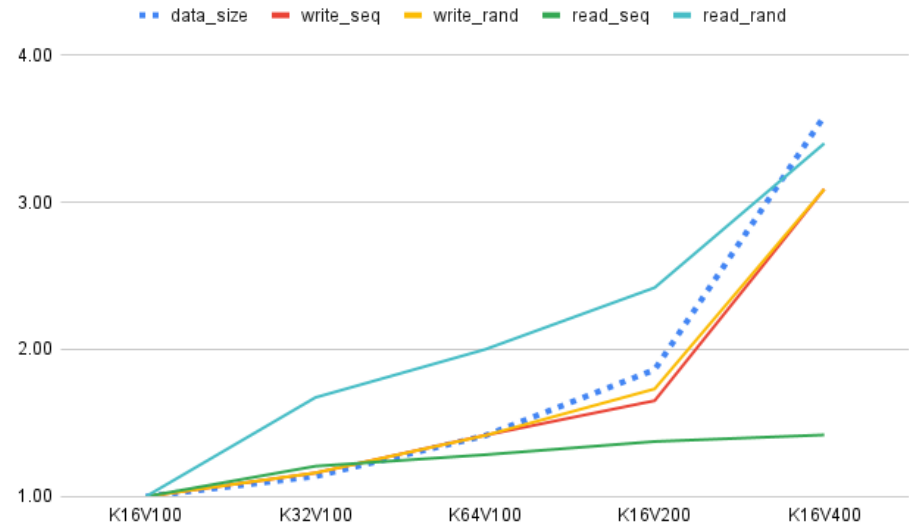
#### ✓ Workload

- 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?

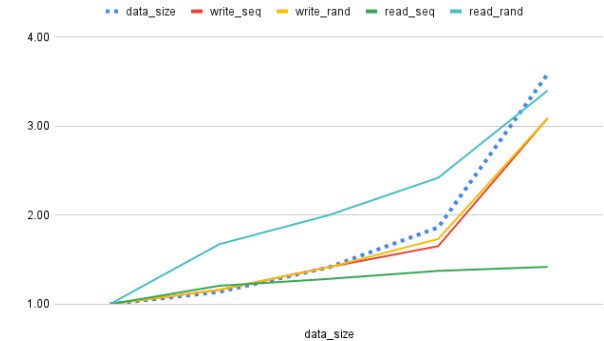
## 2. db\_bench Experiments

### ■ 4. Why read\_ratio is not directly proportional

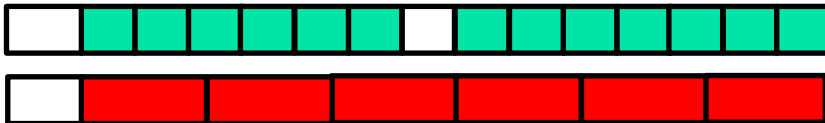
✓ **Workload**



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

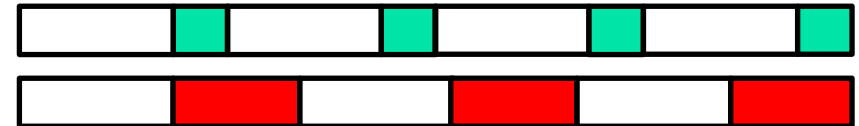


✓ **read\_sequential: iterator**



```
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  $\propto$  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**

# 3. Topic: Key/Value

## ■ RocksDB is Key/Value DB

### Welcome to RocksDB

RocksDB is a storage engine with key/value interface, where keys and values are arbitrary byte streams. It is a C++ library. It was developed at Facebook based on LevelDB and provides backwards-compatible support for LevelDB APIs.

RocksDB supports various storage hardware, with fast flash as the initial focus. It uses a Log Structured Database Engine for storage, is written entirely in C++, and has a Java wrapper called RocksJava. See [RocksJava Basics](#).

RocksDB can adapt to a variety of production environments, including pure memory, Flash, hard disks or remote storage. Where RocksDB cannot automatically adapt, highly flexible configuration settings are provided to allow users to tune it for them. It supports various compression algorithms and good tools for production support and debugging.

## 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>

The slide is from the FAST '20 conference, sponsored by NetApp. It features the Usenix logo and a diagram titled "Real World Workload". The diagram shows a box labeled "Production Workloads from Different Applications" at the top, with an arrow pointing down to a box labeled "RocksDB" which contains a cheetah logo. Below the RocksDB box is a red colon. The Facebook logo is at the bottom left, and logos for the University of Minnesota and CRIS are at the bottom right. A small inset photo of a speaker is in the bottom left corner.

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

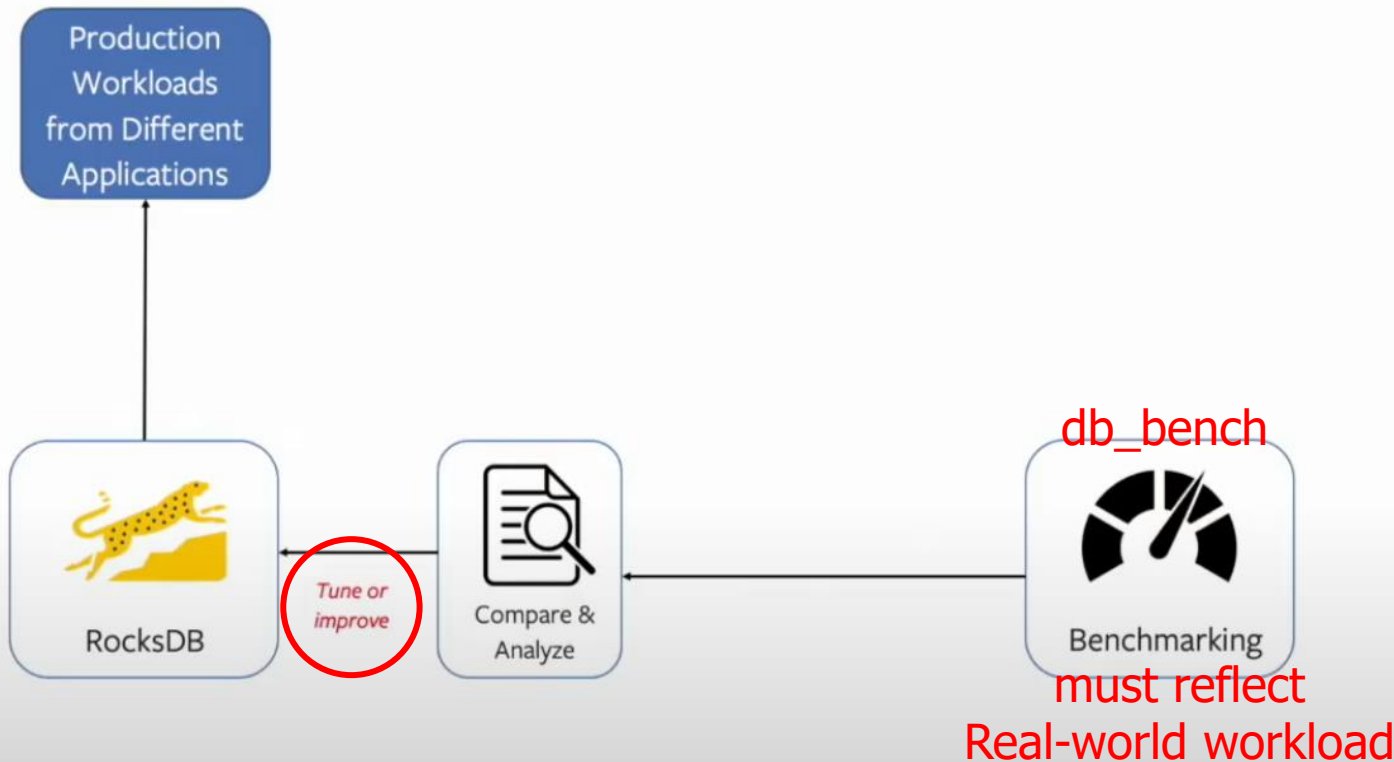
Zhichao Cao, *University of Minnesota, Twin Cities, and Facebook*;

Siying Dong and Sagar Vemuri, *Facebook*;

David H.C. Du, *University of Minnesota, Twin Cities*

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

Real World  
Workload





# 4. Goal

## ■ 1. Latest research trends: Key Trace

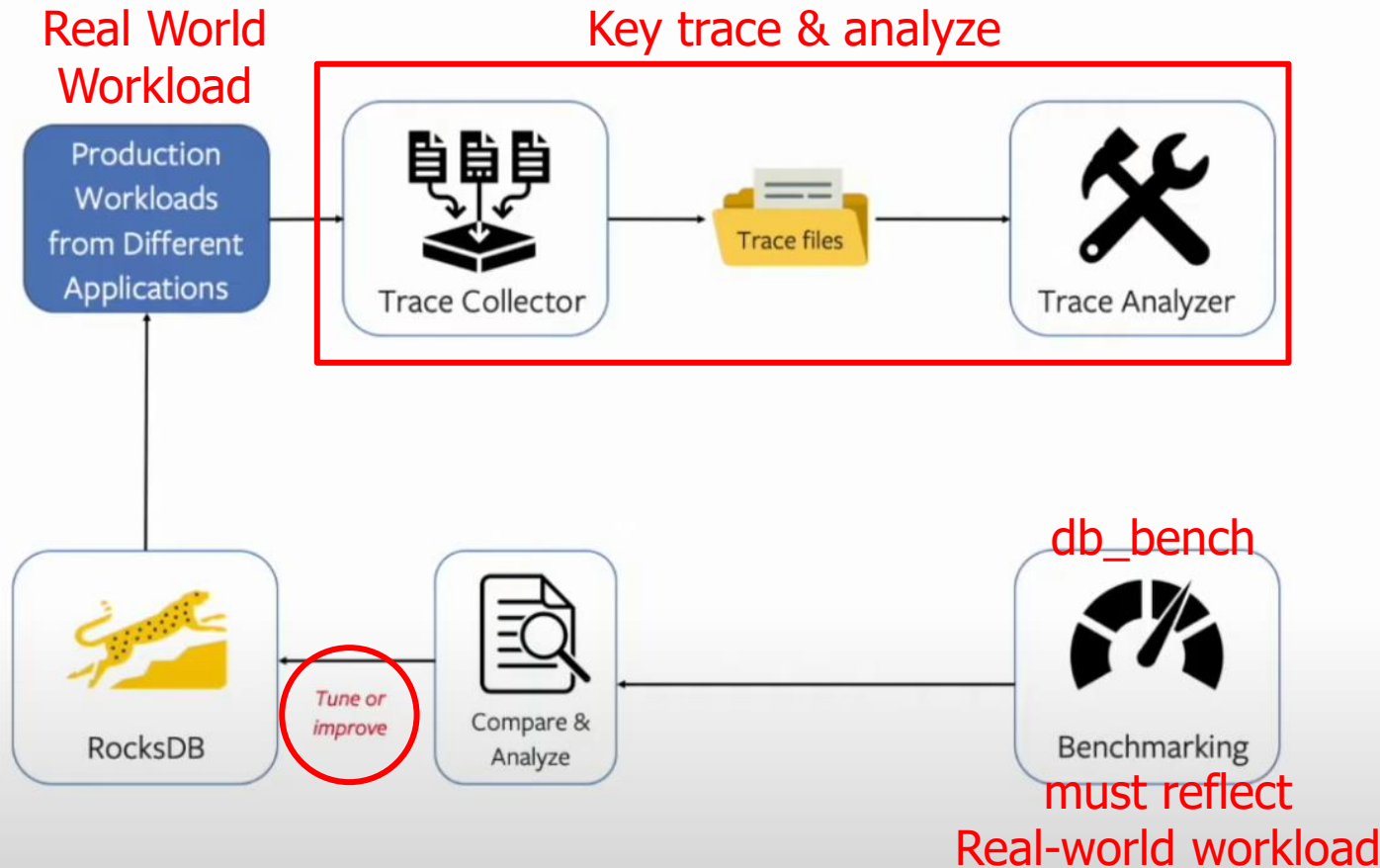
### Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook

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

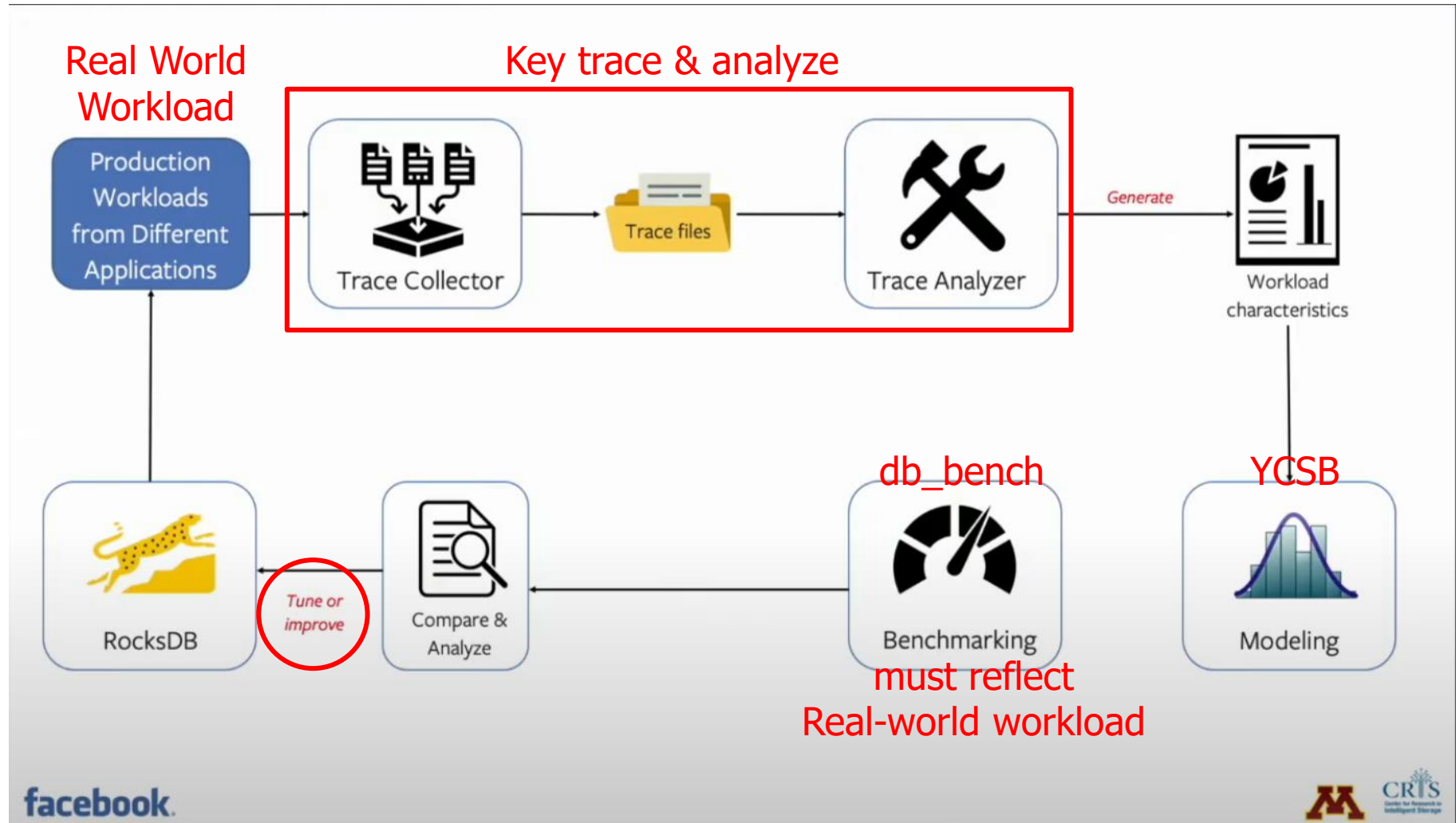
### Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook

Zhichao Cao, University of Minnesota, Twin Cities, and Facebook;

Siying Dong and Sagar Vemuri, Facebook;

David H.C. Du, University of Minnesota, Twin Cities

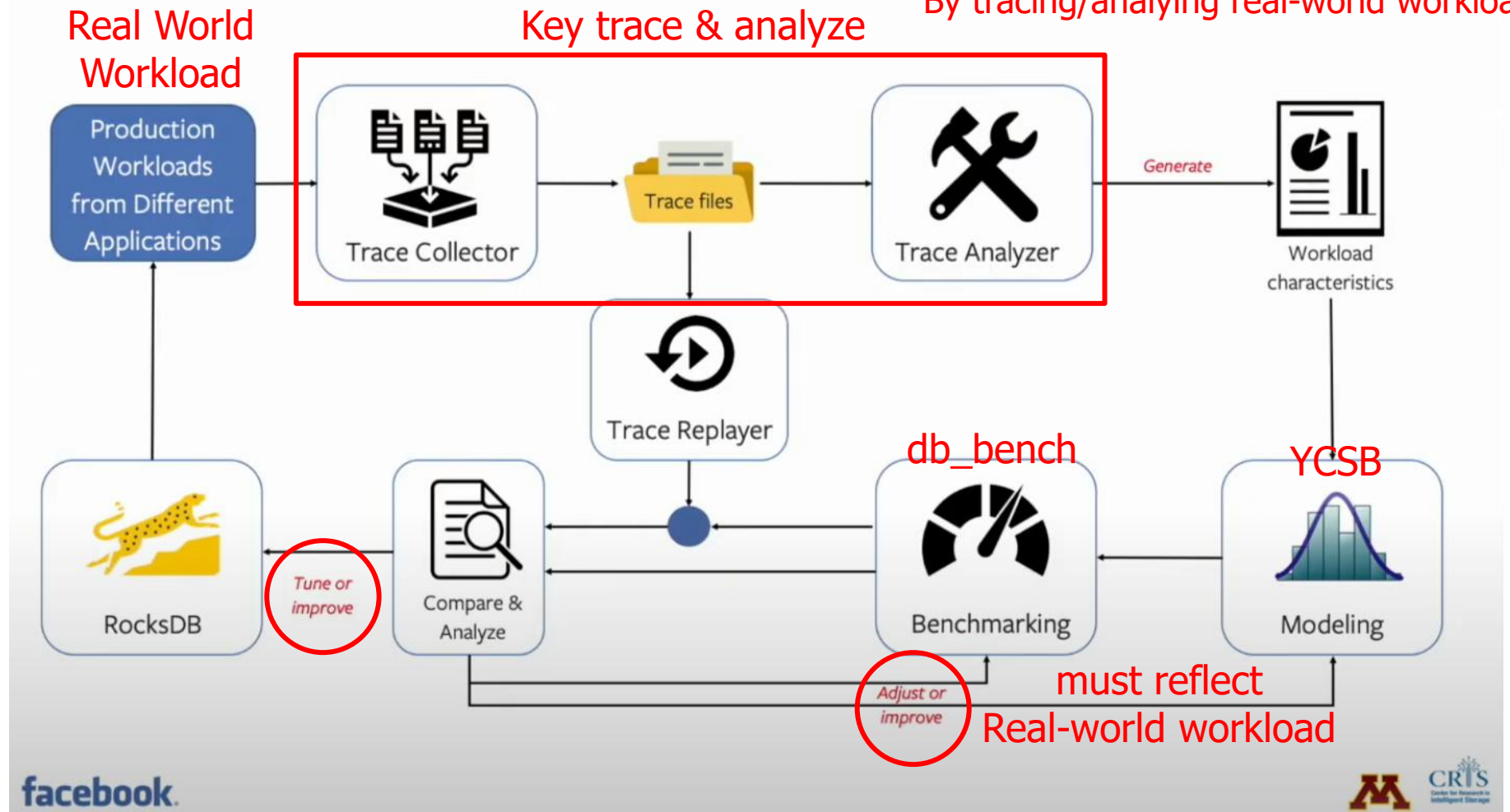
<https://www.usenix.org/conference/fast20/presentation/cao-zhichao>



## 4. Goal

### ■ 1. Latest research trends: Key Trace

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

---

