AC-Key: Adaptive Caching for LSM-based Key-Value Stores

Fenggang Wu, Ming-Hong Yang, Baoquan Zhang, and David H.C. Du

ATC ‘20

Presenter: Kai Li

Background

- In enterprise workloads, read operations exhibit “**hot spots**” in LSM-tree-based KVSs (LSM-KVS) for both point lookups and range queries.
- **Caching** “hot spots” can improve read performance.
- Three type of entries that can be cached in LSM-KVS: Block, Key-Value (KV), and Key-Pointer (KP).

Motivation

- Caching LSM-KVS is **challenging**
  - Data have different sizes/levels -> different cache costs/benefits
    - Cache a record at a deeper level can bring more benefits, as it reduces more storage IOs
  - Different type of read: point lookup, range query
    - **Cache a block** favors range query
    - **Cache a KV/KP** favors point query
    - When value size is **large**
      - Cache a KP is more space-efficient than cache a KV

	Block	KV	KP	Point	Range	Adaptive
LevelDB	Yes	No	No	Inefficient	Supported	Fix-sized
RocksDB	Yes	Yes	No	Large Value inefficient	Supported	Fix-sized
Cassandra	No	Yes	Yes	Efficient	Not Supported	Fix-sized
AC-Key	**Yes**	**Yes**	**Yes**	**Efficient**	**Supported**	**Adaptive-sized**

Source: https://www.usenix.org/system/files/atc20-paper107-slides-wu.pdf

Cache Size Adjustment with Ghost Cache

- Adaptive Replacement Cache (ARC)
  - Real Cache + Ghost Cache
  - Ghost Cache hit will push the boundary
  - Adaptive to different cache components

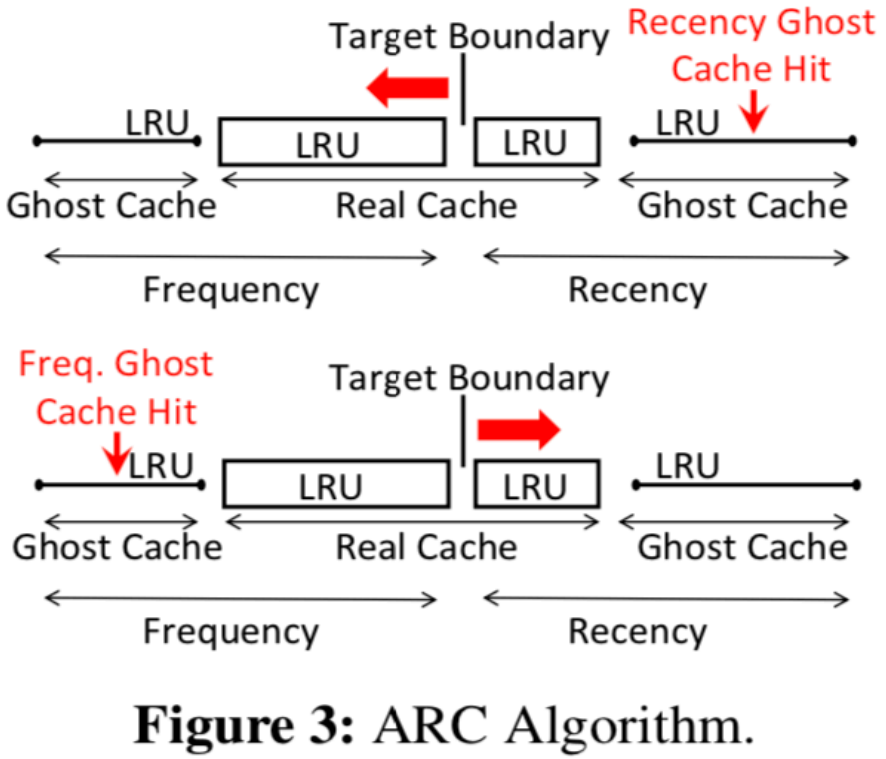
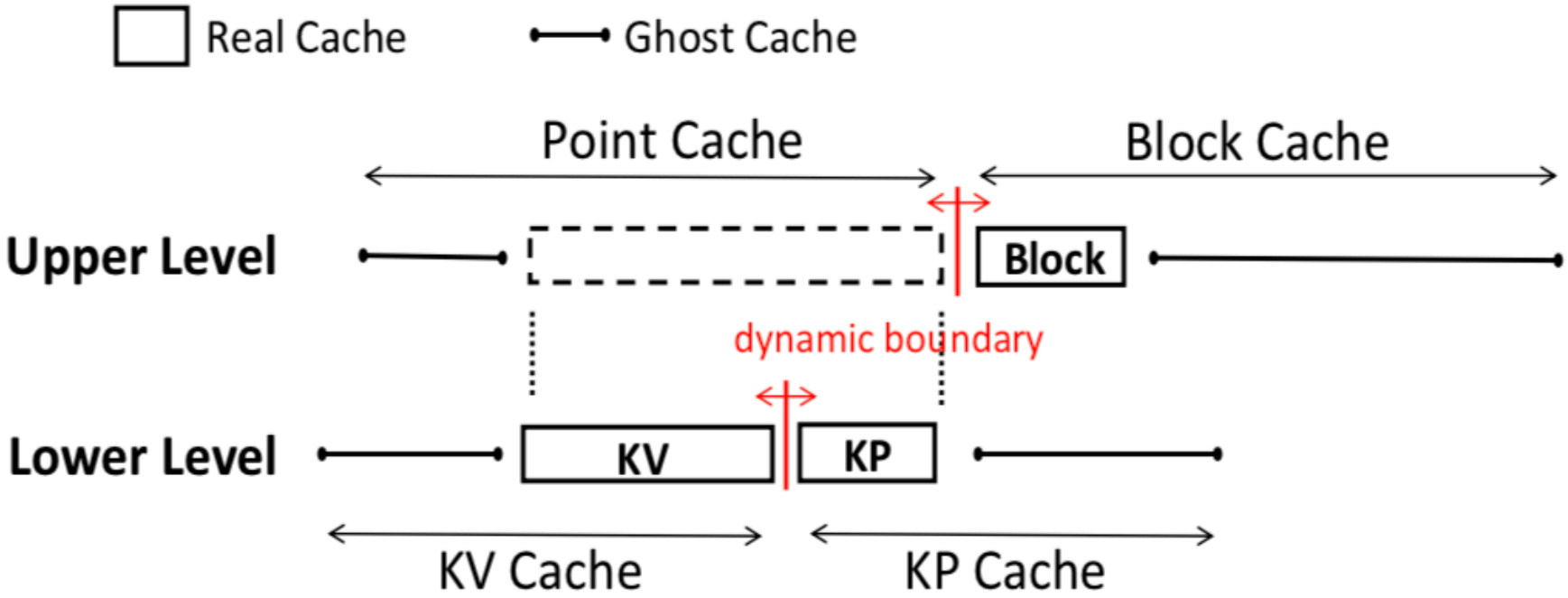


Figure 3: ARC Algorithm.

AC-Key: Hierarchical Adaptive Caching

Upper Level: Point Cache vs Block Cache

Lower level: KV Cache vs KP cache



Get Handling

- *Case I*: Hit in KV Cache
  - The value is returned without any I/O incurred.

- *Case II*: Miss in KV Cache but hit in KP Cache.
  - Check Block Cache (and bring to Block Cache if missing)
  - Promote KP Cache to KV Cache

- *Case III*: Miss both in KV Cache and KP Cache.
  - Search every sorted run level by level
  - Move into KP Cache

Flush and Compaction

- Sync the caches only in flushing time, not during *Put*

- AC-Key updates KP and block Caches when compaction affects any of cached KP entries or blocks

Caching Efficiency Factor

- To quantitively analyze the trade-off between the costs and benefits of the cache entries
  - Target Boundary Adjustment by *Caching Efficiency Factor E*
  - Adjustment $\Delta = kE$

$$E = \frac{b}{s}, \tag{1}$$

where: E = caching efficiency factor of one cached entry,
b = number of saved storage I/O if cached,
s = caching space taken by this entry.

$$b = \begin{cases} 1 & \text{if block,} \\ f(m) & \text{if KV entry,} \\ f(m) - 1 & \text{if KP entry.} \end{cases} \tag{2}$$

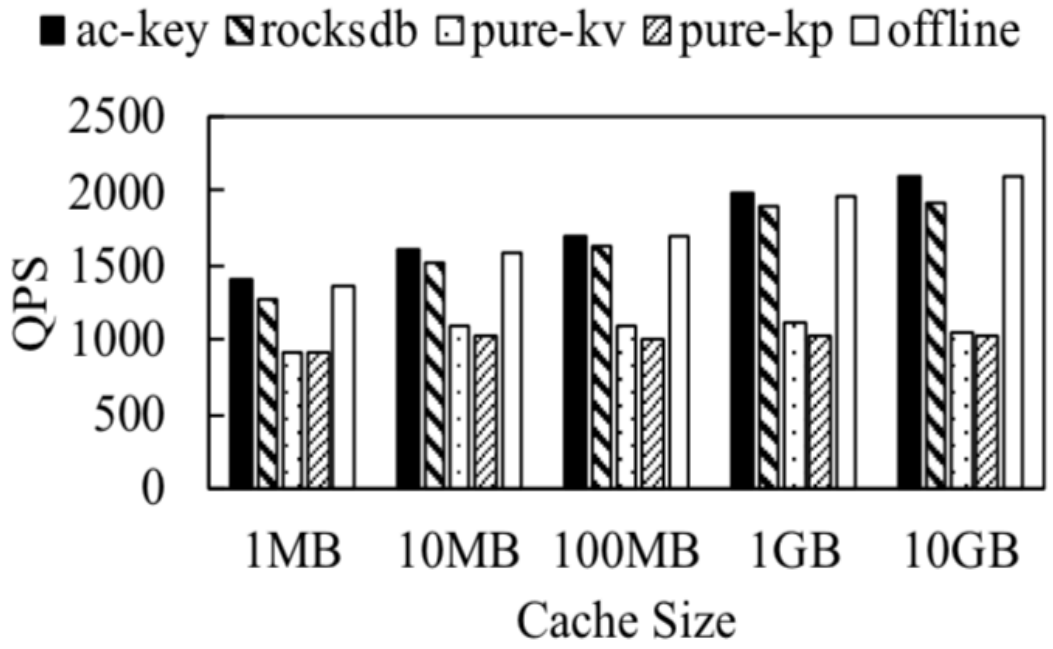
where: m = number of SSTs to search for the key,
$f(m)$ = number I/Os to get a key. It is a function of m.

$$m = \begin{cases} n_0/2 & \text{if } l = 0, \\ l + n_0 & \text{if } l >= 1. \end{cases} \tag{3}$$

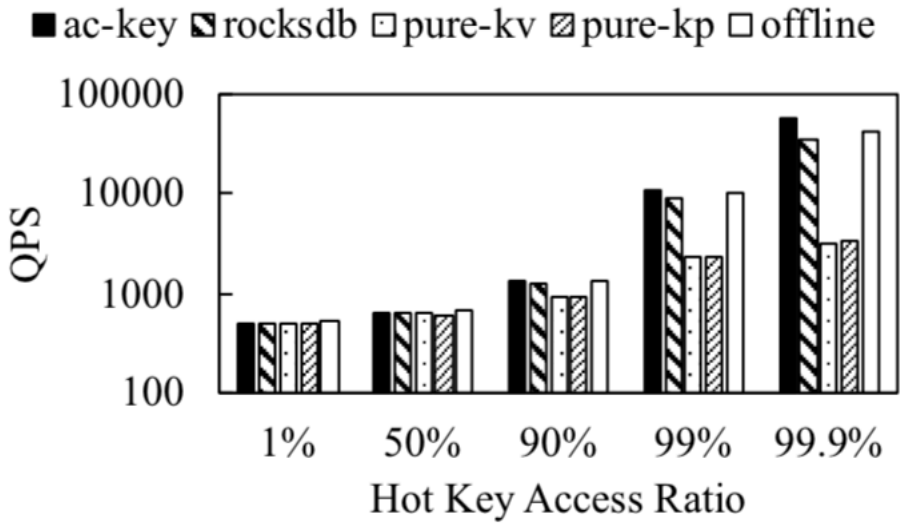
where: n_0 = max number of SSTs L_0 can hold,
l = the level where the key resides.

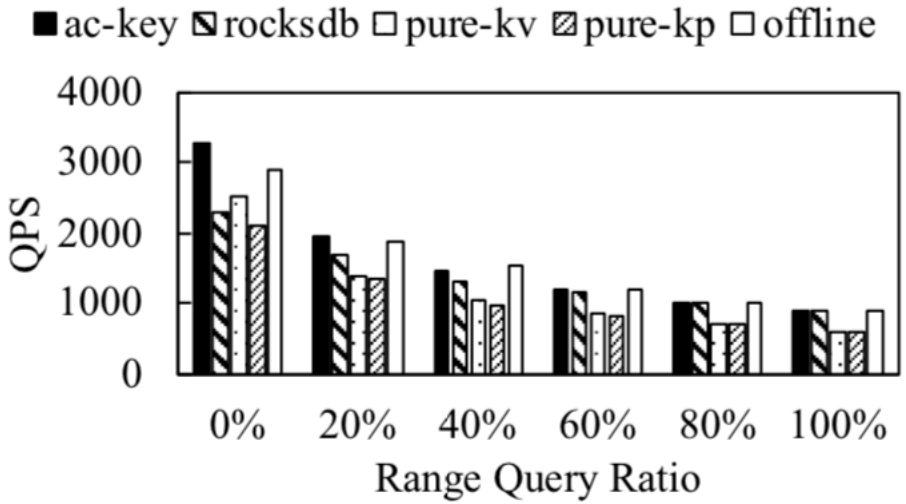
Evaluation

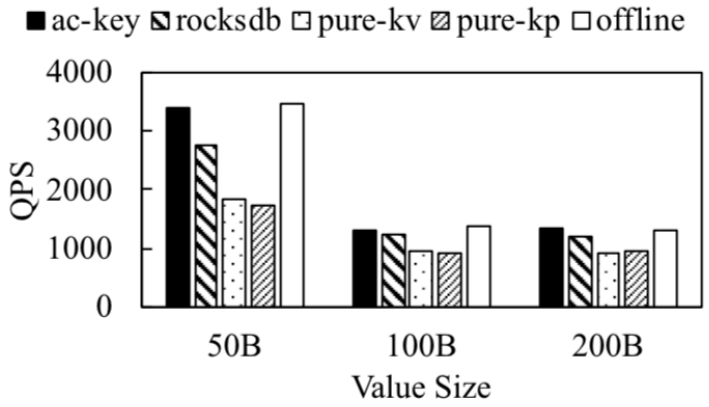
- Implementation on RocksDB, 5.6K LoC.
- Compare *AC-Key* with *pure-kv, pure-kp, rocksdb, offline*

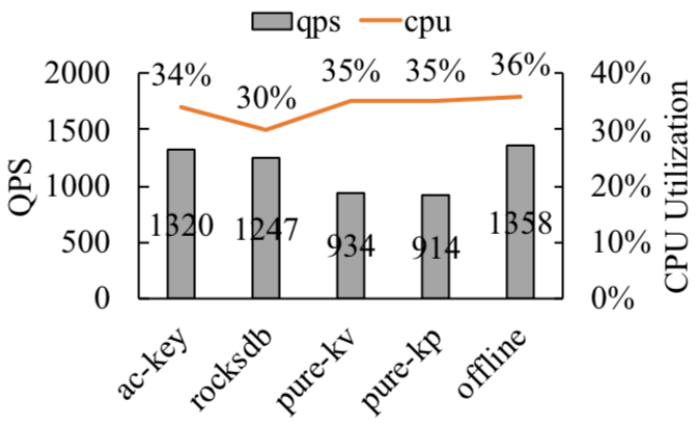


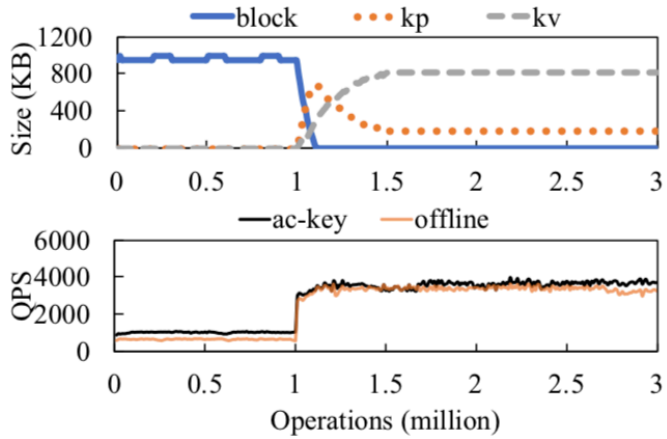
Extensive Evaluations











Thank you!

Q&A