

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

Zhichao Cao^{1,2}, Siying Dong², Sagar Vemuri², and David H.C. Du¹

¹*University of Minnesota, Twin Cities* ²*Facebook*



Key-Value Stores are Widely Used



Key-Value Store is a Hot Research Area

- The number of key-value store related papers appearing in some of the conferences

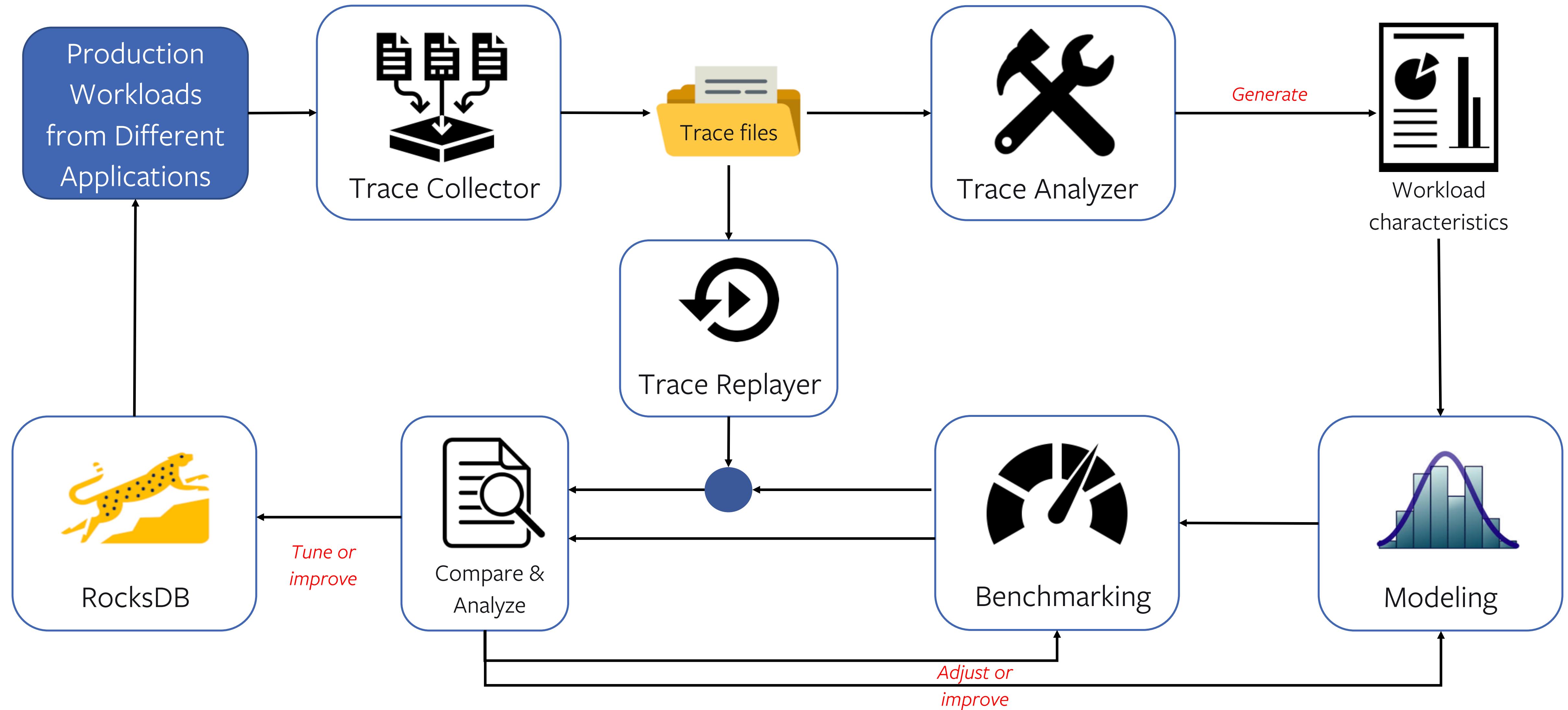
	VLDB	SIGMOD	OSDI/SOSP	FAST	ATC
2017	3	3	3	1	5
2018	3	1	2	1	4
2019	1	1	2	5	4
2020	?	?	?	4	?

However.....

- Key-value workloads in different applications are very different. But there are very limited studies of real-world workload characterization and analysis for different applications that using key-value stores.
- The analytic methods for characterizing KV-store workloads are different from the existing workload characterization. How to capture, characterize, and model the workloads of KV store in different applications are challenging?
- Based on the real-world workloads, what's the limitations of existing key-value benchmarks (e.g., query similarity or storage I/Os)? How to further improve the benchmarks?

Methodology and Tool Set

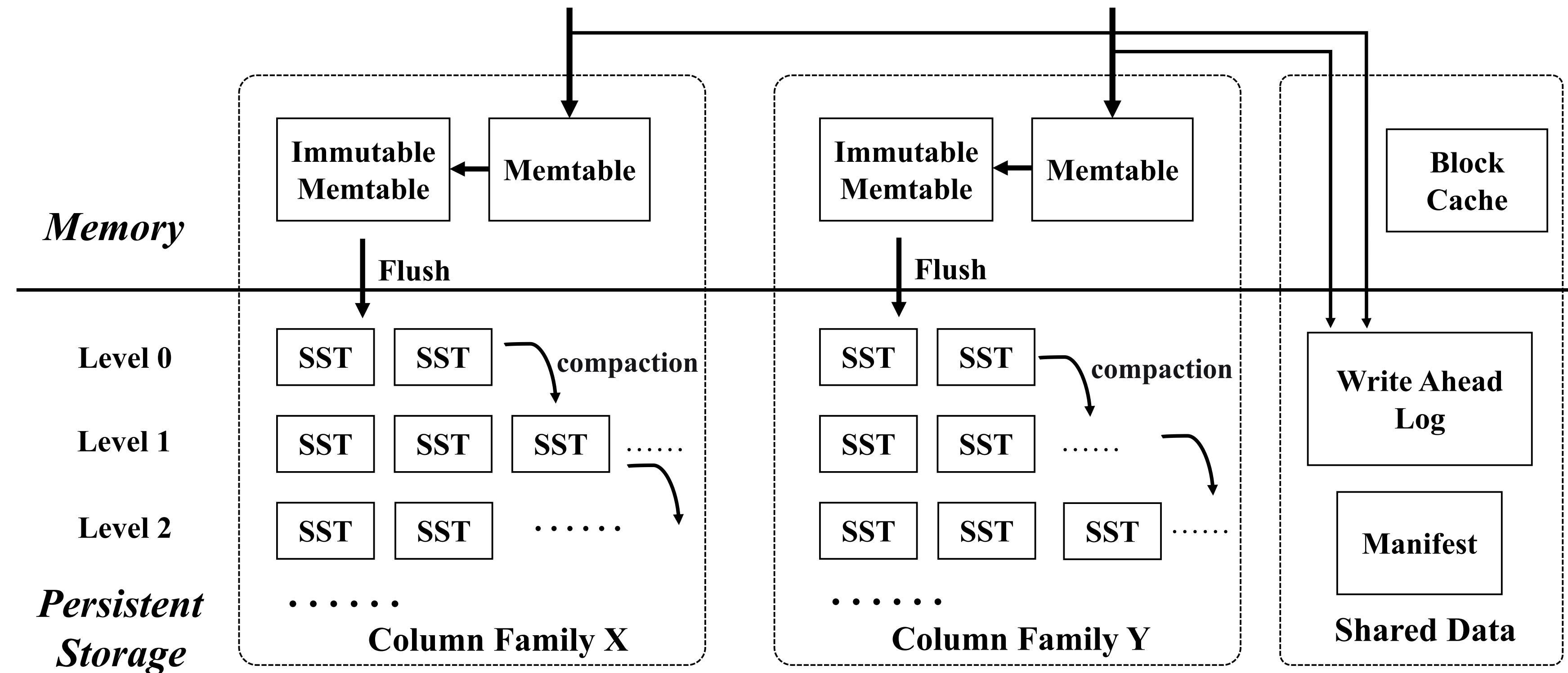
- **Methodology:** collect workloads-> analyze workloads->model the workloads
-> compare and improve benchmarks-> tune and improve key-value stores
- **Tools:** trace collector, trace replayer, trace analyzer, benchmarks



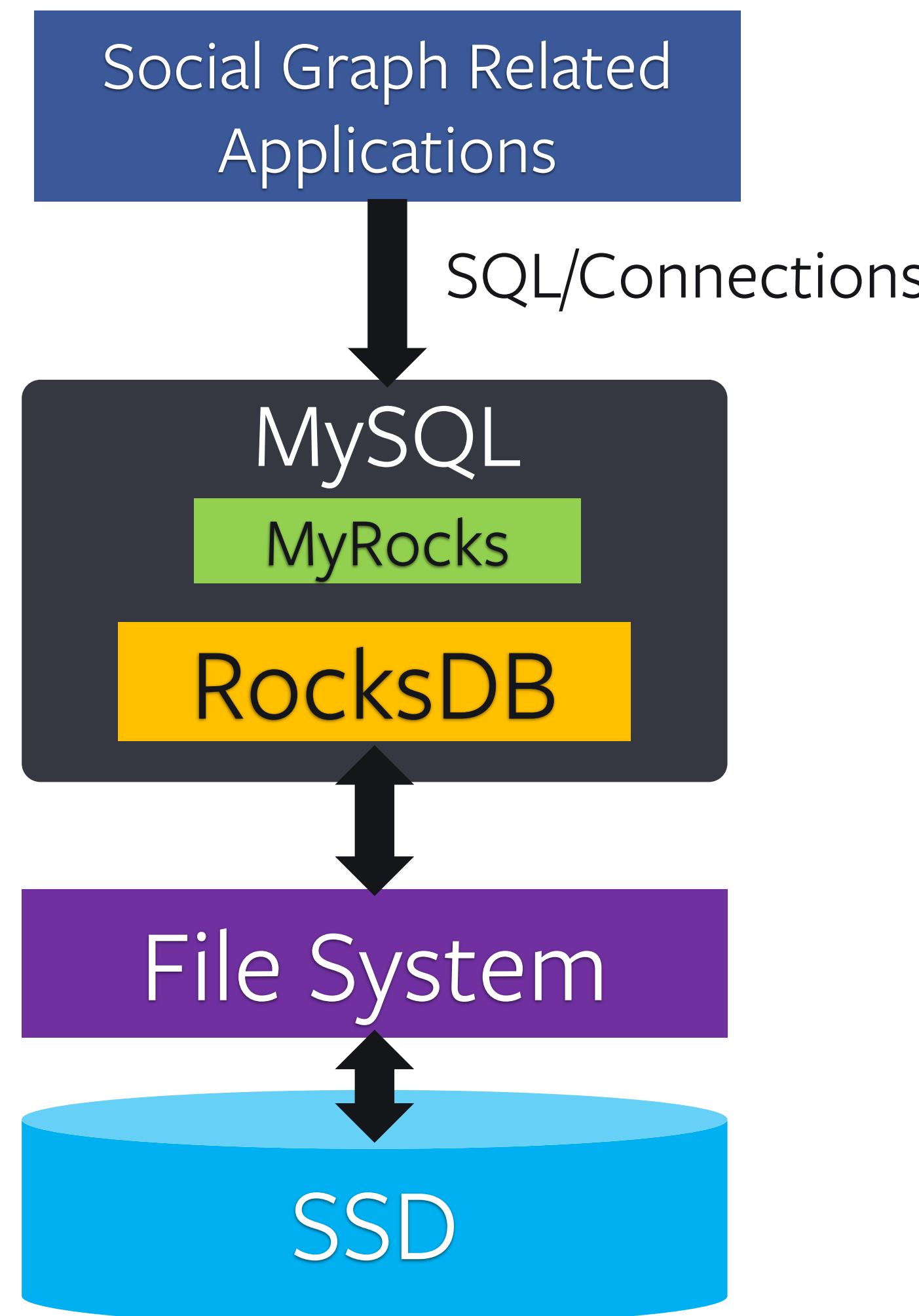
Background

- RocksDB introductions
- Three RocksDB applications: 1) UDB, 2) ZippyDB, and 3) UP2X

RocksDB Preliminary



Use Case 1: UDB



Primary key indexed raw mapping to a RocksDB KV-pair

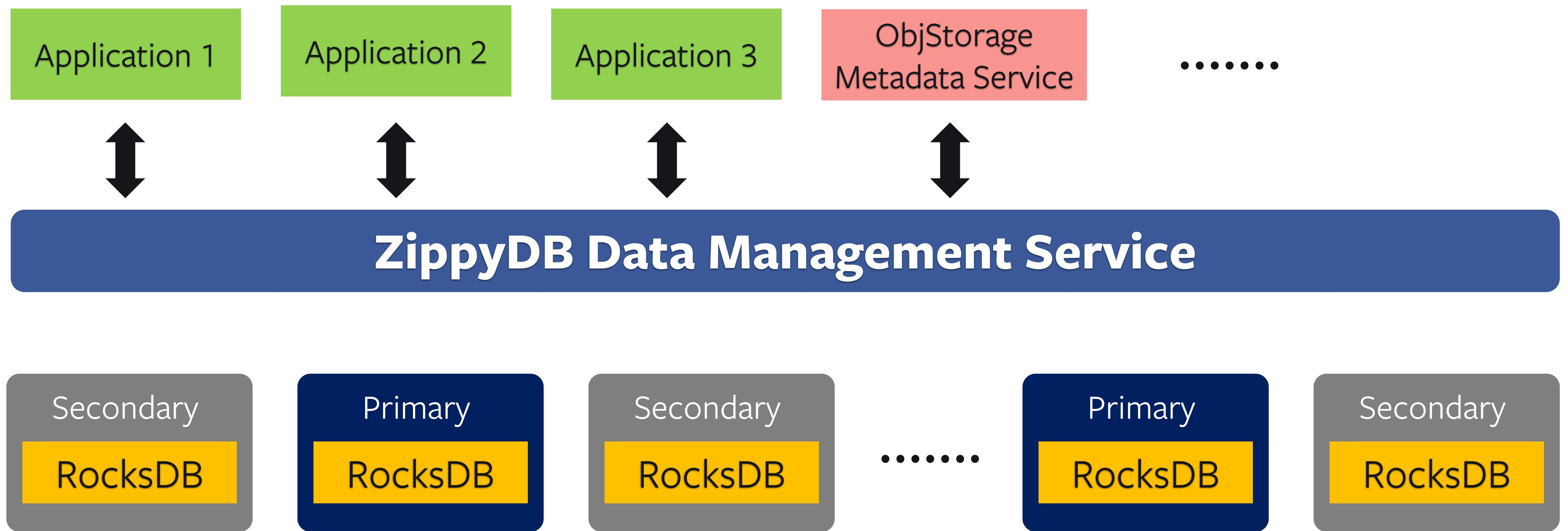
RocksDB Key	RocksDB Value		
Table Index Number	Primary Key	Columns	Checksum

Secondary key indexed raw mapping to a RocksDB KV-pair

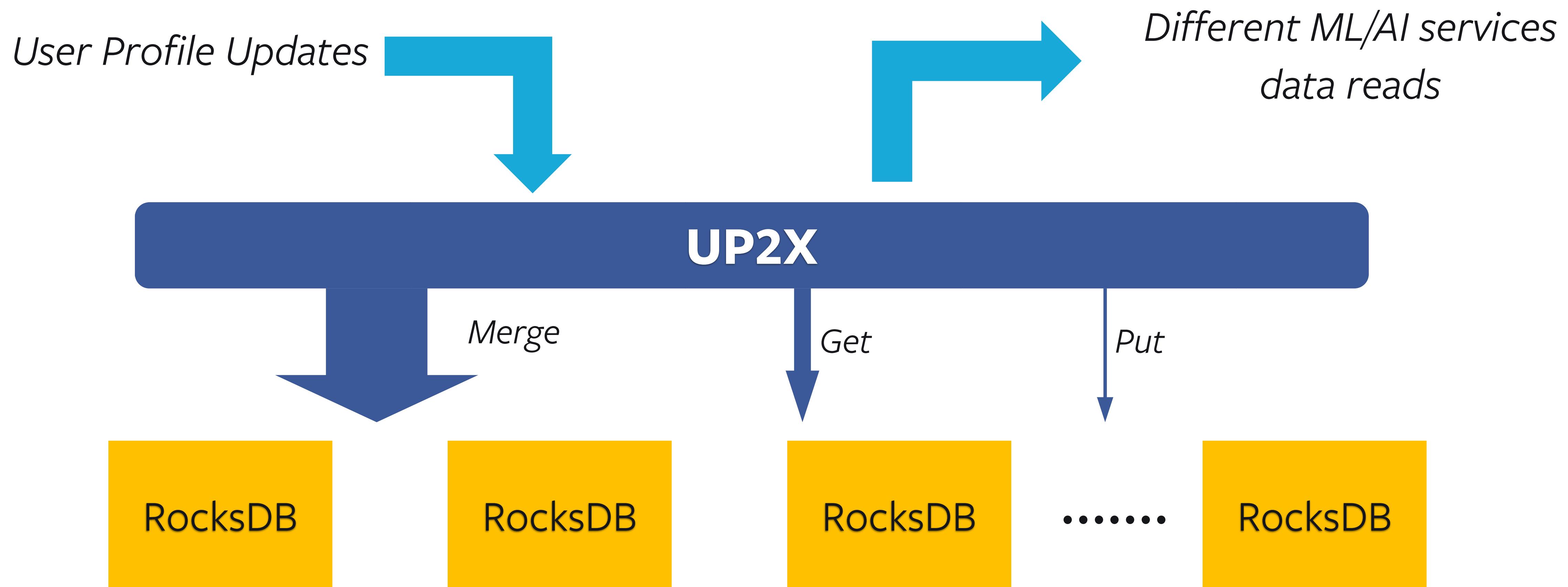
RocksDB Key	RocksDB Value		
Table Index Number	Secondary Key	Primary Key	Checksum

Graph data: Objects and associations

Use Case 2: ZippyDB



Use Case 3: UP2X

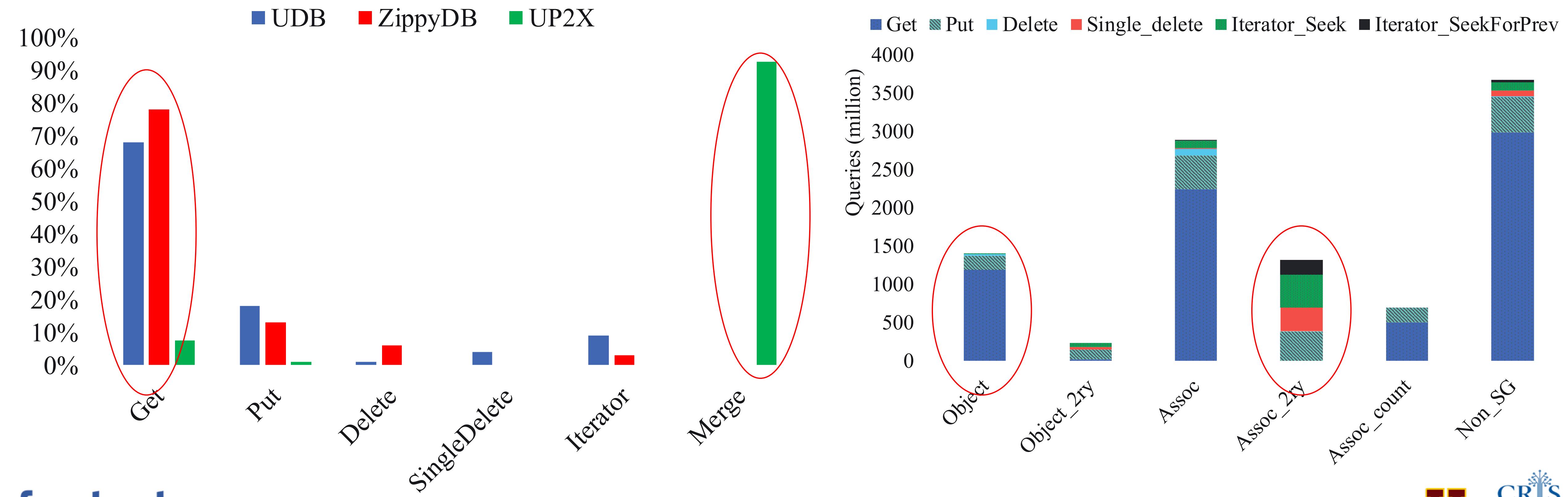


Workload Characterization for Three RocksDB Production Use cases

- 24 hours to 14 days traces.
- Query composition
- Key and value size statistics and distributions
- KV-pair hotness and access count distributions
- Query per second (QPS)
- Hot key distributions in key-space
- Key-space and temporal localities
-

Key-Value Query Composition

- Query composition can be very different in different applications and even in different CFs in the same DB.
- Get is the most frequently used query type in UDB and ZippyDB, while Merge dominates the queries in UP2X.
- Query composition can be very different in different CFs in UDB



Key and Value Size Statistics

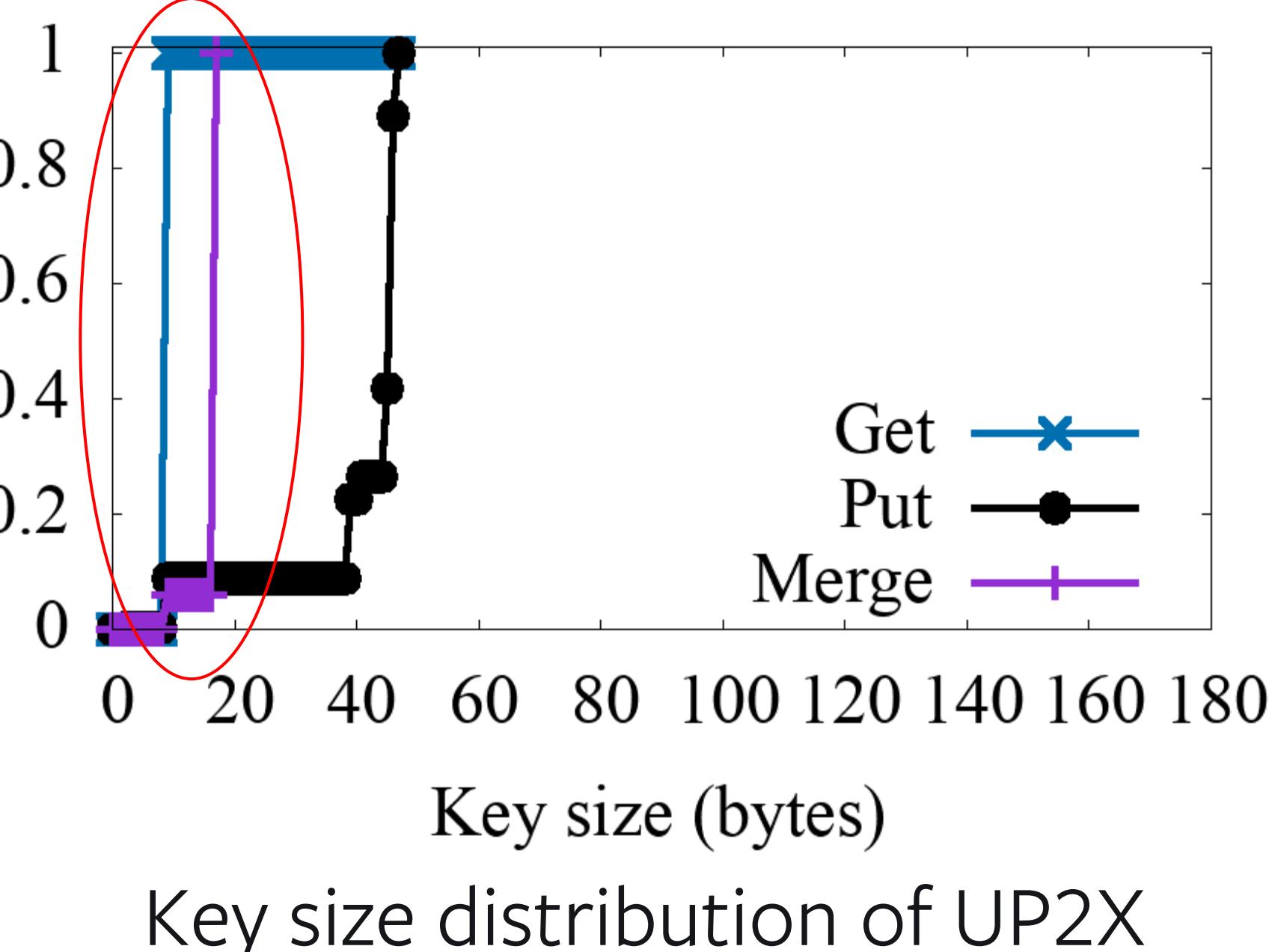
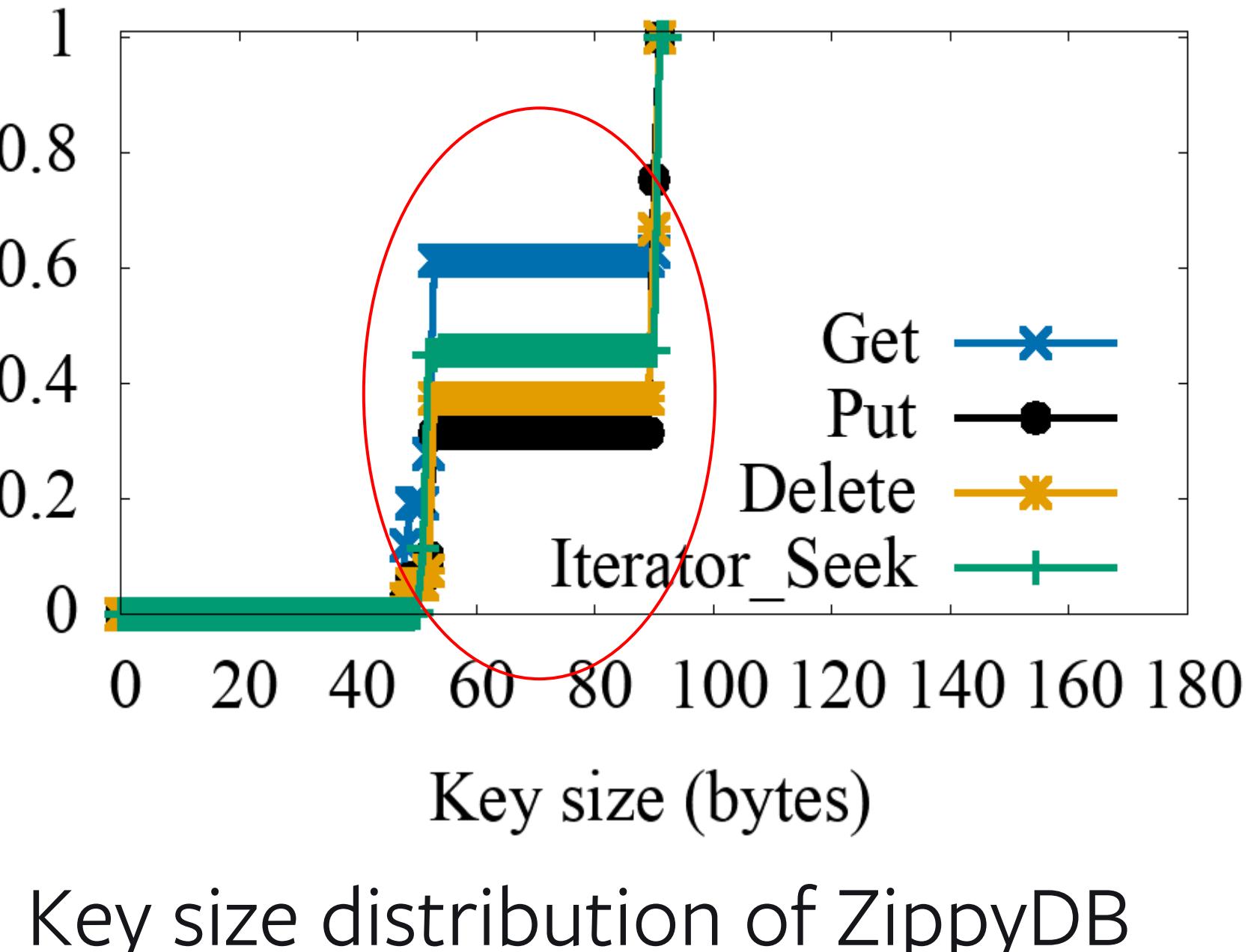
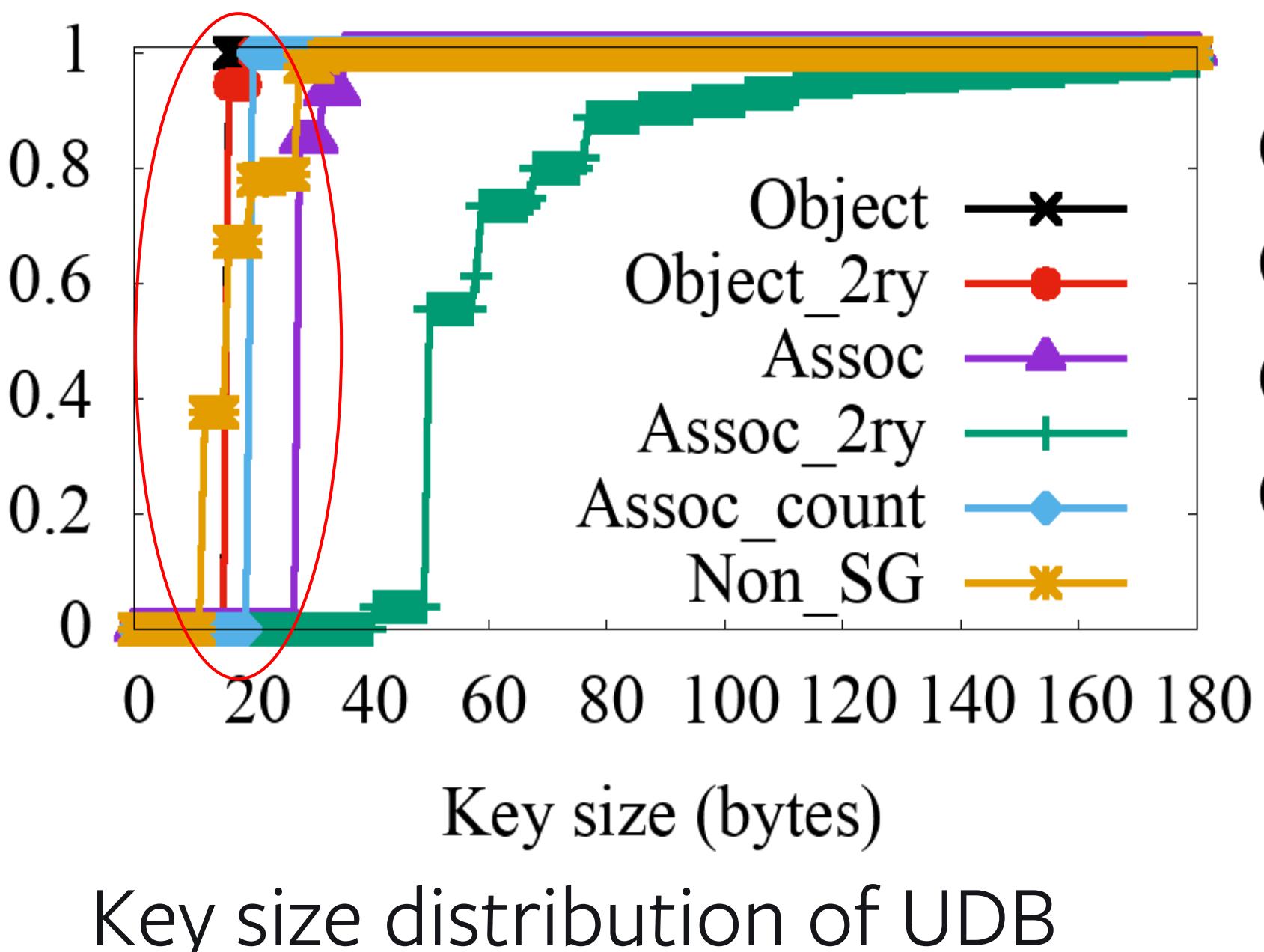
- The standard deviation of key sizes is relatively small, while the standard deviation of value size is large.
- The average value size of UDB is larger than the other two.

Table 2: The average key size (AVG-K), the standard deviation of key size (SD-K), the average value size (AVG-V), and the standard deviation of value size (SD-V) of UDB, ZippyDB, and UP2X (in bytes)

	AVG-K	SD-K	AVG-V	SD-V
UDB	27.1	2.6	126.7	22.1
ZippyDB	47.9	3.7	42.9	26.1
UP2X	10.45	1.4	46.8	11.6

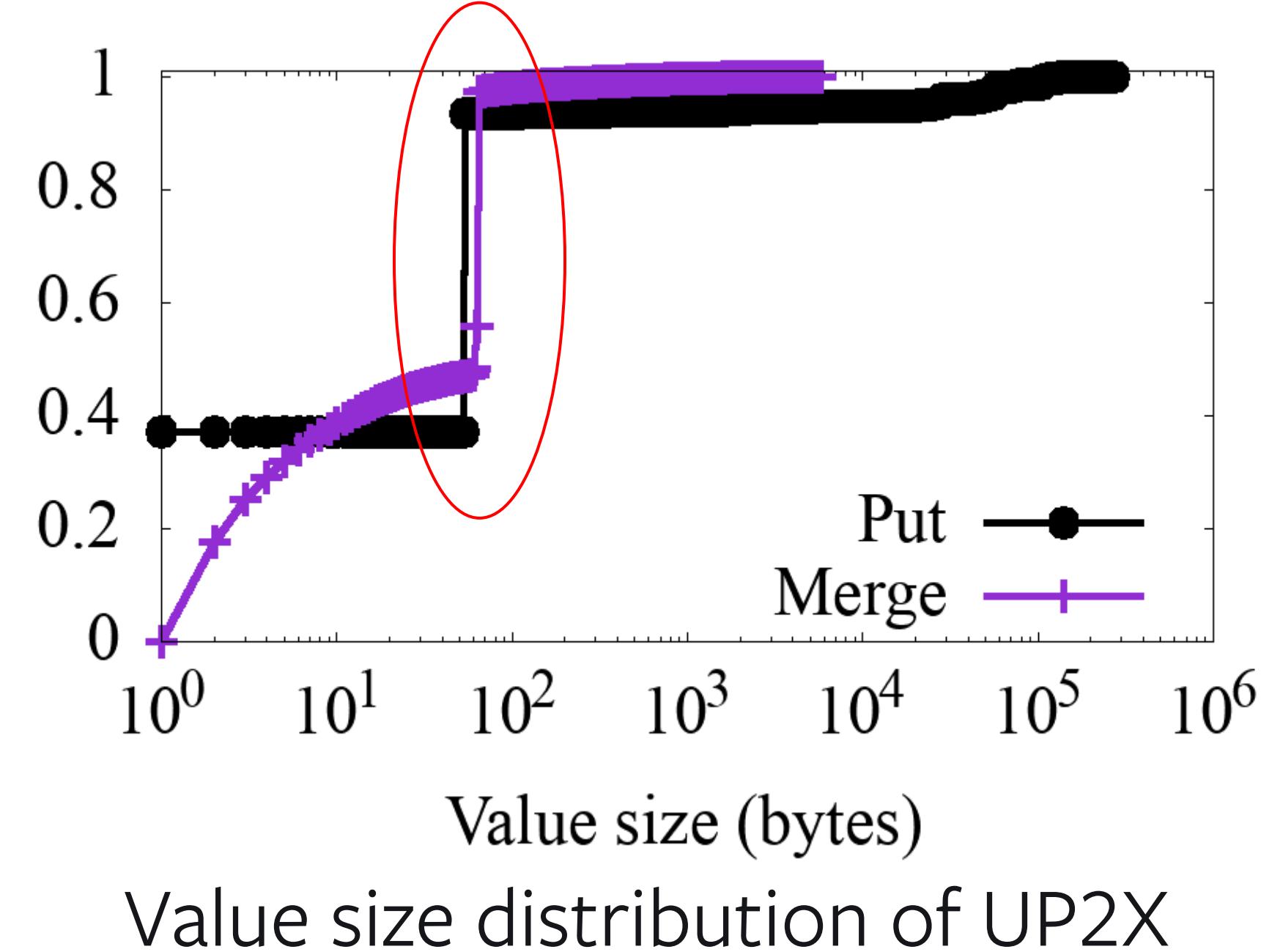
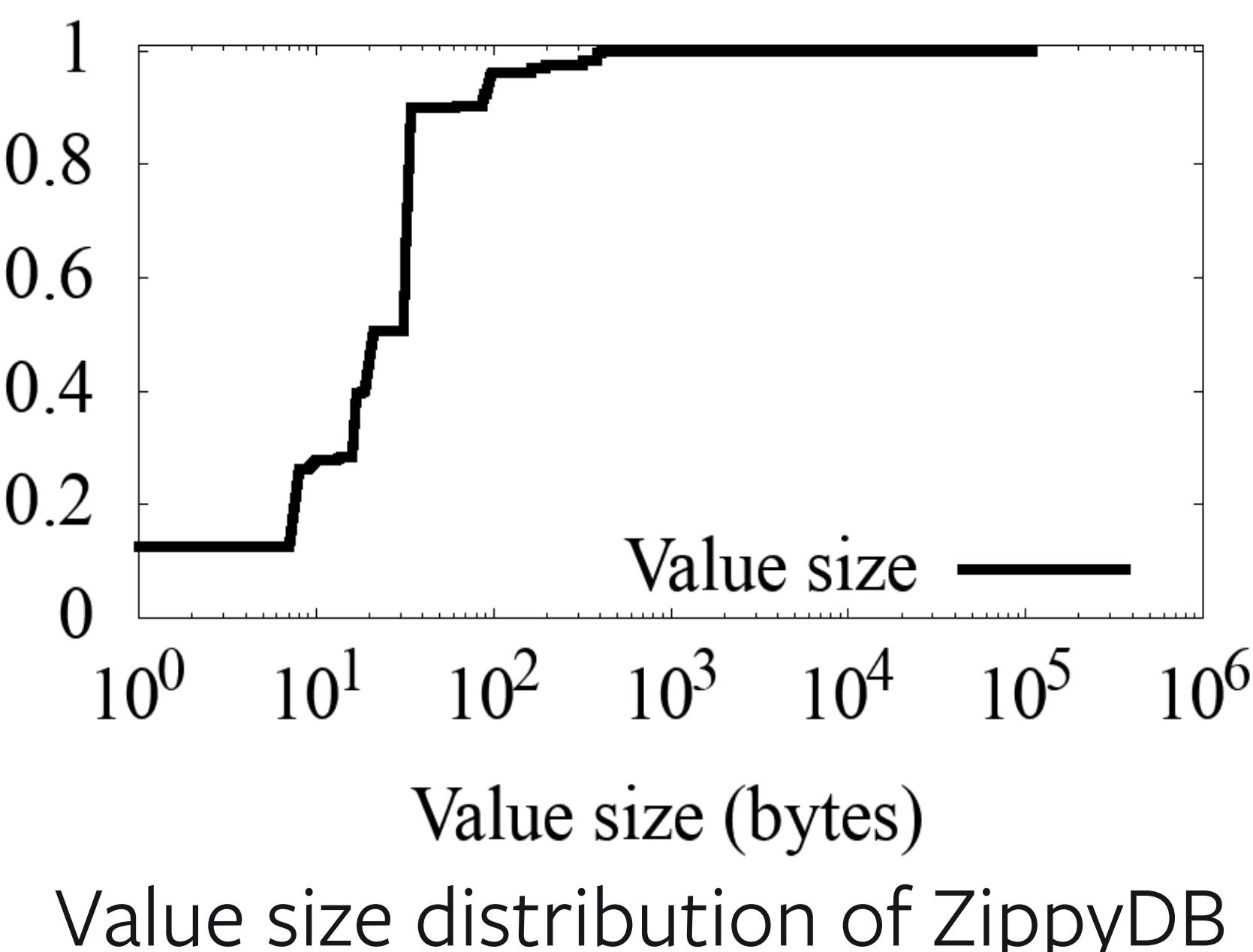
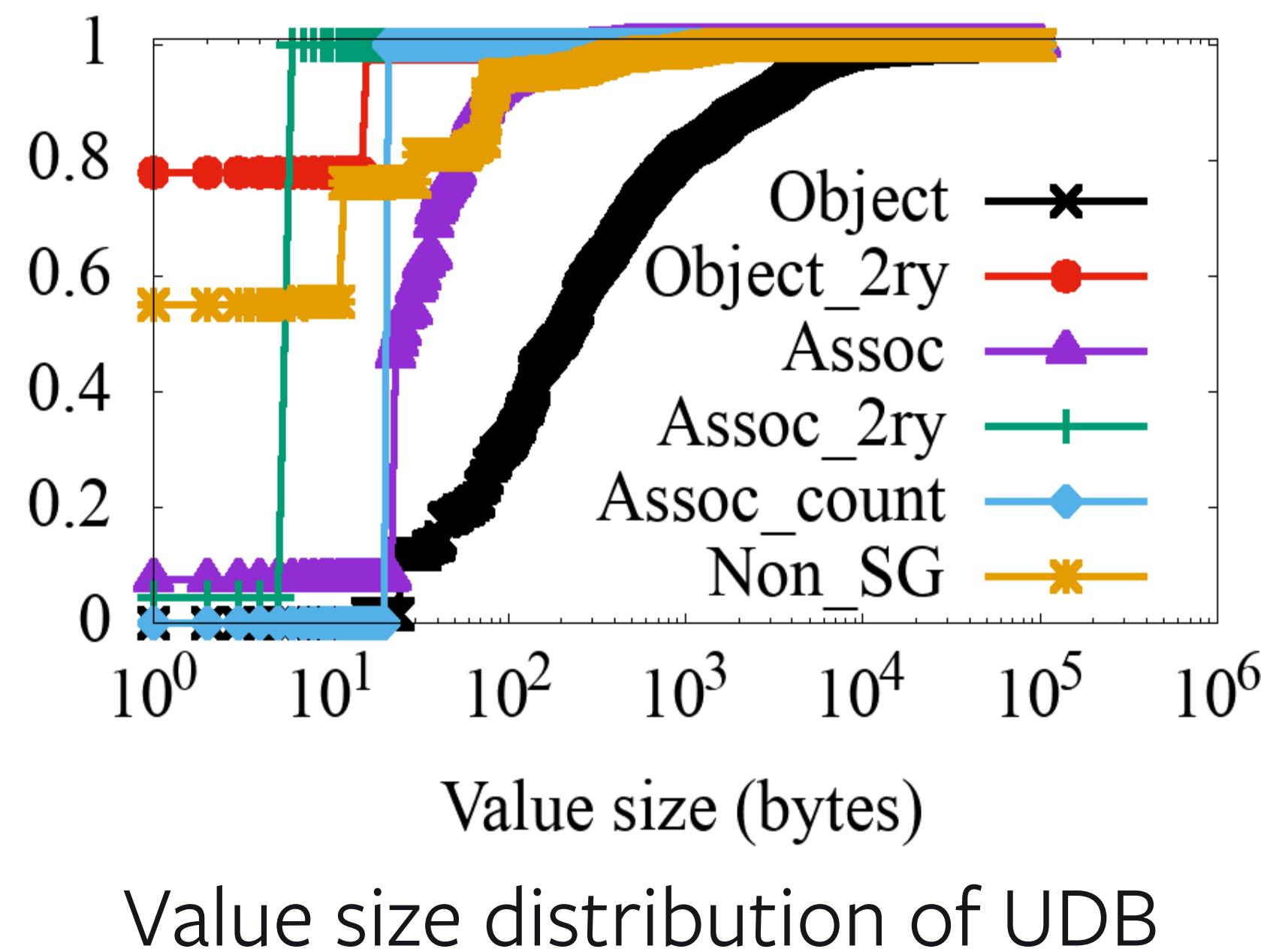
Key Size Distributions in Three Use Cases

- Key size is closely related to the way of key compositions in upper layer applications
- key sizes are usually small and have a narrow distribution
- The key size distributions in different CFs and in different query types are different.



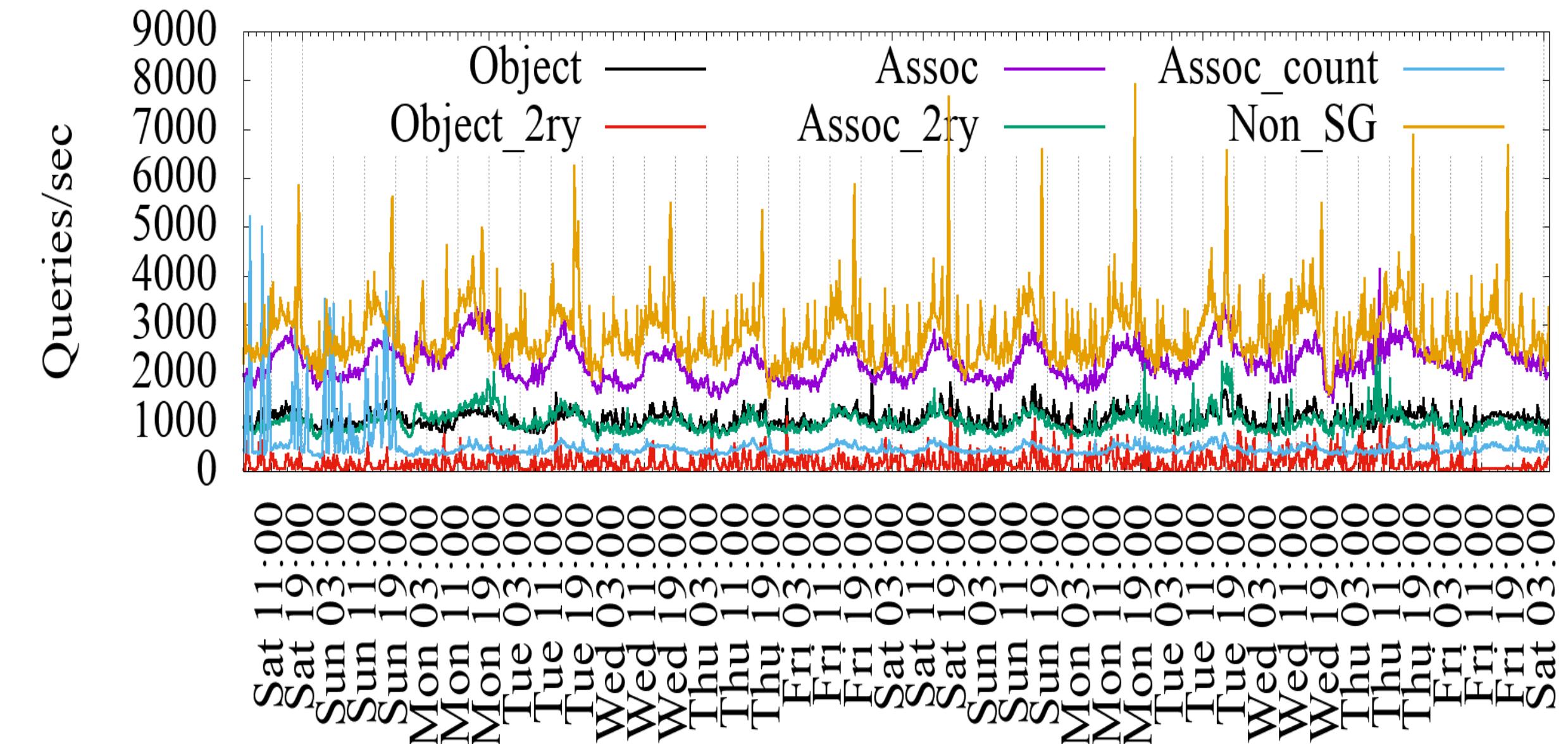
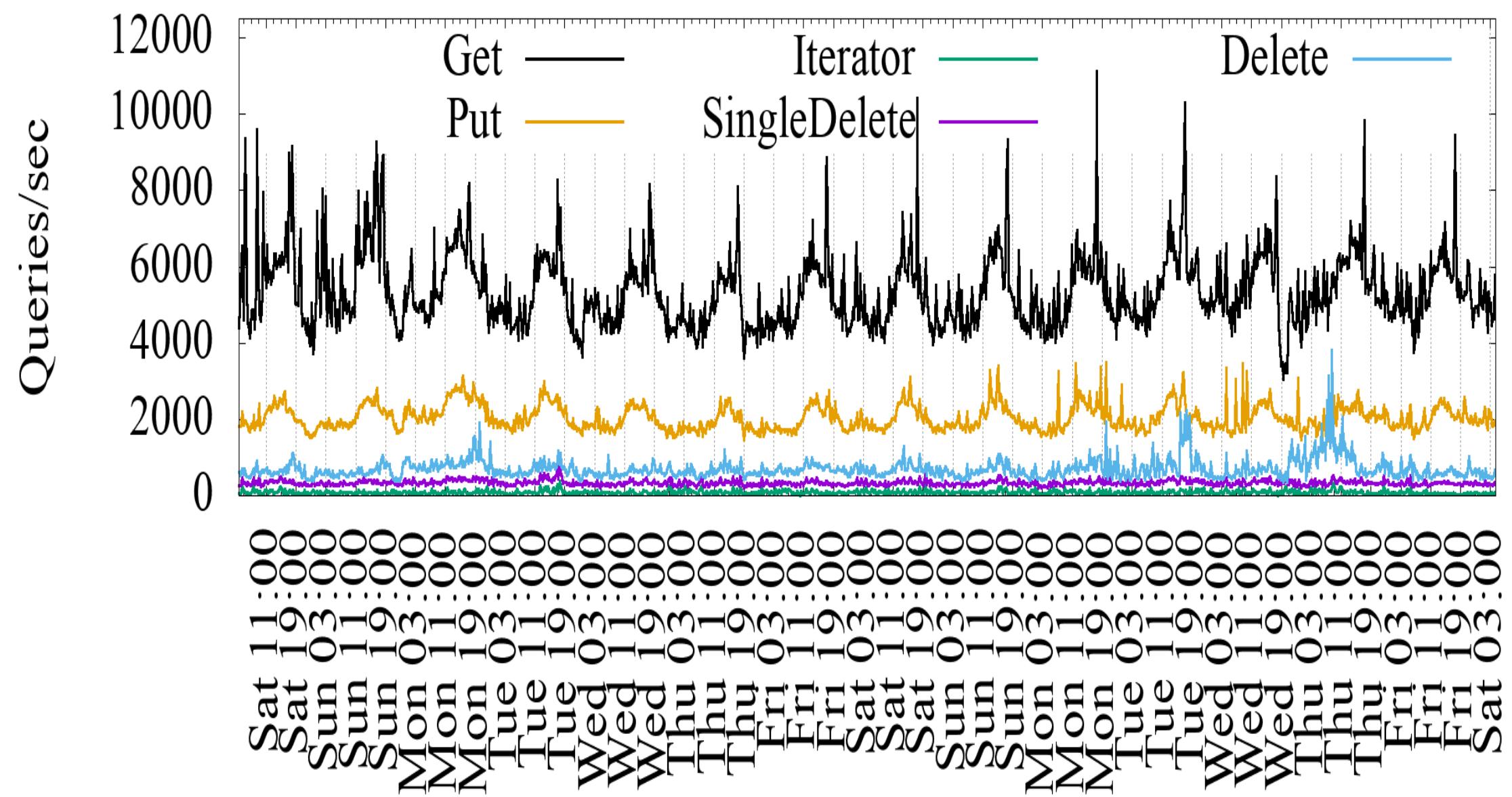
Value Size Distributions in Three Use Cases

- Value size distribution is wider than key size distribution
- value sizes are closely related to the types of data being stored by different applications



Query Per Second (QPS)

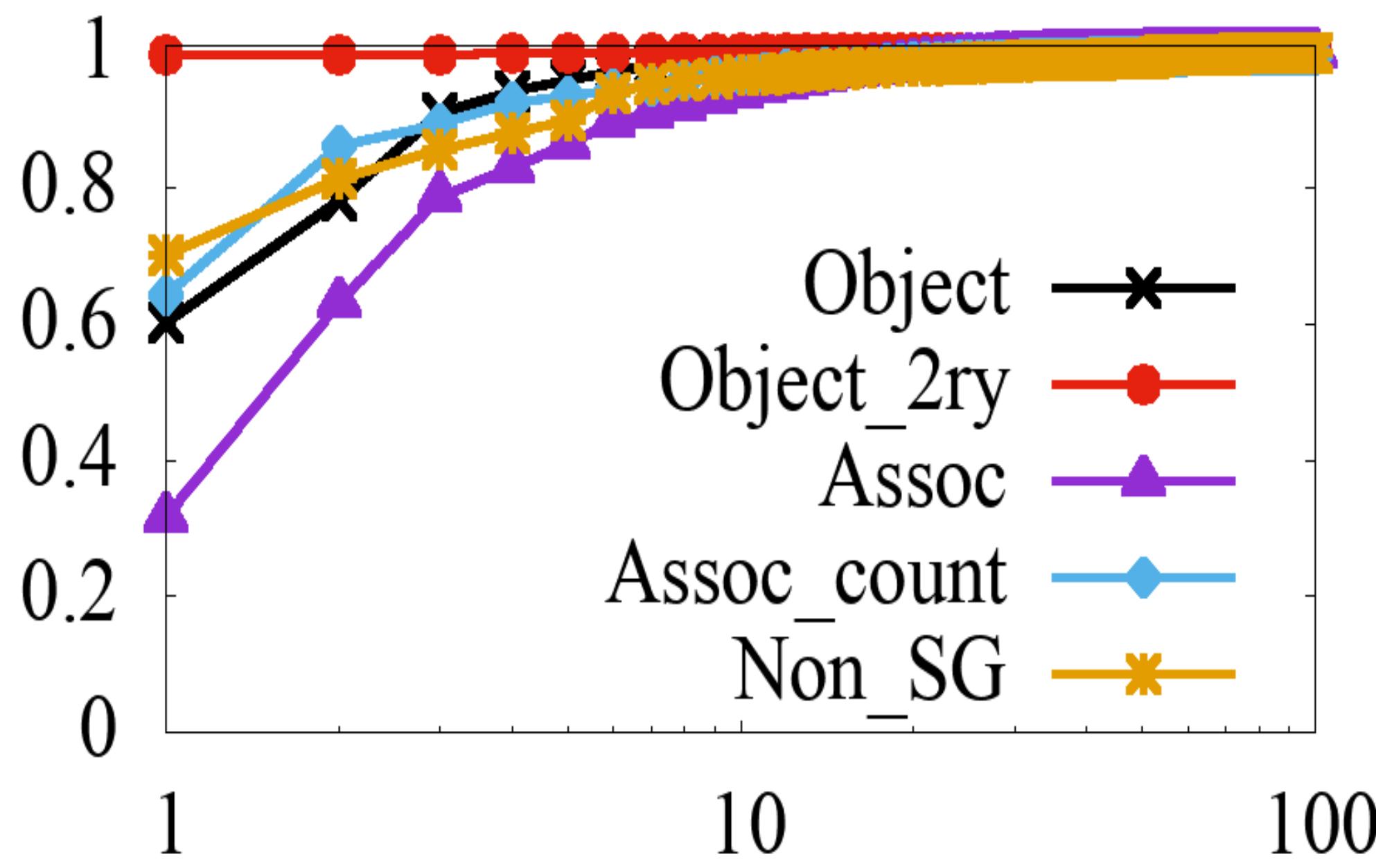
- The daily QPS variations of UDB are related to social network behaviors.
- The QPS of some CFs in UDB have strong diurnal patterns, while we can observe only slight QPS variations during day and night time in ZippyDB and UP2X.



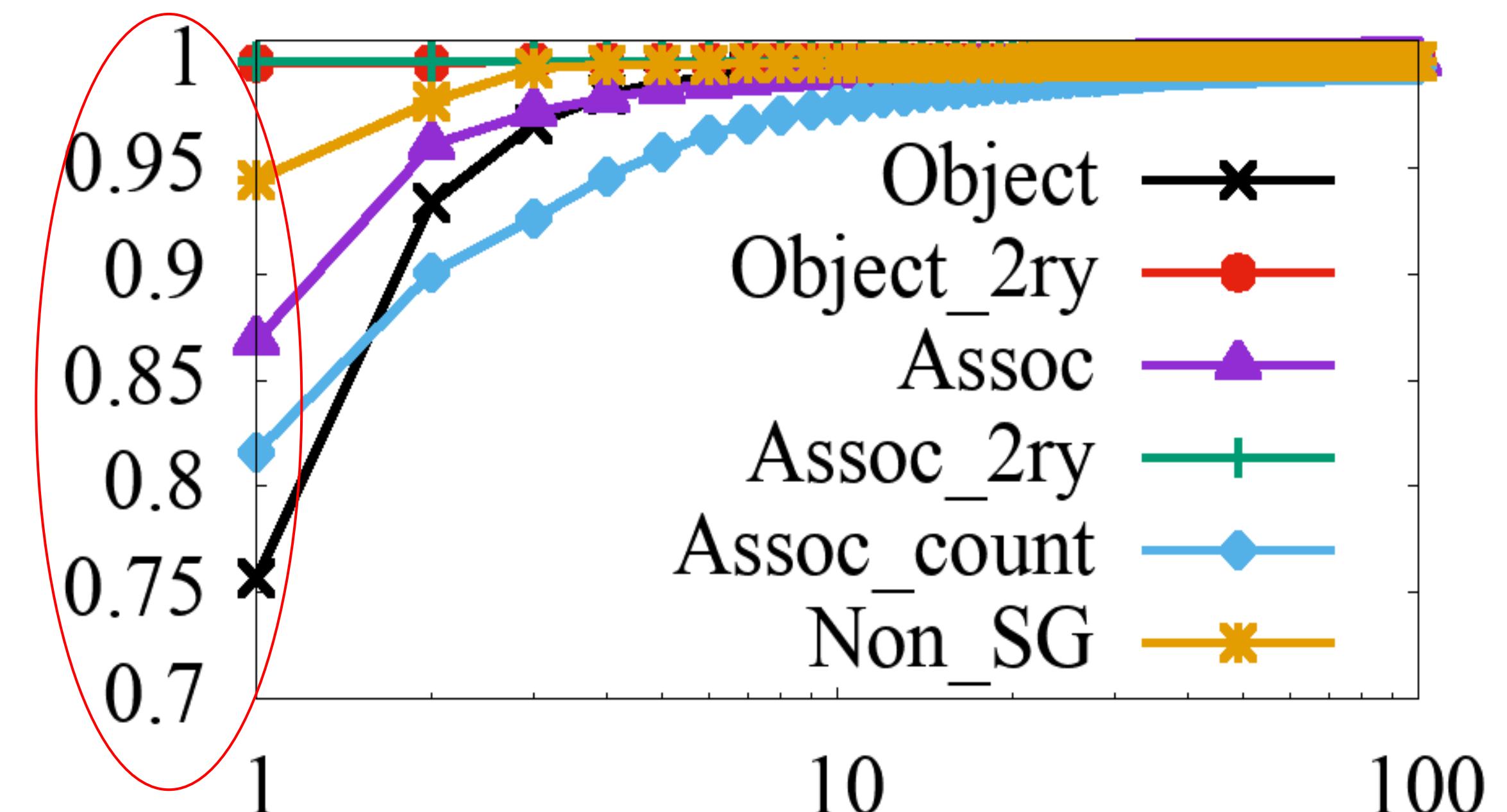
QPS of different query types and different column families in UDB during 14-day period

KV-pair Hotness and Access Count Distributions

- The access count distributions are very different in different applications and CFs
- In UDB, only a very small portion of KV-pairs are hot and most KV-pairs are cold.
- The access count distributions are different even in different CFs.



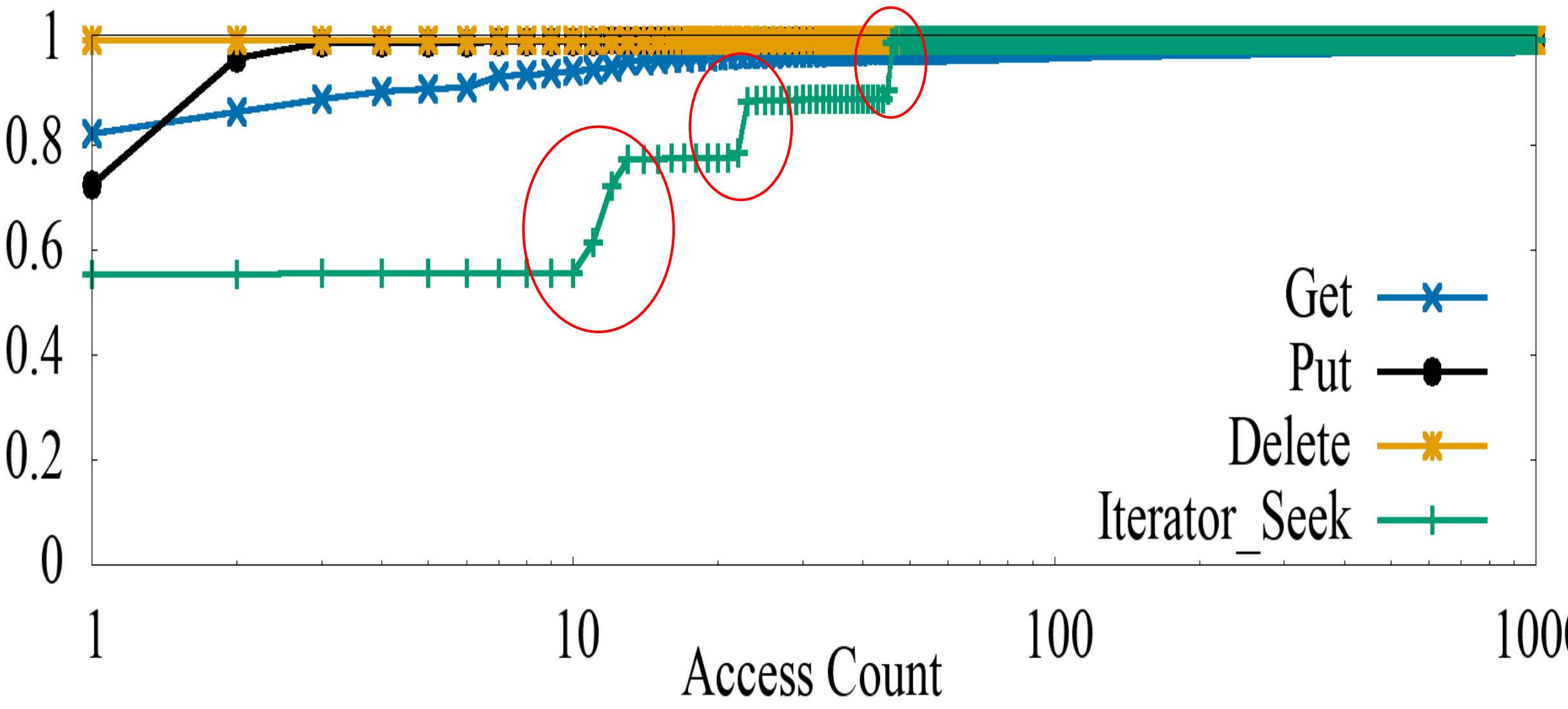
Access count distribution of Get in UDB



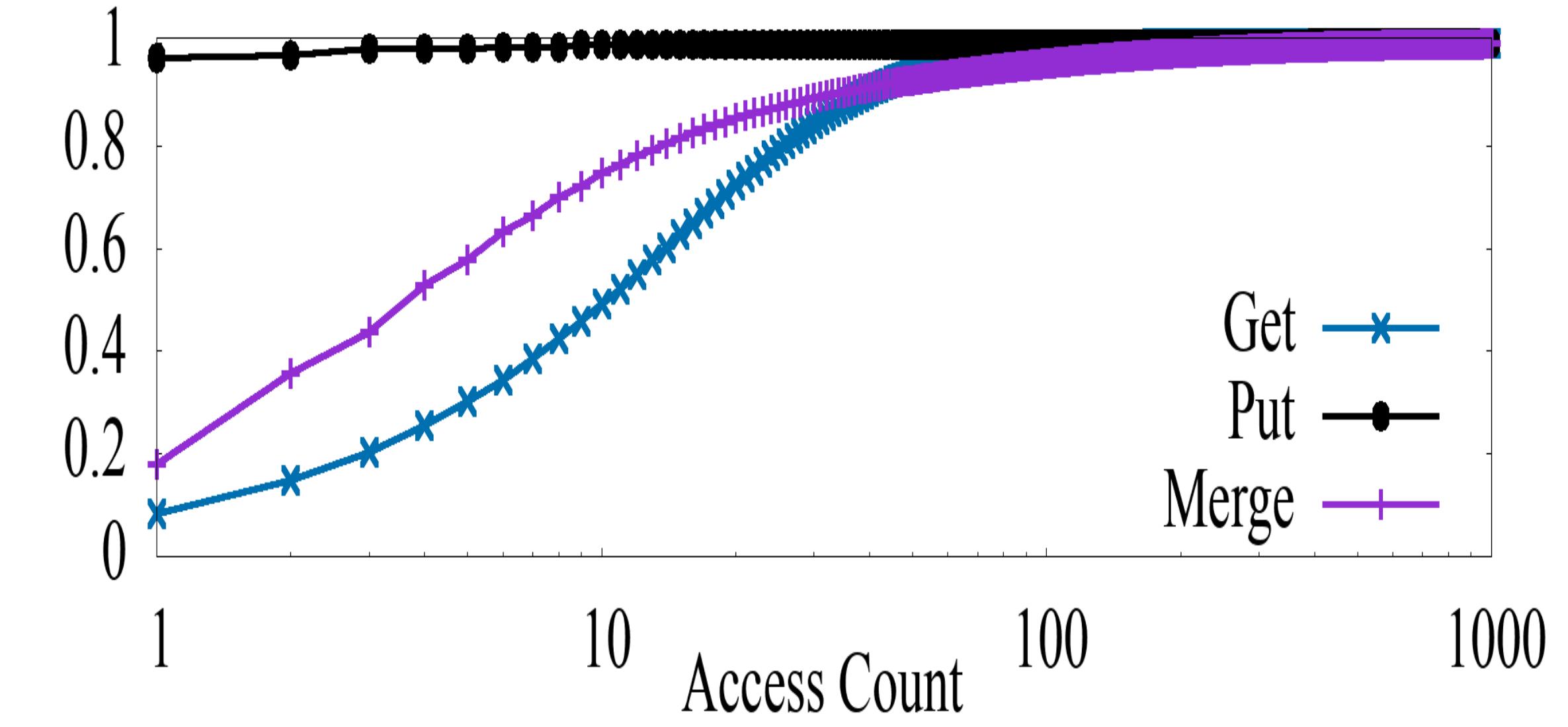
Access count distribution of Put in UDB

KV-pair Hotness and Access Count Distributions

- The access count distributions are very different in different applications and CFs
- In ZippyDB, only a very small portion of KV-pairs are hot and the access count distributions of different query types are different
- UP2X has a wider distribution especially for Get and Merge.



Access count distribution ZippyDB



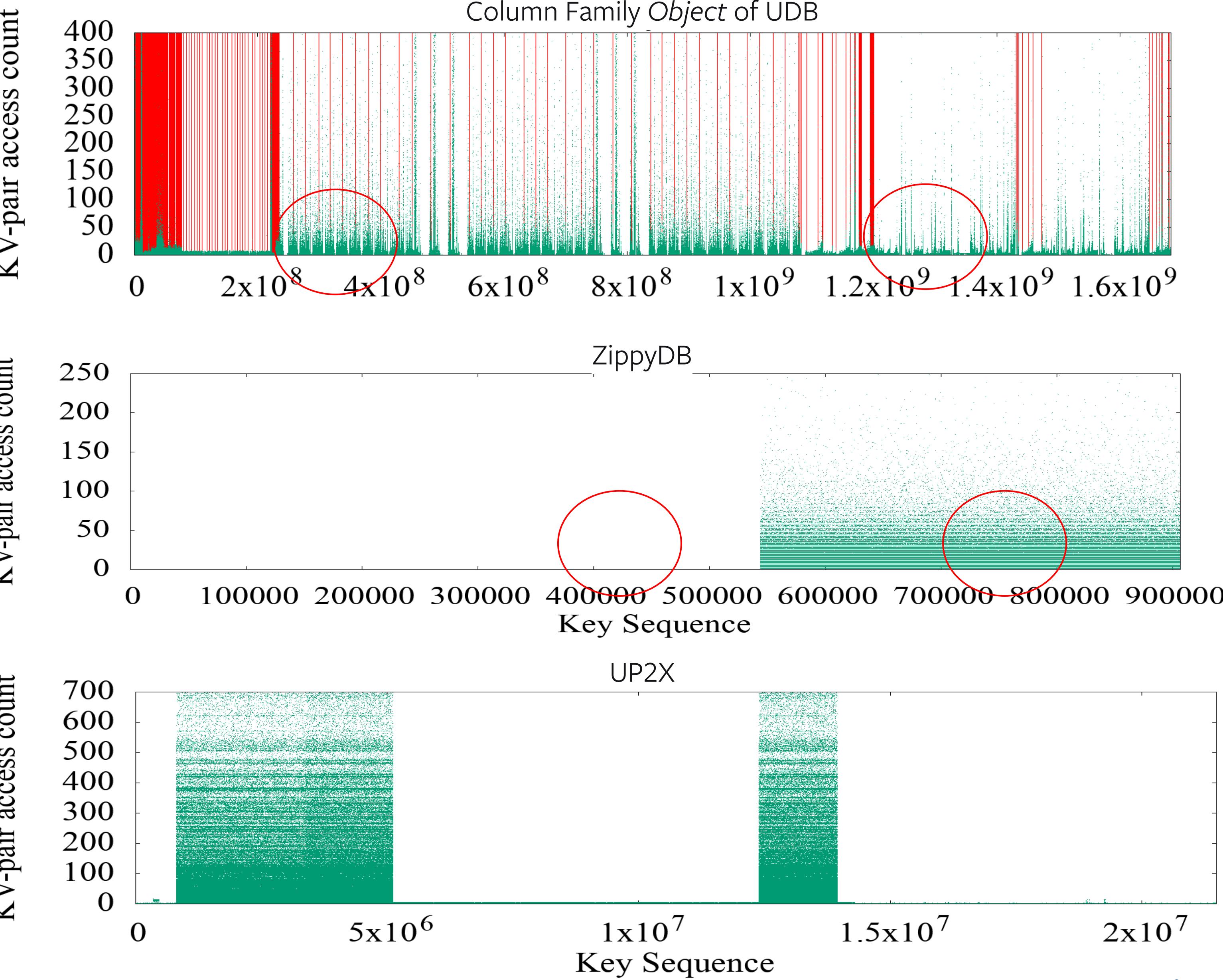
Access count distribution of UP2X

Access Heat-map

- The heat-maps of the three use cases show a strong key-space locality.
- Hot KV-pairs are not evenly distributed in the key-space. Instead, they are closely located in the key-space.

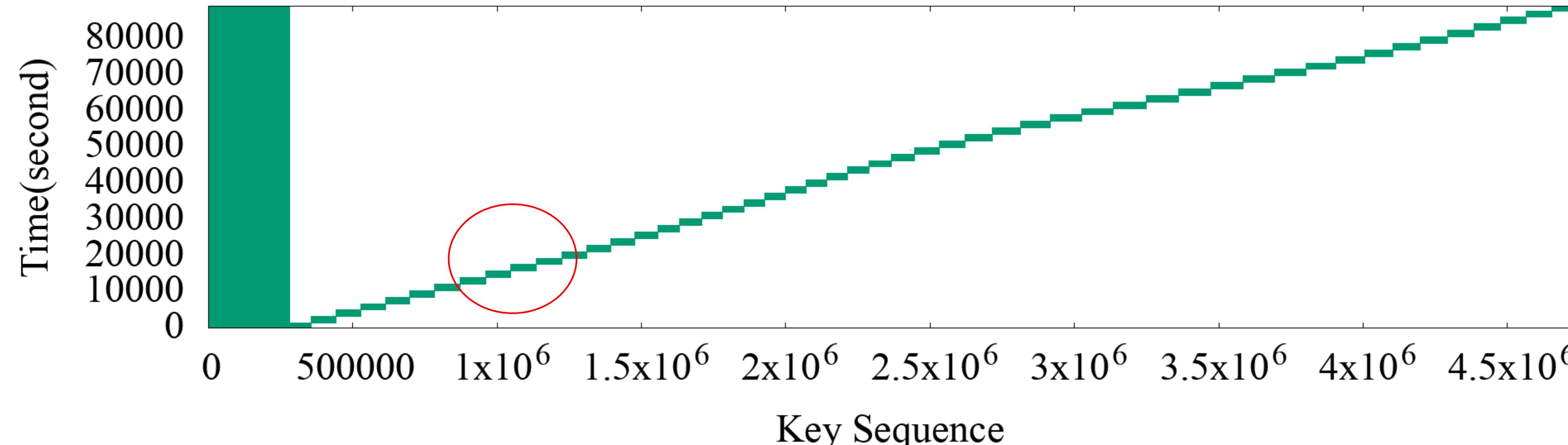
Note:

- 1) Keys are first sorted the same as they are in the database, and assigned with a integer as key sequence number.
- 2) Y-axis is the access count of a KV-pair during 24-hour period.
- 3) Red line is the separation of MySQL tables.



Key-Space and Temporal Localities

- The time series figures of Delete and SingleDelete for UDB and Merge for UP2X show strong temporal locality.
- For some query types, KV-pairs in some key-ranges are intensively accessed during a short period of time.



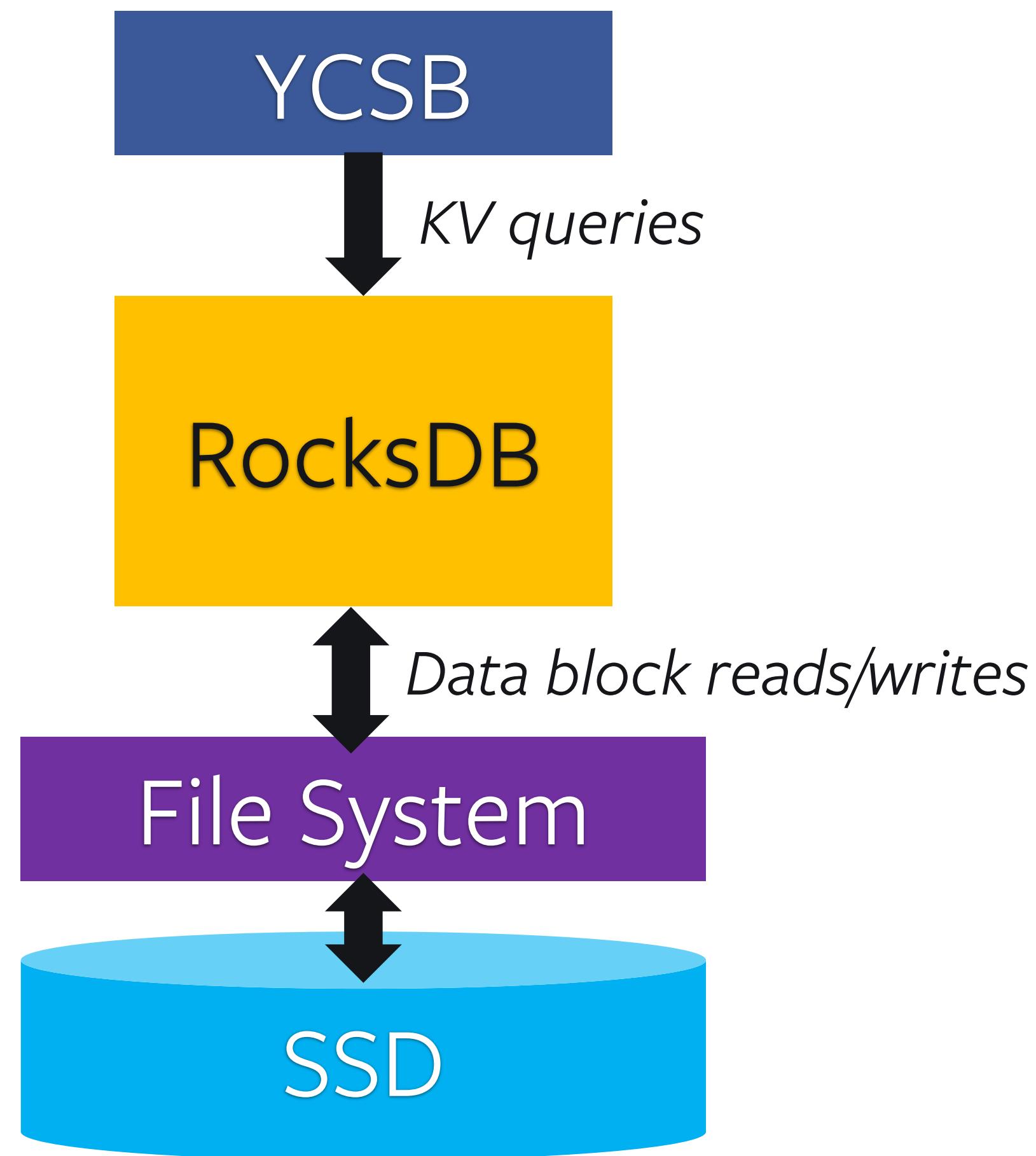
Time series figure of KV-pairs accessed by Merge in UP2X

Note: 1) X-axis is the key sequence number which is the same as that in heatmap, 2) Y-axis is time starting and starts at the time when tracing begins, and 3) Red line is the separation of MySQL tables.

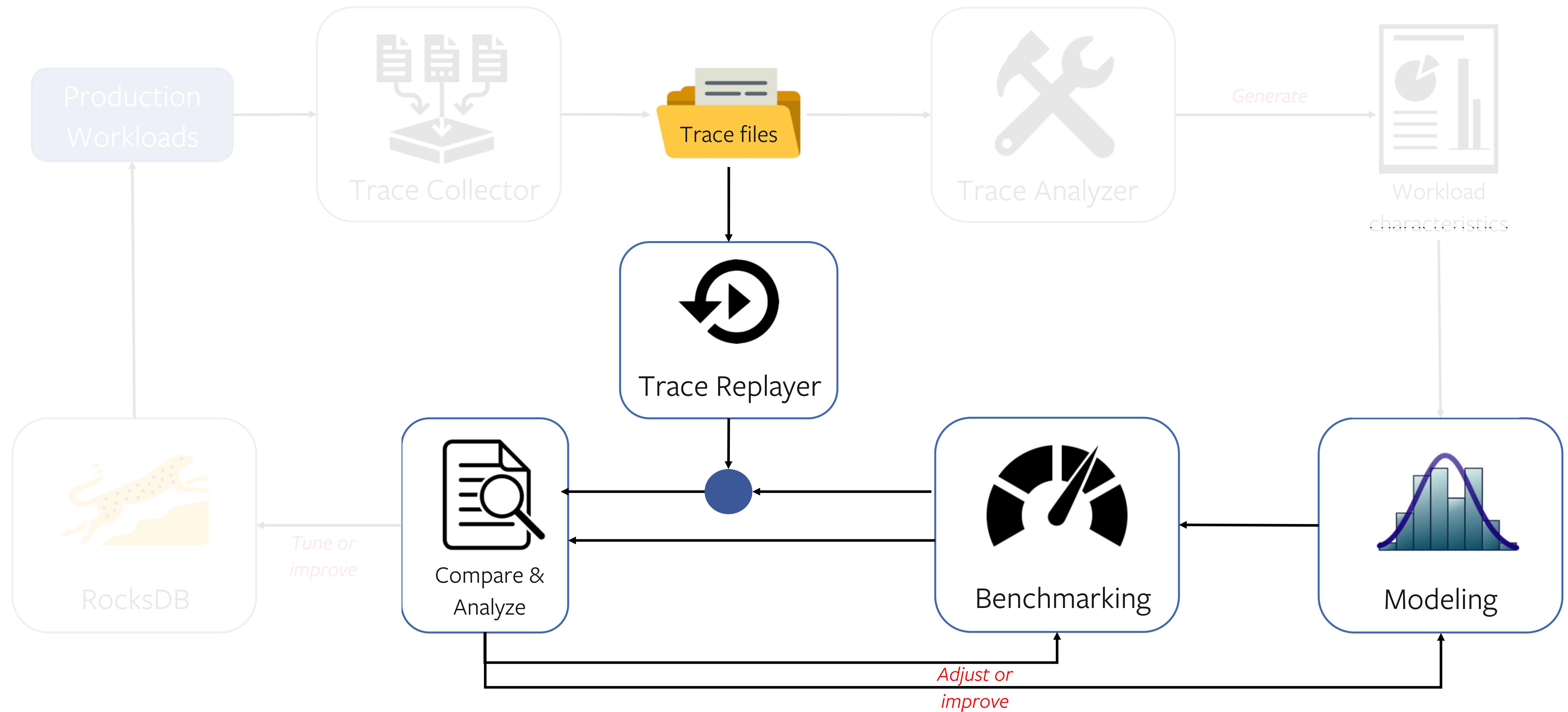
Modeling and Benchmarking

- After we have the detailed characteristics of real workload, we can compare the benchmarking results with the real-world workloads to explore its effectiveness.
- If the existing benchmark has limitations, the new workload modeling can be done with the help from workload characteristics. We can further improve the existing benchmarks or develop new benchmarks based on the new models.

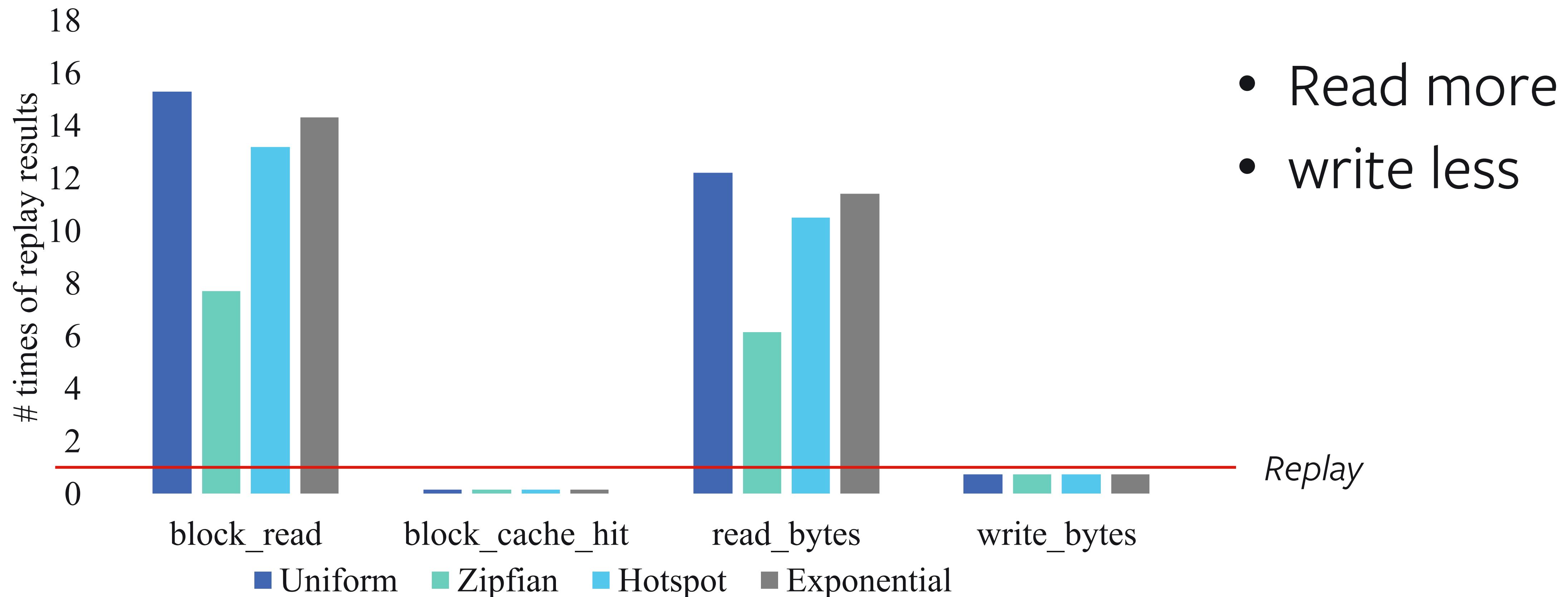
Investigate the Backend Storage I/Os



- YCSB can generate queries that have similar statistics for a given query type ratio, KV-pair hotness distribution, and value size distribution as those in realistic workloads.
- However, it is unclear whether their generated workloads can match the I/Os of underlying storage systems in realistic workloads.
- We focus on: *block reads* (`block_read`), *block cache hits* (`block_cache_hit`), *bytes being read* (`read_bytes`), and *bytes being written* (`write_bytes`).

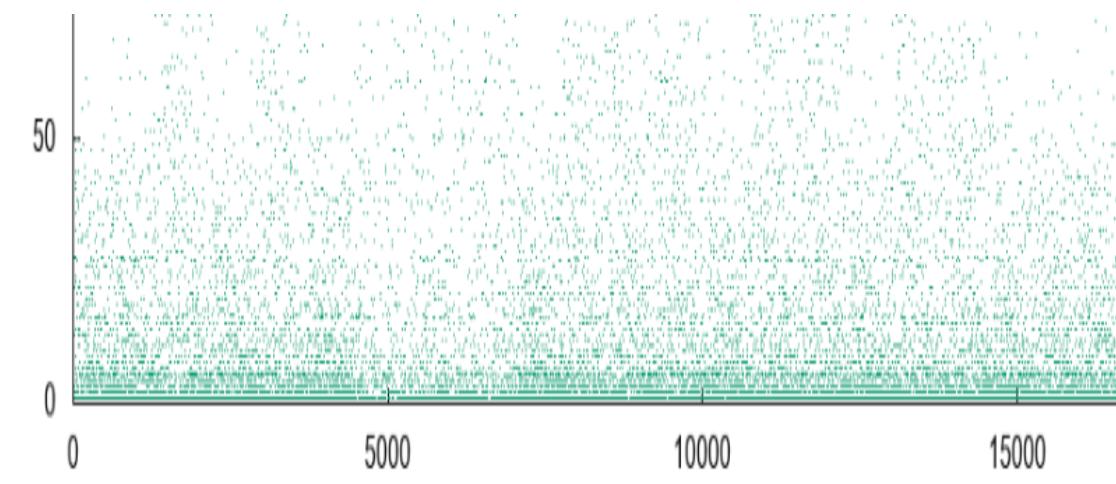


Replay of ZippyDB vs. Fitted YCSB

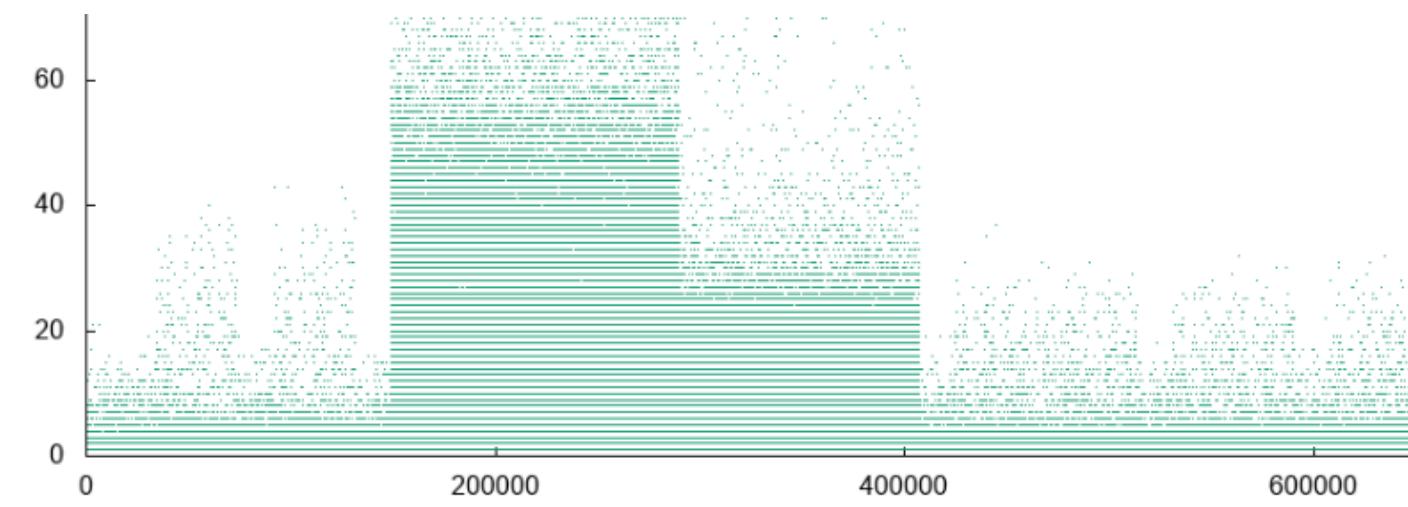


Problem and Solution

- The hot KV-pairs are actually randomly distributed in the whole key-space
- Due to the cache space limit, a large number of hot data blocks that consist of the requested KV-pairs will not be cached, which triggers an extremely large number of block reads.
- **Instead of only modeling the overall KV-pair hotness, we cut the whole key space into small key-ranges and models the hotness of these key-ranges.**
- **In each key-range, KV-pairs follow the overall hotness distributions.**
- **Build the new benchmark based on key-range based modeling**

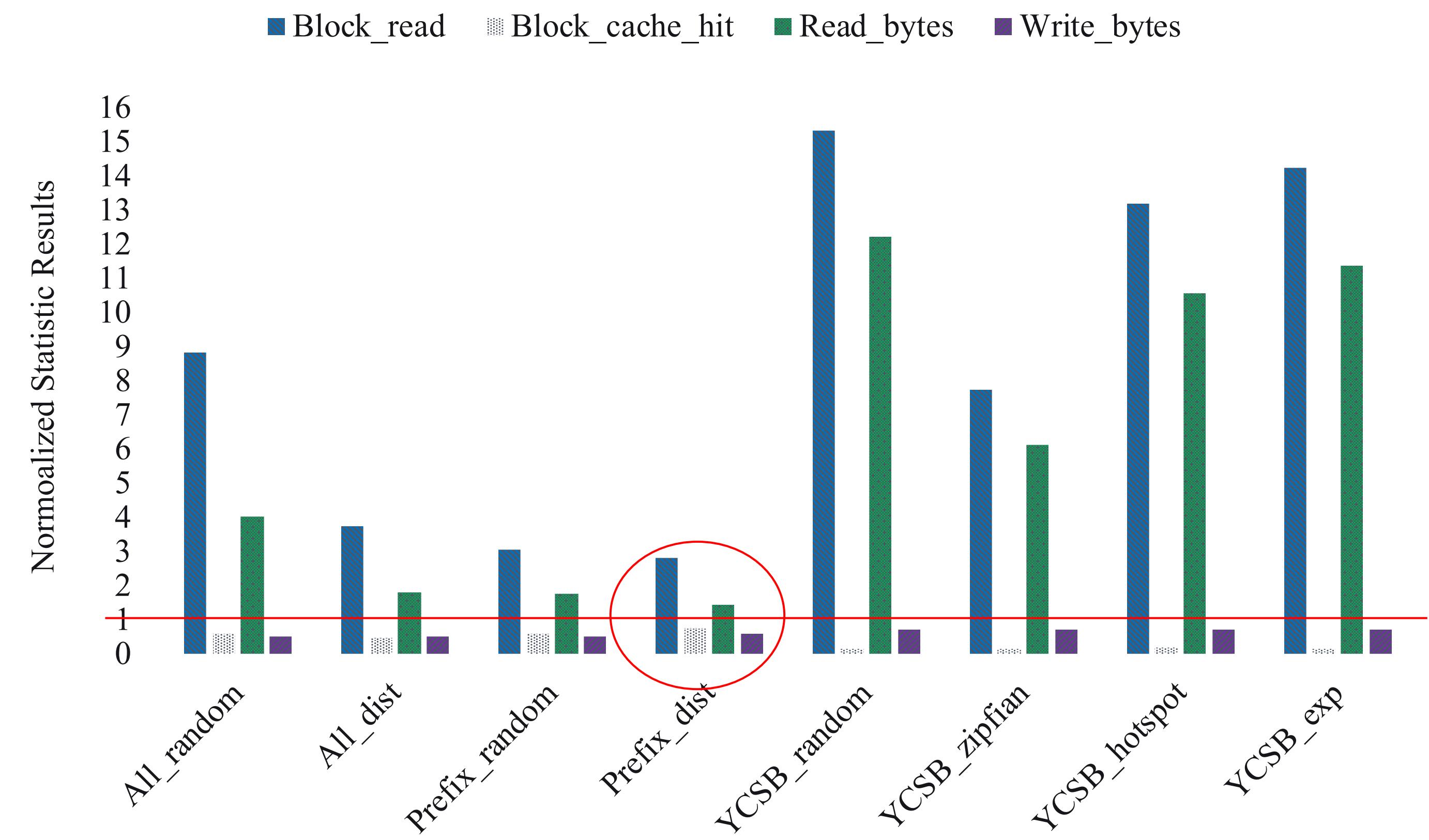


Heatmap changes to



New Benchmark vs. YCSB

- **All_random**: KV-pairs are randomly distributed
- **All_dist**: hot KV-pairs are allocated together
- **Prefix_random**: KV-pairs are allocated to different key-ranges based on key-range hotness. KV-pairs in a key-range are randomly distributed.
- **Prefix_dist**: KV-pairs are allocated to different key-ranges and hot KV-pairs in a key-range are allocated close by



Conclusion and Future Work

- We introduce the key-value workload analyzing, modeling, and benchmarking methodologies. The tools are **open-sourced** and people can use them to extend the exploration to more applications
- With the help of tracing, replaying, and analyzing tools, we characterized key-value workloads of three typical RocksDB production use cases at Facebook. The findings of key/value size distribution, access patterns, key-range localities, and workload variations provide insights that can help optimize KV-store performance.
- We propose a key-range based model to better preserve key-space localities. The new benchmark not only provides a good emulation of workloads at the query level, but also achieves more precise RocksDB storage I/Os than that of YCSB.
- In the future, we will continue improve YCSB such as key-range based distribution and QPS variation. Also, we will extend the workload characterization to other dimensions such as the correlations between queries, the correlation between KV-pair hotness and KV-pair sizes.

Future of RocksDB

A **fast** and **easy to use** persistent key-value store for **any workload** on **any hardware platform**.



Efficiency



Easy To Use



Performance

[facebook / rocksdb](#)

Unwatch releases ▾ 990 ★ Star 16.1k ⌂ Fork 3.6k

Code Issues 294 Pull requests 139 Actions Projects 1 Wiki Security Insights FB Internal

A library that provides an embeddable, persistent key-value store for fast storage. <http://rocksdb.org>

8,769 commits 104 branches 0 packages 195 releases 1 environment 504 contributors View license

<https://github.com/facebook/rocksdb>

Thank You!

Q&A



facebook