

Workload Diversity with RocksDB

Siying Dong, Database Engineering Team, Facebook

Oct 9, 2017

Key-Value Stores are Popular

W

Key-value database - Wikipedia

Secure | https://en.wikipedia.org/wiki/Key-value_database

om.

KV – eventually consistent [edit]

- Dynamo
- Cassandra
- Oracle NoSQL Database
- Project Voldemort
- Riak^[3]
- OpenLink Virtuoso

KV – ordered [edit]

- Berkeley DB
- FairCom c-treeACE/c-treeRTG
- FoundationDB
- HyperDex
- IBM Informix C-ISAM
- InfinityDB
- LMDB
- MemcacheDB

KV – RAM [edit]

- Aerospike
- Apache Ignite
- Coherence
- FairCom c-treeACE
- GridGain Systems
- Hazelcast
- memcached
- OpenLink Virtuoso
- Redis
- XAP

KV – solid-state drive or rotating disk [edit]

- Aerospike
- CDB
- Clusterpoint Database Server
- Couchbase Server
- FairCom c-treeACE
- GT.M^[4]
- Hibari
- Keyspace
- LevelDB
- LMDB
- MemcacheDB (using Berkeley DB or LMDB)
- NoSQLz
- Coherence
- Oracle NoSQL Database
- quasardb
- RocksDB (fork of LevelDB)
- OpenLink Virtuoso
- Tarantool
- Tokyo Cabinet and Kyoto Cabinet
- Tuple space

References [edit]

↑ <http://db-engines.com/en/ranking>

In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP'11)
This version is reformatted from the official version that appears in the conference proceedings.

SILT: A Memory-Efficient, High-Performance Key-Value Store

Hyeontaek Lim¹, Bin Fan¹, David G. Andersen¹, Michael Kaminsky²

¹Carnegie Mellon University, ²Intel Labs

	2008 → 2011	Increase
	731 → 1,170 M	60 %
	0.062 → 0.153 GB/s	147 %
	0.134 → 0.428 GB/s	219 %
	4.92 → 15.1 GB/s	207 %

to 2011, flash and hard disk capacity
r than either CPU transistor count or

An Efficient Design and Implementation of LSM-Tree based Key-Value Store on Open-Channel SSD

Peng Wang Guangyu Sun

Peking University

{wang-peng, gsun}@pku.edu.cn

Song Jiang *

Peking University and
Wayne State University

Jian Ouyang *

Baidu Inc

{ouyangjian, linshidi

HyperDex: A Distributed, Searchable Key-Value Store

Robert Escriva

Computer Science
Department
Cornell University

escriva@cs.cornell.edu

Bernard Wong

Charlton School of Computer

Emin Gün Sirer

Computer Science

FlashStore: High Throughput Persistent Key-Value Store

Biplob Debnath^{*}

University of Minnesota
Twin Cities, USA

biplob@umn.edu

Sudipta Sengupta

Microsoft Research
Redmond, USA

sudipta@microsoft.com

Jin Li

Microsoft Research

jli@

ABSTRACT

We present FlashStore, a high throughput persistent key-value store, that uses flash memory as a non-volatile cache between RAM and hard disk. FlashStore is designed to store the working set of key-value pairs on flash and use one flash read per key lookup. As the working set changes over time, space is made for the current working set by destaging recently unused key-value pairs to hard disk and recycling pages in the flash store. FlashStore organizes key-value pairs in a log-structure on flash to exploit faster sequential write performance. It uses an in-memory hash table to index them, with hash collisions resolved by a variant of cuckoo hashing. The in-memory hash table stores compact key signatures instead of full keys so as to strike tradeoffs between RAM usage and false flash read operations. FlashStore can be used as a high throughput persistent key-value storage layer for a broad range of server class applications. We compare FlashStore with BerkeleyDB, an embedded key-value store application, running on hard disk and flash separately, so as to bring out the performance gain of FlashStore in not only using flash as a cache above hard disk but also in its use of flash aware algorithms. We

NVMKV: A Scalable, Lightweight, FTL-aware Key-Value Store

Leonardo Marmol[†], Swaminathan Sundararaman[†], Nisha Talagala[†], Raju Rangaswami[‡]

[†]SanDisk [‡]Florida International University

Abstract

Key-value stores are ubiquitous in high performance data-intensive, scale-out, and NoSQL environments. flash devices for meeting their per- sever, by using flash as a sim- ple KV stores are unable to fully capabilities that exist within Flash (FTLs). NVMKV is a lightweight es native FTL capabilities such as namic mapping, transactional per- for high-levels of lock free paral- n of NVMKV demonstrates that it h-performance, and ACID compli- close to raw device speeds.

is able to achieve single I/O *get/put* operations with performance close to that of the raw device, representing a significant improvement over current KV stores. NVMKV uses the advanced FTL capabilities of *atomic multi-block write*, *atomic multi-block persistent trim*, *exists*, and *iterate* to provide strictly atomic and synchronous durability guarantees for KV operations. Two complementary factors contribute to increased collocation requirements for KV stores running on a single flash device. First, given the increasing flash densities, the performance points of flash devices are now based on capacity with larger devices being more cost-effective [42]. Second, virtualization supports increases in collocation requirements for workloads. A recent

Cache Craftiness for Fast Multicore Key-Value Storage

Yandong Mao, Eddie Kohler[‡], Robert Morris

MIT CSAIL, [†]Harvard University

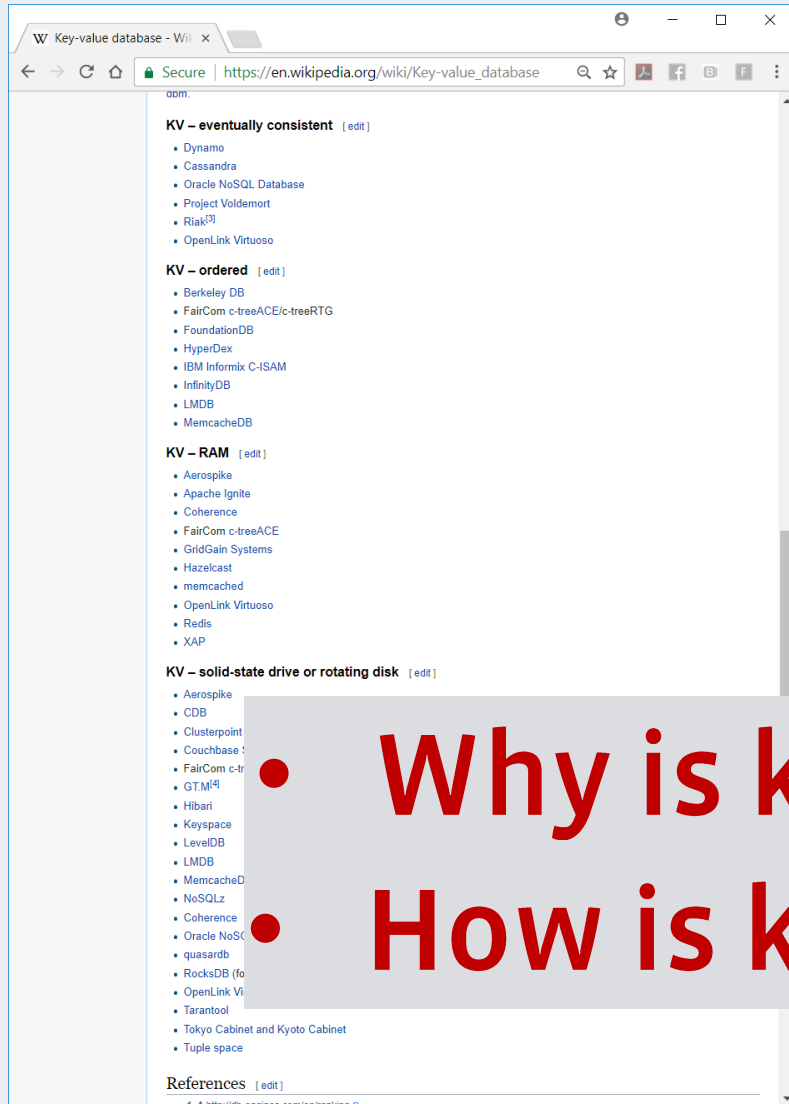
Abstract

We present Mastree, a fast key-value database designed for SMP machines. Mastree keeps all data in memory. Its main data structure is a trie-like concatenation of B⁺-trees, each of which handles a fixed-length slice of a variable-length key. This structure effectively handles arbitrary-length possi-

sufficiently fast single servers. A common route to high performance is to use different specialized storage systems for different workloads [4]. This paper presents Mastree, a storage system specialized for key-value data in which all data fits in memory, but must persist across server restarts. Within these constraints,

<https://creativecommons.org/licenses/by-sa/3.0/>

Key-Value Stores are Popular



- Why is key-value store popular?
- How is key-value store used?

SILT: A Memory-Efficient, High-Performance Key-Value Store

Hyeontaek Lim¹, Bin Fan¹, David G. Andersen¹, Michael Kaminsky²

¹Carnegie Mellon University, ²Intel Labs

2008 → 2011	Increase
731 → 1,170 M	60 %
0.062 → 0.153 GB/s	147 %
0.134 → 0.428 GB/s	219 %
4.92 → 15.1 GB/s	207 %

to 2011, flash and hard disk capacity
r than either CPU transistor count or

An Efficient Design and Implementation of LSM-Tree based Key-Value Store on Open-Channel SSD

Peng Wang Guangyu Sun
Peking University
{wang-peng, gsun}@pku.edu.cn

Song Jiang *
Peking University and
Wayne State University

Jian Ouyang
Baidu Inc.
{ouyangjian, linshidi

HyperDex: A Distributed, Searchable Key-Value Store

Robert Escriva
Computer Science
Department
Cornell University
escriva@cs.cornell.edu

Bernard Wong
Charlton School of Computer

Emin Gün Sirer
Computer Science

NVMKV: A Scalable, Lightweight, FTL-aware Key-Value Store

Leonardo Marmol[†], Swaminathan Sundararaman[†], Nisha Talagala[†], Raju Rangaswami[‡]
[†]SanDisk [‡]Florida International University

Abstract

Key-value stores are ubiquitous in high performance
data-intensive, scale
out, and NoSQL environments.
flash devices for meeting their per-
sewer, by using flash as a sim-
se KV stores are unable to fully
capabilities that exist within flash
FTLs). NVMKV is a lightweight
cooperative FTL capabilities such as

is able to achieve single I/O get/put operations with
performance close to that of the raw device, represent-
ing a significant improvement over current KV stores.
NVMKV uses the advanced FTL capabilities of *atomic
multi-block write, atomic multi-block persistent trim,
exists, and iterate* to provide strictly atomic and syn-
chronous durability guarantees for KV operations.

Two complementary factors contribute to increased
collaboration environments for KV stores running on a sin-
First, given the increasing flash den-
dence points of flash devices are now
y with larger devices being more cost-
second, virtualization supports increases
requirements for workloads. A recent

Key-Value Storage

Morris

Abstract

We present Masstree, a fast key-value database designed for
SMP machines. Masstree keeps all data in memory. Its main
data structure is a trie-like concatenation of B⁺-trees, each of
which handles a fixed-length slice of a variable-length key.
This structure effectively handles arbitrary-length possibi-

sufficiently fast single servers. A common route to high per-
formance is to use different specialized storage systems for
different workloads [4].

This paper presents Masstree, a storage system special-
ized for key-value data in which all data fits in memory, but
must persist across server restarts. Within these constraints,

get/put operations.

Categories and Subject Descriptions
D.4.7 [Operating Systems]: Organiza

Keywords

Key-Value Store, NoSQL, Fault-Toleran-
tenacy, Performance

hashing. The in-memory hash table stores compact key sig-
natures instead of full keys so as to strike tradeoffs between
RAM usage and false flash read operations.

FlashStore can be used as a high throughput persistent
key-value storage layer for a broad range of server class ap-
plications. We compare FlashStore with BerkeleyDB, an
embedded key-value store application, running on hard disk
and flash separately, so as to bring out the performance
gain of FlashStore in not only using flash as a cache above
hard disk but also in its use of flash aware algorithms. We

DRAM, with cost decrease
characteristics have led to
sumer electronic devices, s
cameras.

However, it is only recent
of Solid State Drives (SS
tion in desktop and server
pace.com recently switch
its servers to using PCI E



Key-Value Storage on Log-Structure Merge-Tree

RocksDB is versatile

RocksDB Application Diversity

- Inside Facebook:
 - **MyRocks**: MySQL Engine
 - **ZippyDB**: distributed Key-value store
 - **Laser**: data publishing service
 - **Dragon**: distributed graph query engine
 - **LogDevice**: distributed data store for logs
 - **Stylus**: stream processing framework

Workload Diversity

RocksDB Workload Diversity

- 1** Document Store
- 2** Social Graph Edges
- 3** Time-Ordered Events
- 4** Counter Service
- 5** Storage For Logs
- 6** Cache

RocksDB Workload Diversity

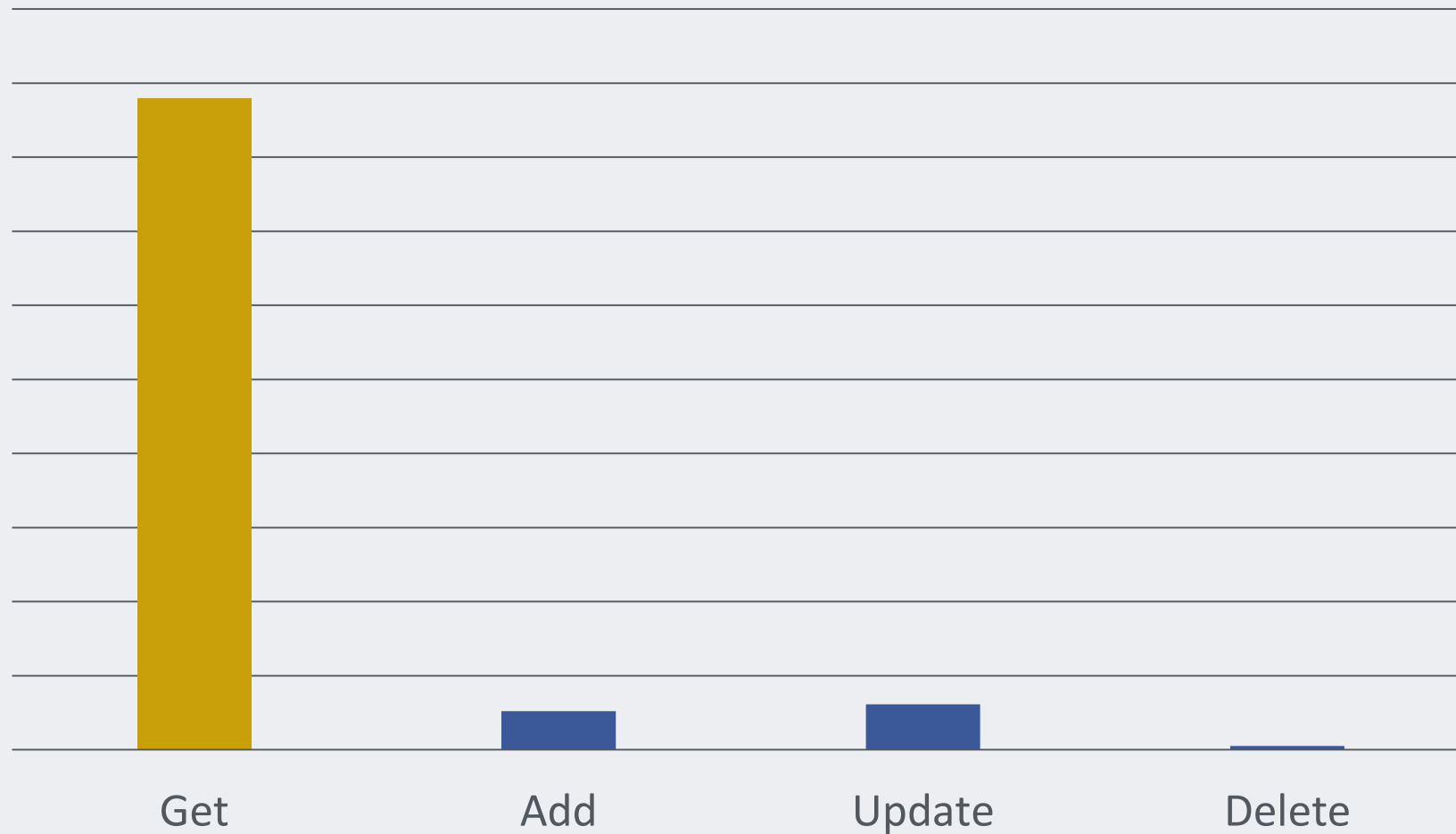
- 1** Document Store
- 2** Social Graph Edges
- 3** Time-Ordered Events
- 4** Counter Service
- 5** Storage For Logs
- 6** Cache

Document Store Example: Tao object

- TAO: Facebook's Distributed Data Store for the Social Graph
- Sample object:

```
'timestamp' => 1505514049,  
'author' => 100008855205466,  
'message_type' => 308,  
'body' => 'Performance results for load, create index and table scan for .....',  
'gid' => 530977083778847,  
'lang' => 'en',  
'creator_id' => 100008855205466,  
'detected_dialect' => 'en_XX',  
'detected_dialect_confidence' => 100,  
'modified_time' => 1505514292,  
.....
```

Tao Object: Operation Ratio



Tao Object: Object Size

	Number of Bytes
Mean	168
Median	78
P75	246
P90	441
P95	733
P99	1688

Document encoding

- Customized Format
- C++ struct
- Thrift (RPC Protocol) struct
- JSON/BSON

Workload

- Point lookup only
- Balance Read/write.

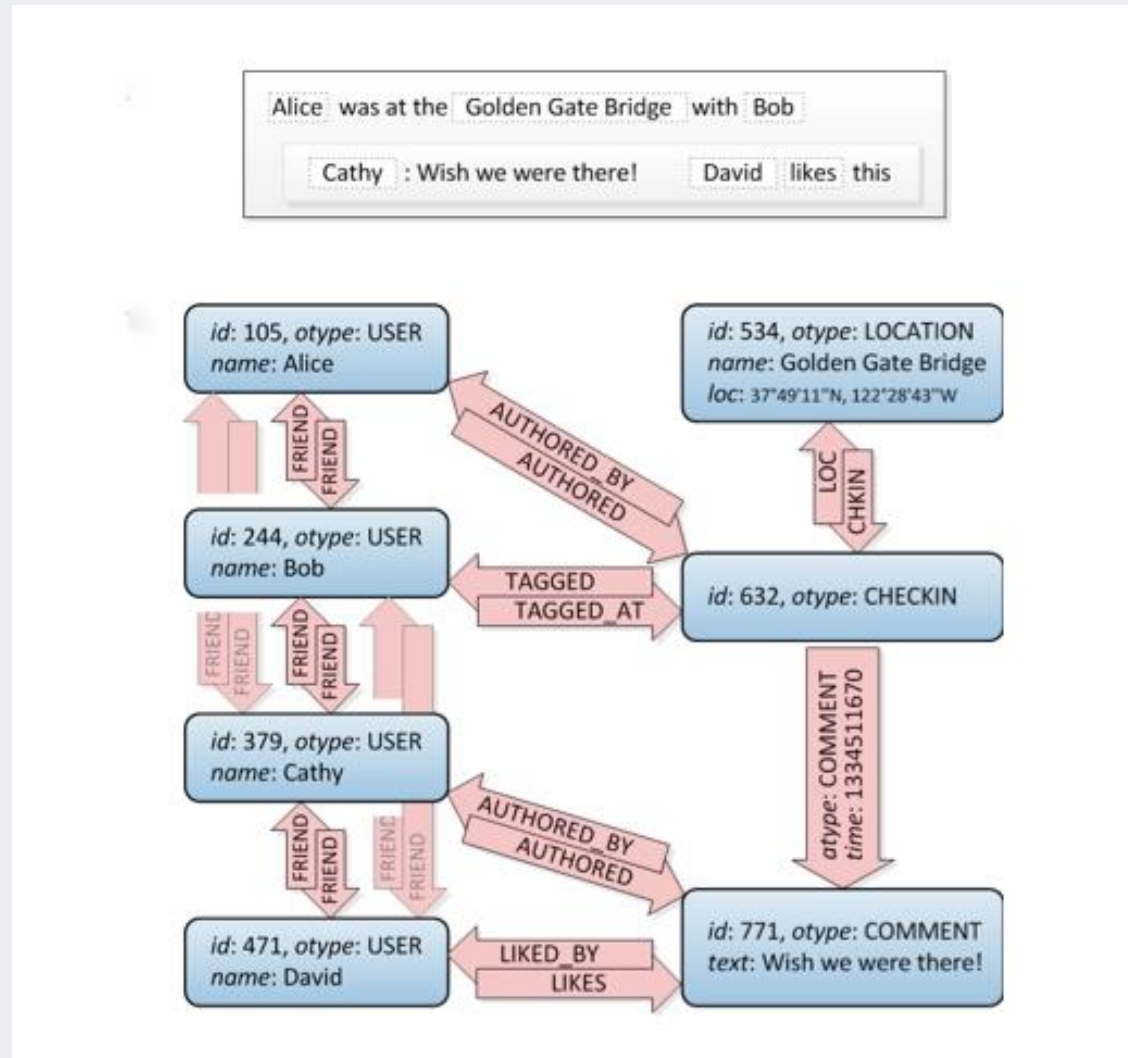
Update a small subset of attributes?

- Read-modify-write: read optimized
- Blindly write delta and merge in read: write optimized

RocksDB Workload Diversity

- 1** Document Store
- 2** Social Graph Edges
- 3** Time-Ordered Events
- 4** Counter Service
- 5** Storage For Logs
- 6** Cache

Example: Tao “associations”



FB page => admin

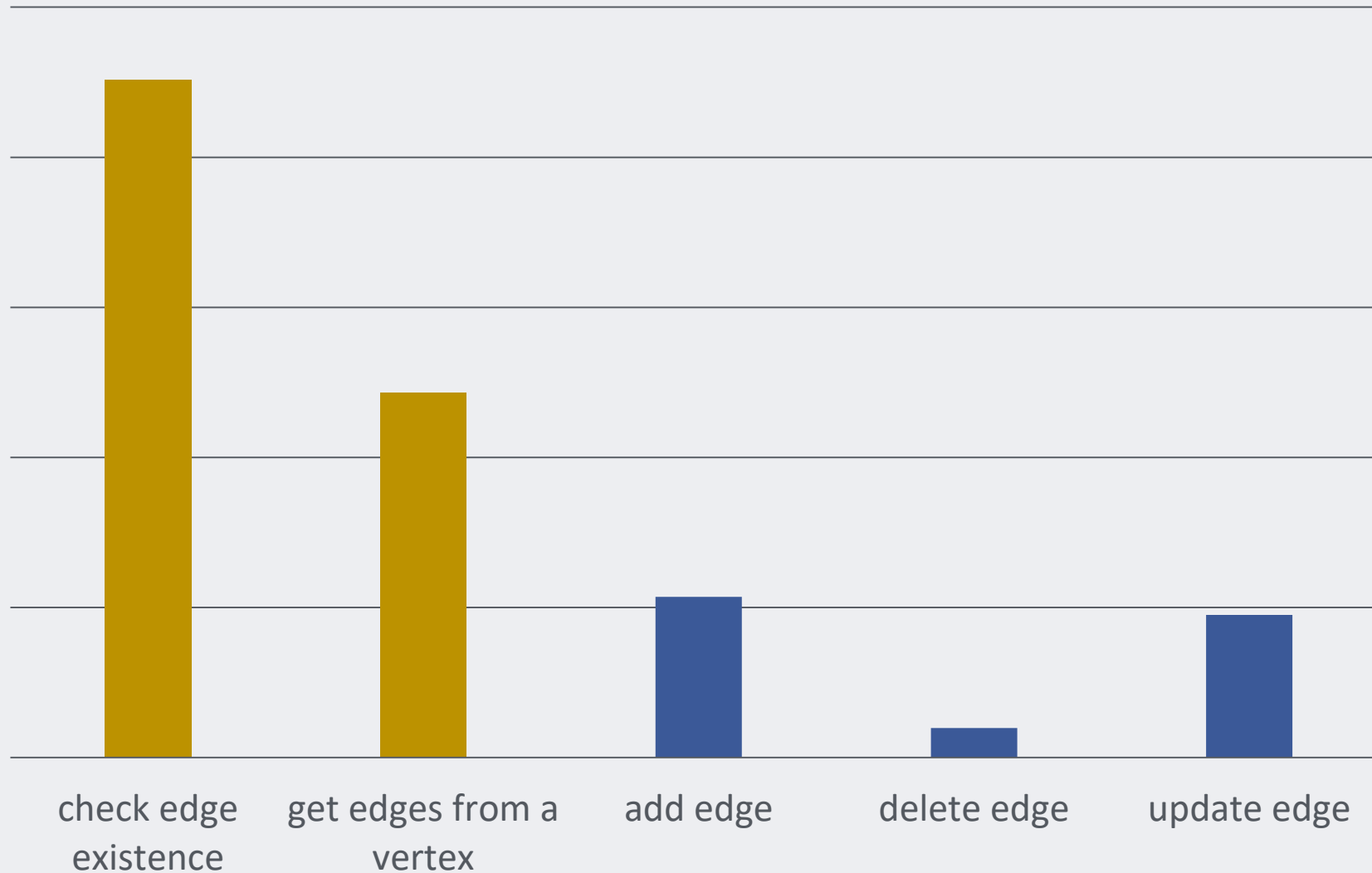
- Key: [*page_id*, *user_id*]
- Value: [*update_time*, *is_deleted*...]
- Common Query: find admins of a page of *page_id*:
 - Scan keys in the range prefixed with *page_id*

Number of Edges Per Vertex

	Number of Edges
Mean	3.85
Median	1
P75	1
P95	3
P99	22

Optimization: bloom filter using range prefix

Tao associations: Operation Ratio



Ranked Comments of a post

- Key: [*post_id*, *language_id*, *ranking_score*, *comment_id*]
- Value: (empty)
- Common Query: find top ranked comments of *post_id*:
 - Scan the first N keys between
[*post_id*, *viewer_language*, *min_score*] to [... *max_score*]

RocksDB Workload Diversity

- 1** Document Store
- 2** Social Graph Edges
- 3** Time-Ordered Events
- 4** Counter Service
- 5** Storage For Logs
- 6** Cache

Example: Recent Time-Ordered Events

- Key: $[user_id, event_timestamp]$ with *timestamp* in reverse order
- Value: some metadata of the event.
- Read Query: range scan between $[user_id, max_ts]$ to $[user_id, min_ts]$ limit N

Workload of the example

- Write/Read 63:1
- Average keys per range query: ~1,100
- Average value size per key: 230 bytes

Retire Old Time-Ordered Events

- Time-Ordered Events should retire if
 - It is too old
 - Too many Time-Ordered Events from a user
- Solution: compaction filter

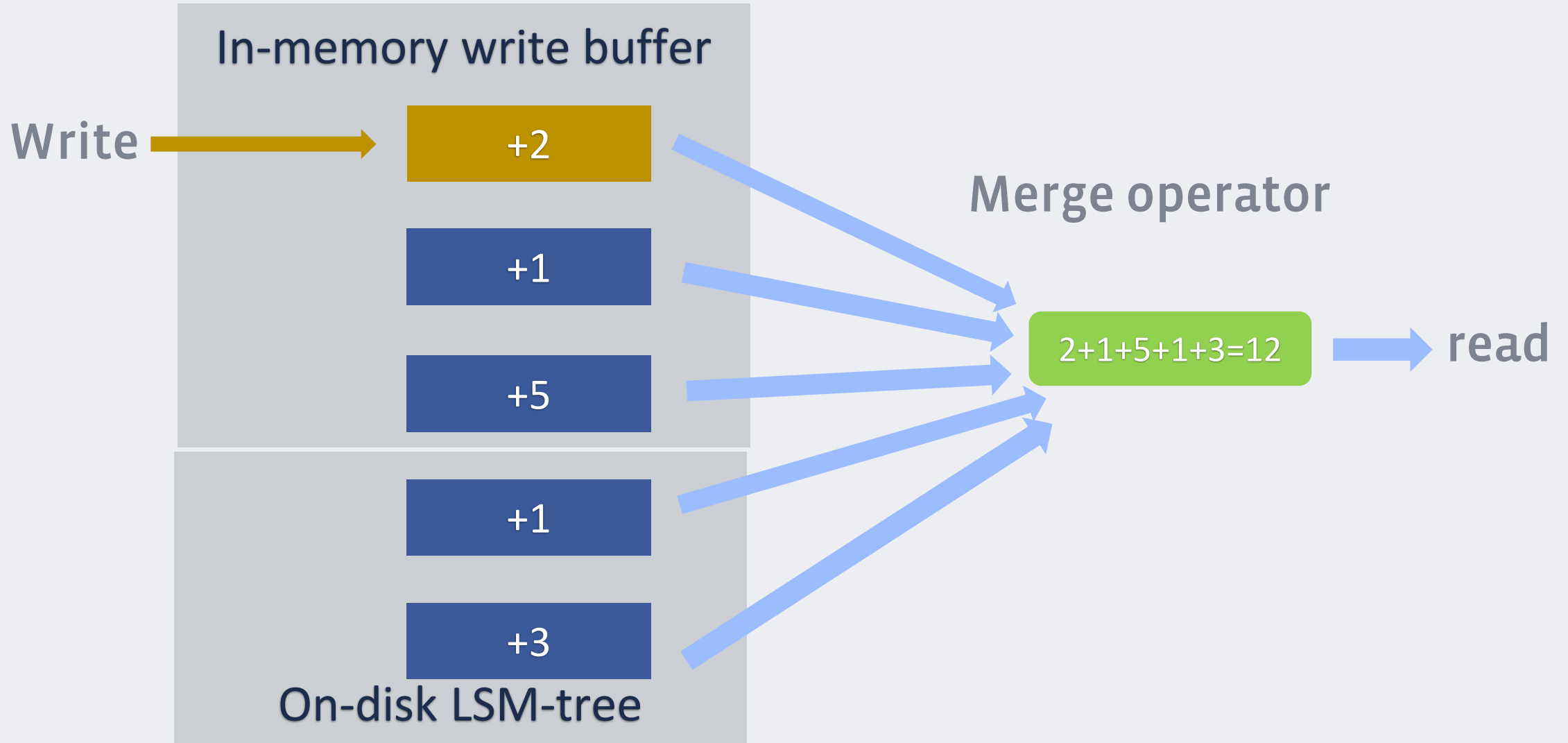
RocksDB Workload Diversity

- 1** Document Store
- 2** Social Graph Edges
- 3** Time-Ordered Events
- 4** Counter Service
- 5** Storage For Logs
- 6** Cache

Counter Service

- Key: counter_id
- Value: the counter. Common counter types:
 - Plain count
 - Unique count (using hyperloglog)
 - Auto-decaying Counter Service
- Usually update heavily

Delta Updates (merge operator)



Workload in one use case

- 180K key updates + 19K keys read per second per host
- Average key-update size 26 bytes
- How many delta entries to merge when read:
 - Median: 7
 - P90: 801

RocksDB Workload Diversity

- 1** Document Store
- 2** Social Graph Edges
- 3** Time-Ordered Events
- 4** Counter Service
- 5** Storage For Logs
- 6** Cache

Storage For Logs

- Key: $[logging_id, seq_id]$ where seq_id is incremental
- Value: message contents
- Common Query: range scan from $[logging_id, last_seen_seq_id]$ for all keys of $logging_id$.
- Write-heavy, most reads are against recent updates.
- Older data is deleted.

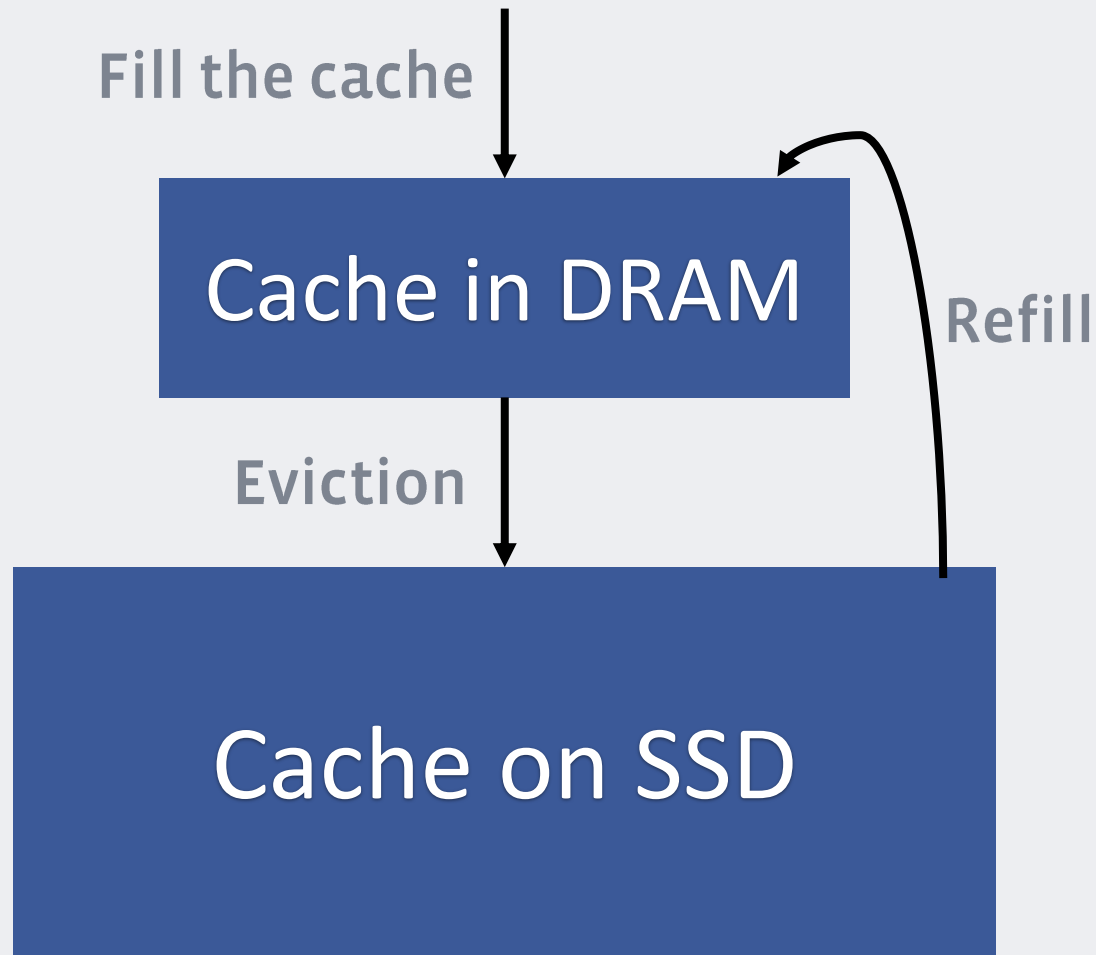
Example: A LogDevice Use Case

- For a host:
 - 260K keys inserted per second
 - 350K keys read per second
 - Every read gets 1.13 keys
 - Average log size 80 bytes

RocksDB Workload Diversity

- 1** Document Store
- 2** Social Graph Edges
- 3** Time-Ordered Events
- 4** Counter Service
- 5** Storage For Logs
- 6** Cache

RocksDB as a Cache: an example



RocksDB as a Cache: an example

- Put()/Get() only.
- 3.4K write/s, 14K read/s per host
- Hit rate about 13%
- Write heavy, sometimes read heavy too.
- Evict old data in FIFO.

	Number of Bytes
Mean	422
Median	141
P95	340
P99	4000

RocksDB as a Cache:

Some Tuning Experience

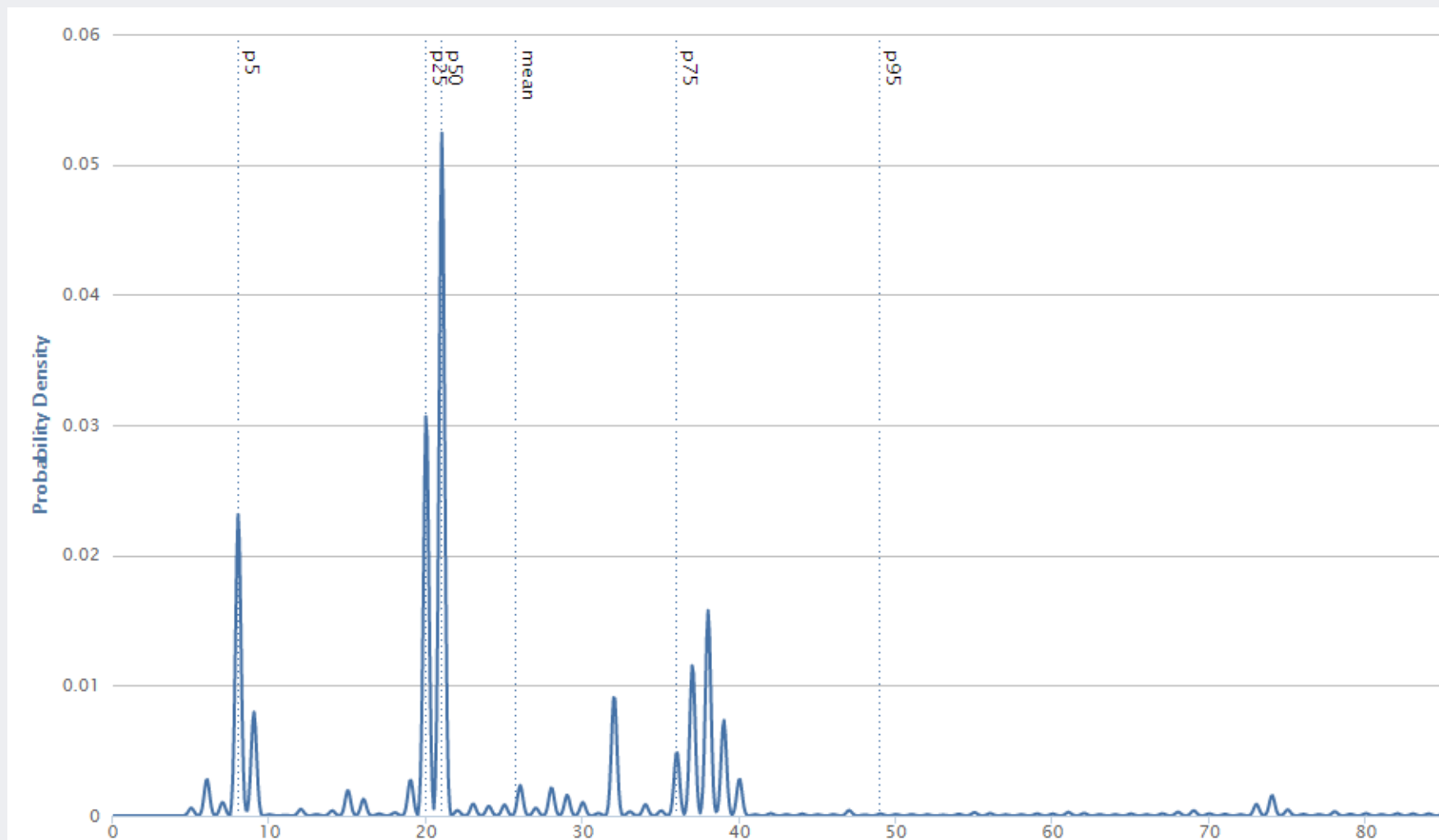
- Minimize compaction
- Be careful about bloom filter false positive rate
- Drop oldest data directly
- Key and value should be stored separately for large values.

RocksDB Workload Diversity

- 1** Document Store
- 2** Social Graph Edges
- 3** Time-Ordered Events
- 4** Counter Service
- 5** Storage For Logs
- 6** Cache

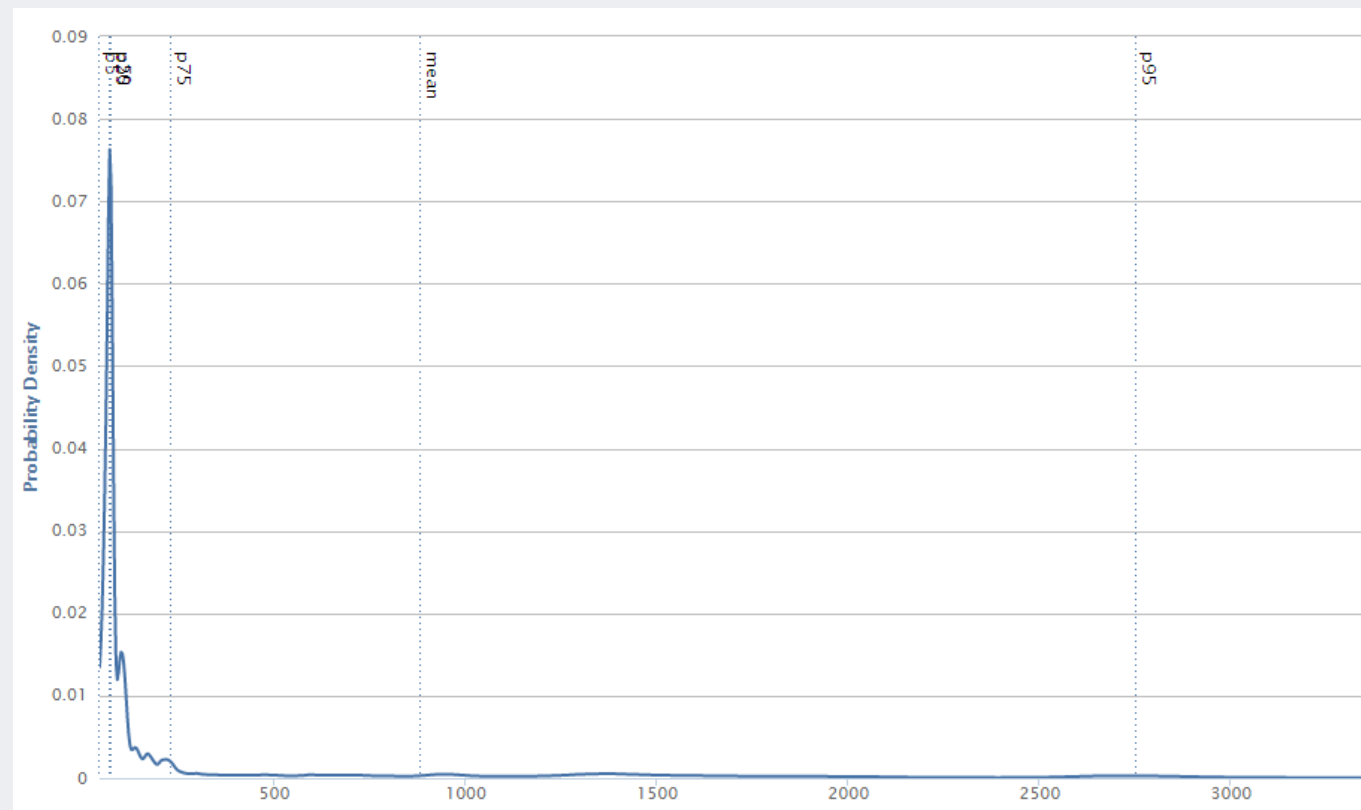
Common Workload Pattern

Key Length Distribution



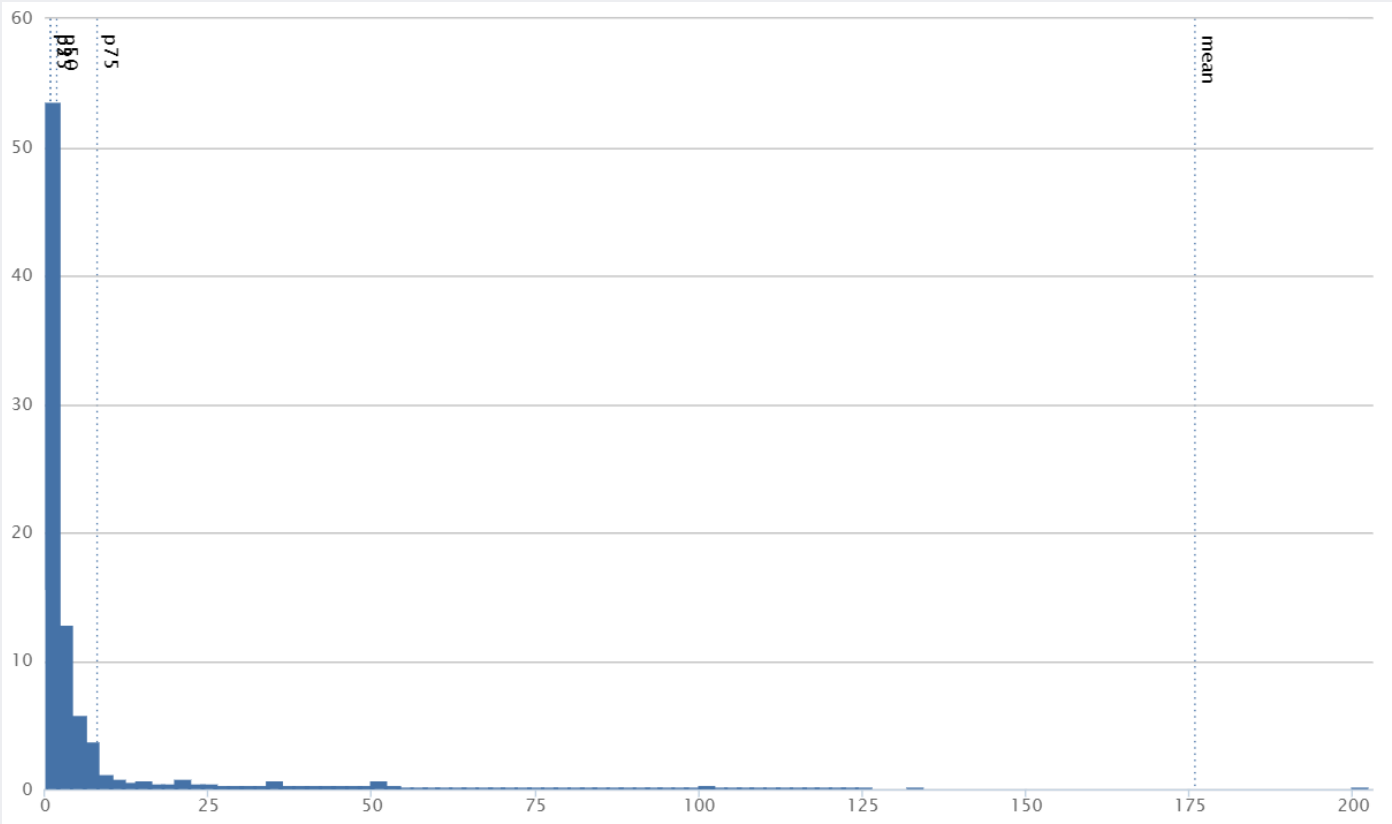
Key + Value Length Distribution

	K+V length (Bytes)
Mean	869
Median	59
P75	216
P90	1.68K
P95	2.75K
P99	12.2K



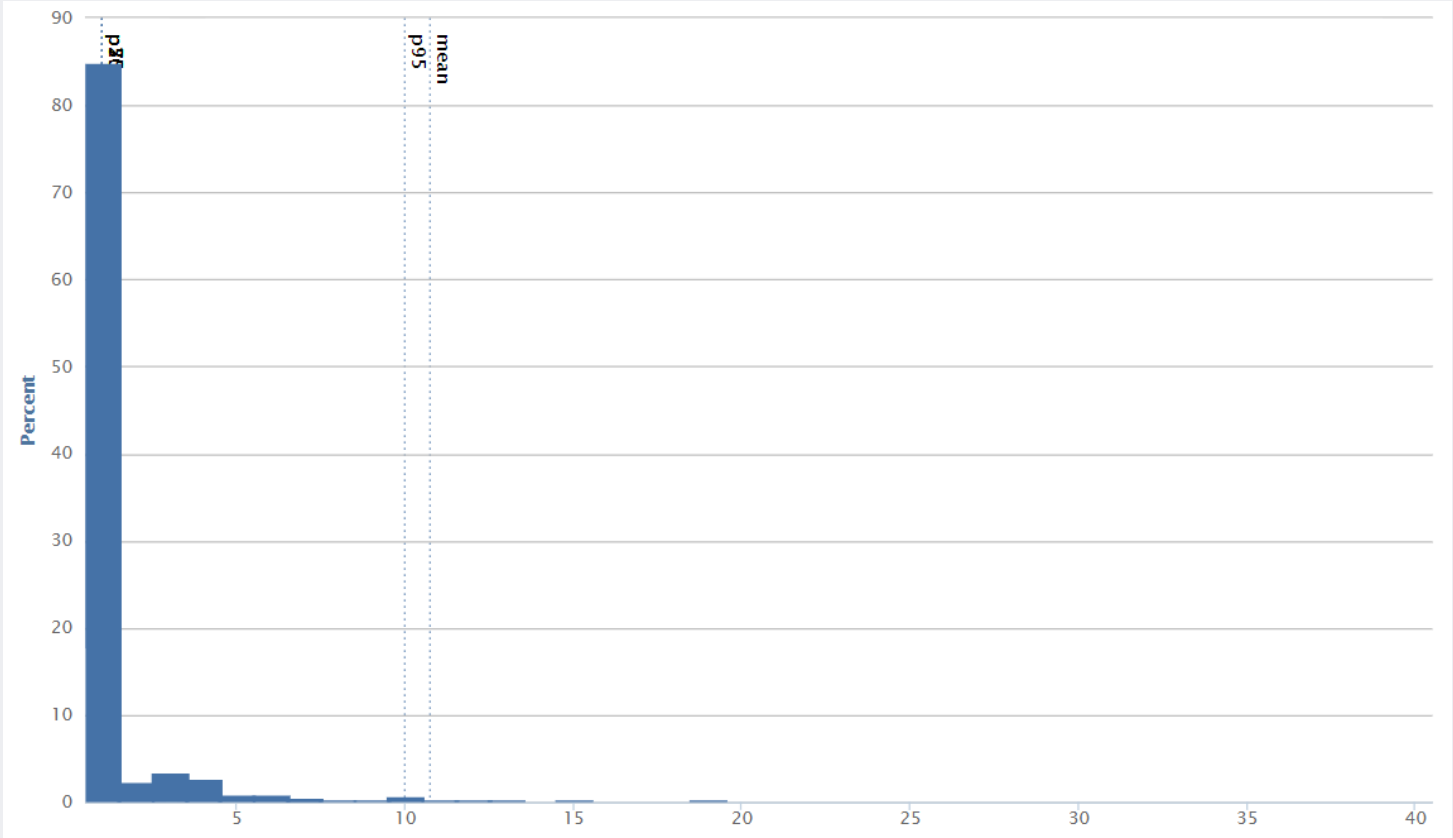
Number of Keys Per Range Query

	#keys updated per write
Mean	176
Median	2
P75	8
P95	732
P99	3.67K



Number of Keys Updated Per Writex

	#keys updated per write
Mean	10.8
Median	1
P95	10
P99	250



Take-Away

- RocksDB is versatile:
 - Diversified Applications
 - Diversified Workloads

Thank You!



Key-Value Storage on Log-Structure Merge-Tree