HotRing: A Hotspot-Aware In-Memory Key-Value Store

Jiqiang Chen, Liang Chen, Sheng Wang, Guoyun Zhu,

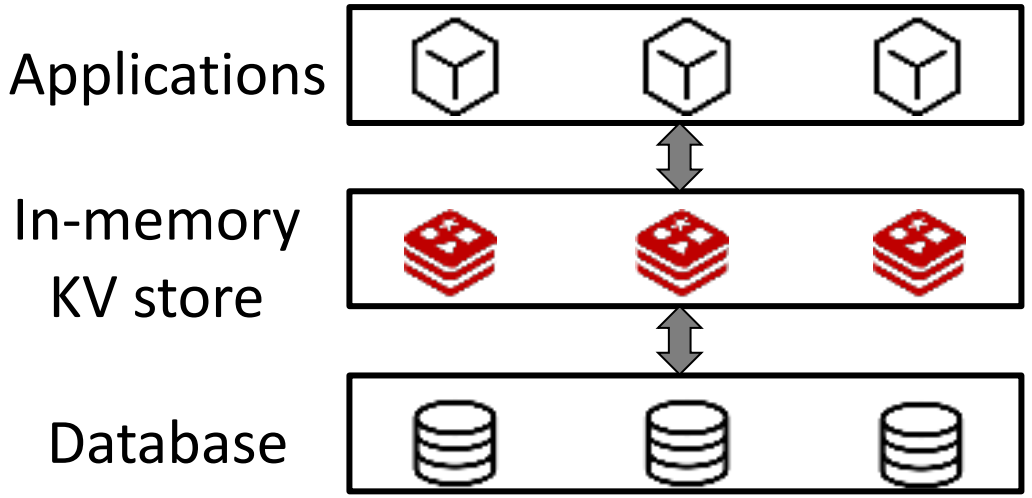
Yuanyuan Sun, Huan Liu, and Feifei Li, Alibaba Group

FAST 2020

Background

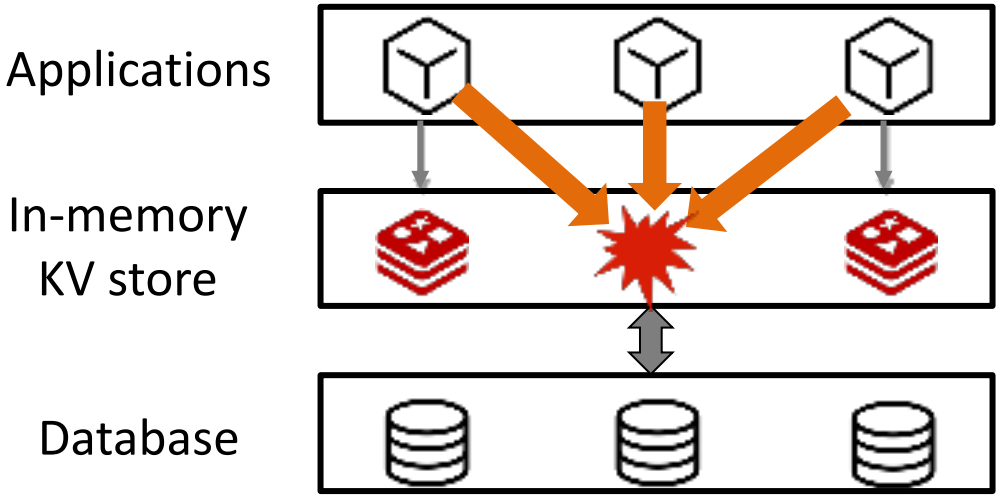
➢ In-memory Key-Value Store

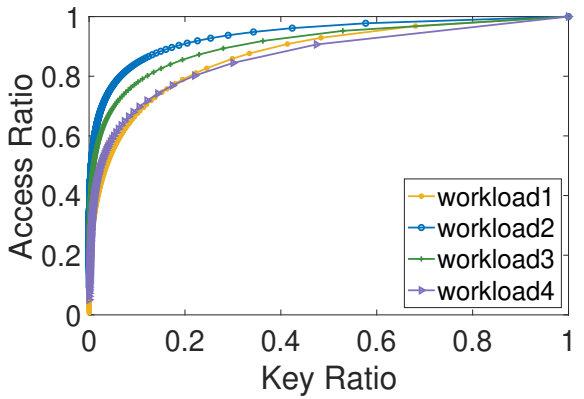
- Cache frequently accessed data for faster access and scalability
- The essential component in real world
- KV Store: ***Redis***; ***Memcached***



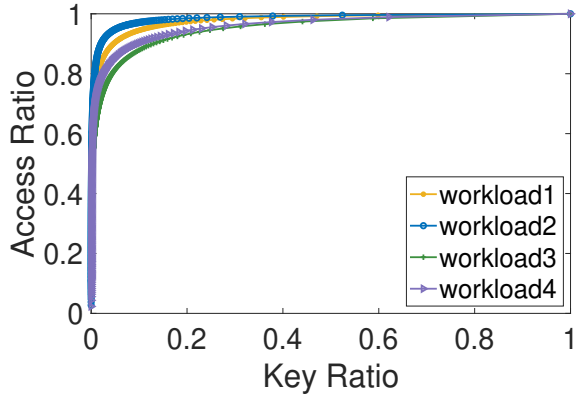
Hotspot Issue

- A small fraction of data which are frequently accessed in a highly-skewed workload
  - Skewed workload: the same key is queried in the extreme short time
  - Daily cases: ***50% of accesses for only 1% items***
  - Extreme cases: ***90%***





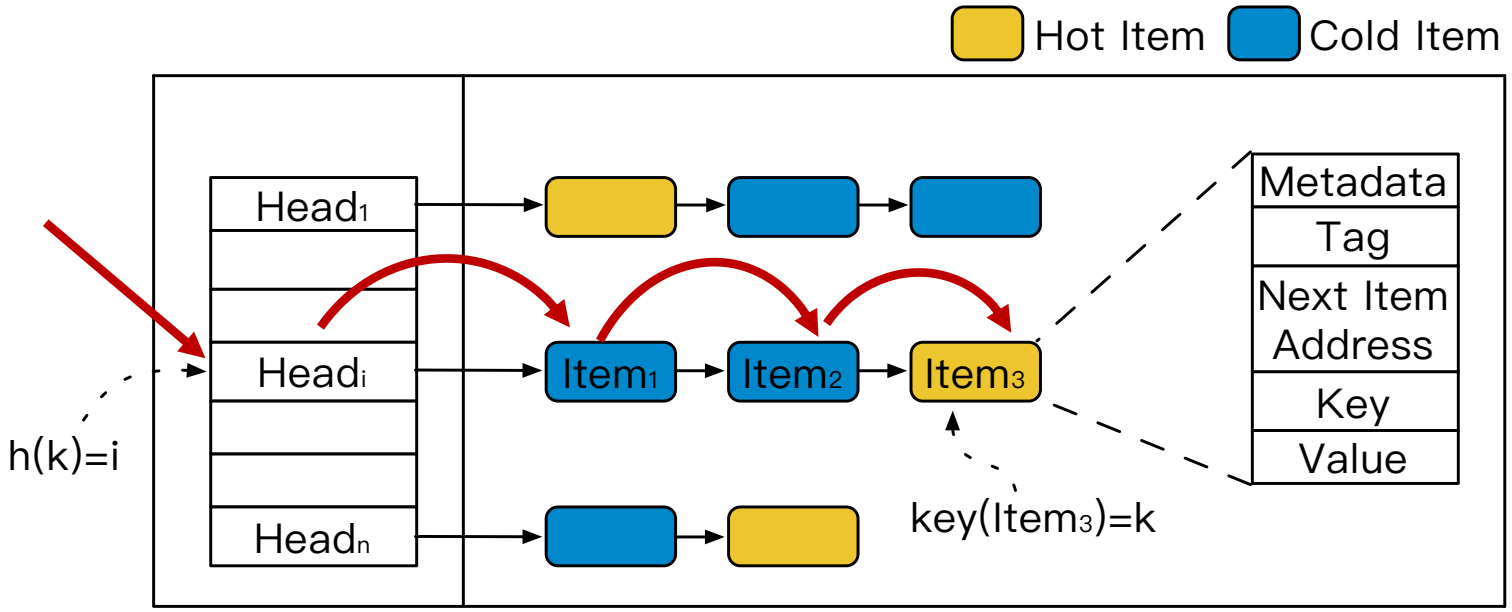
Daily distributions



Extreme distributions

Typical Hash Index Structure

- **Hash table with collision chain for each entry**
  - Item close to the tail requires more memory accesses (e.g., $Item_3$)
  - Hotter item should be close to the head pointer for faster read



Cannot be aware of hotspot

Main Idea

- **How to reduce memory access overhead for hotspots?**
  - Make hotspot ***closer to the head pointer***
- **How to identify the hotspot and its dynamic changes?**
  - Based on request frequency or access cost
- **How to provide efficient concurrent accesses at the same time?**
  - Using locks or non-locks schemes

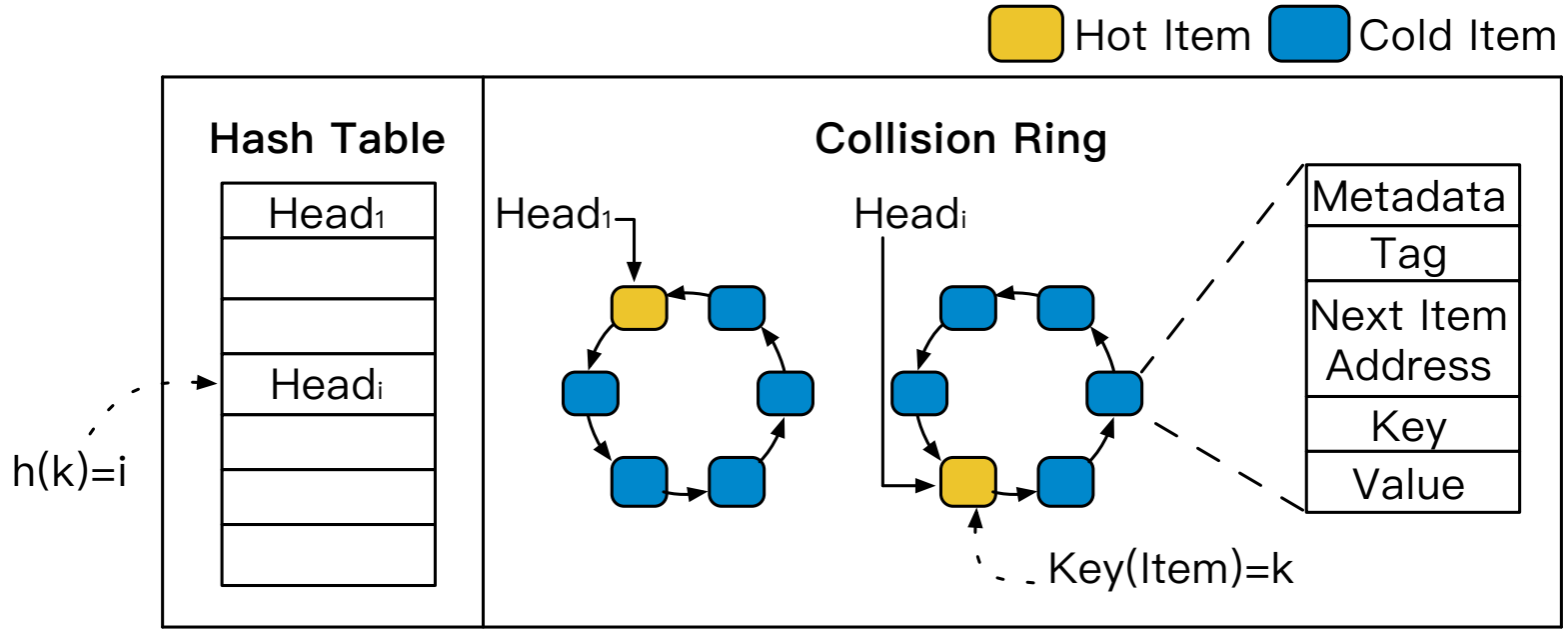
Contributions

- HotRing, an ***ordered-ring hash structure*** to enable hotspot-aware design
  - **Move the head pointer closer to the hot item**
  - Adopt a lightweight strategy to detect hotspot shifts at run-time
- Lock-free design for concurrency
  - Including hotspot shift detection, head pointer movement and ordered-ring rehash

HotRing

➢ The collision ordered-ring

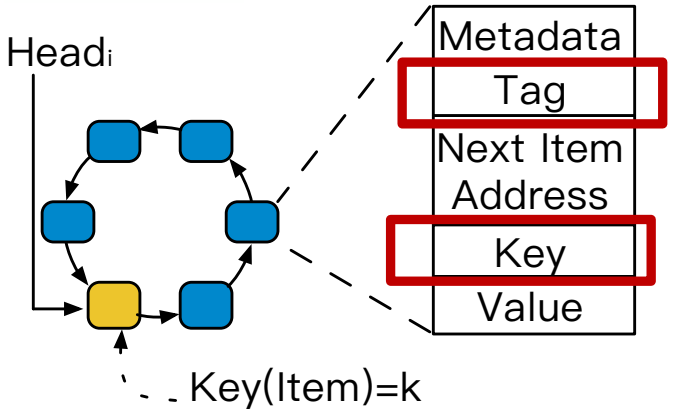
- Make the head pointer closer to the hot item
- Utilize ordered feature to faster lookup process (i.e., use tag and key)

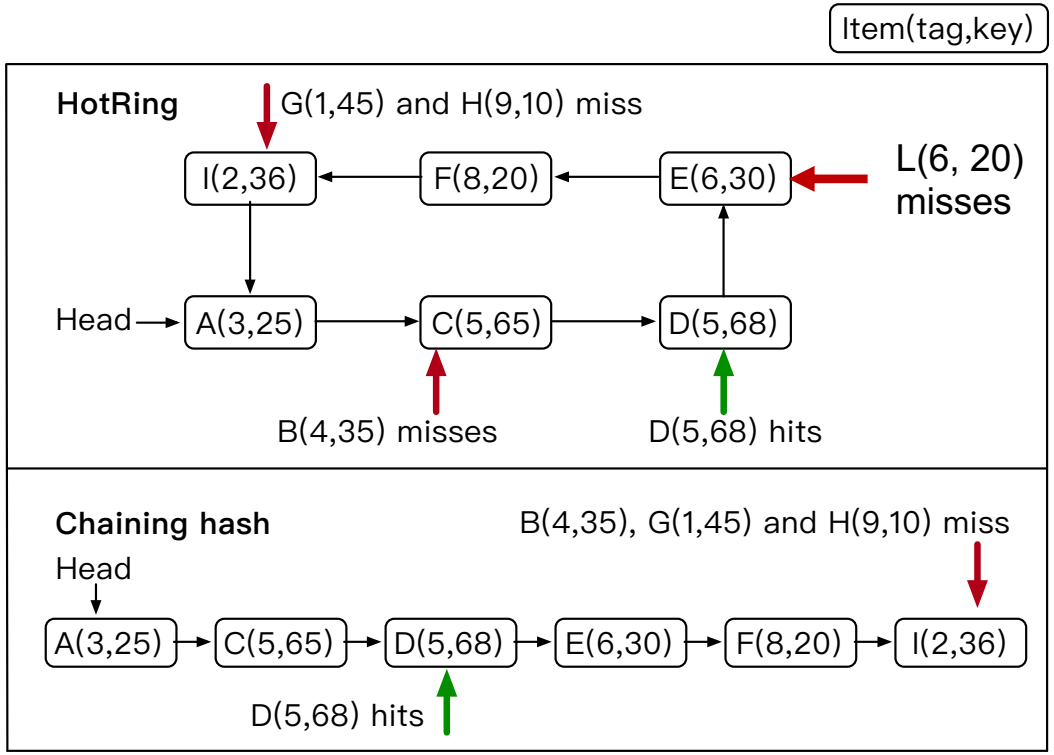


HotRing

➢ The collision ordered-ring

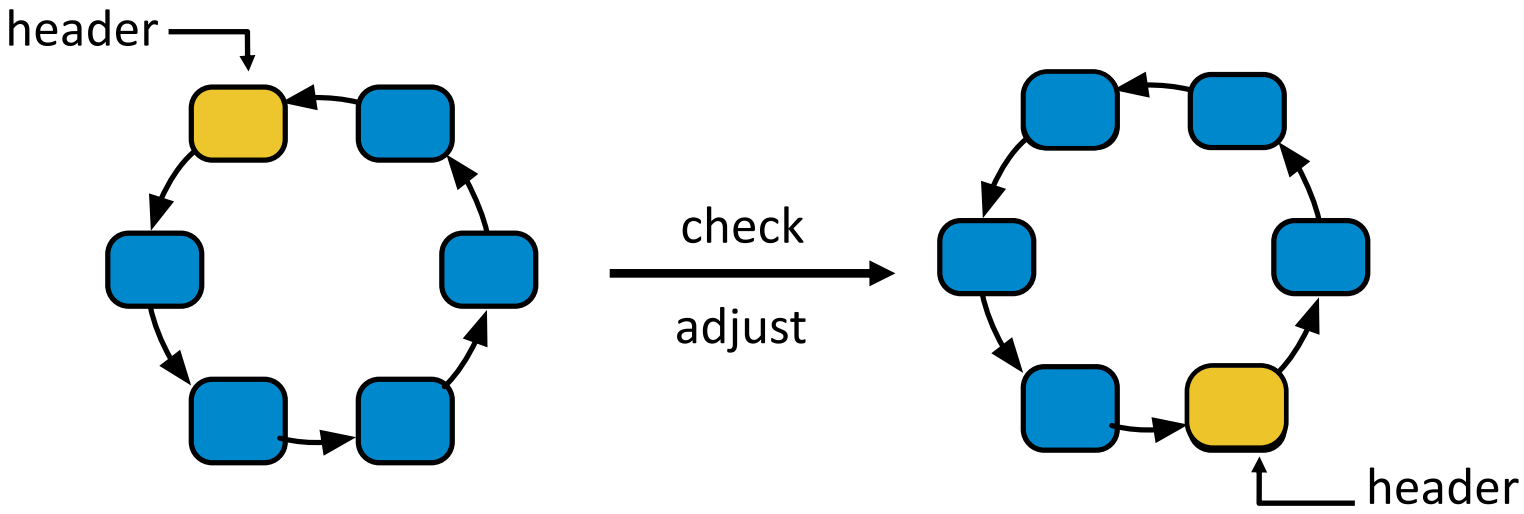
- $order_k = (tag_k, key_k)$
- Use the order to faster lookup process or termination





Random Movement Strategy

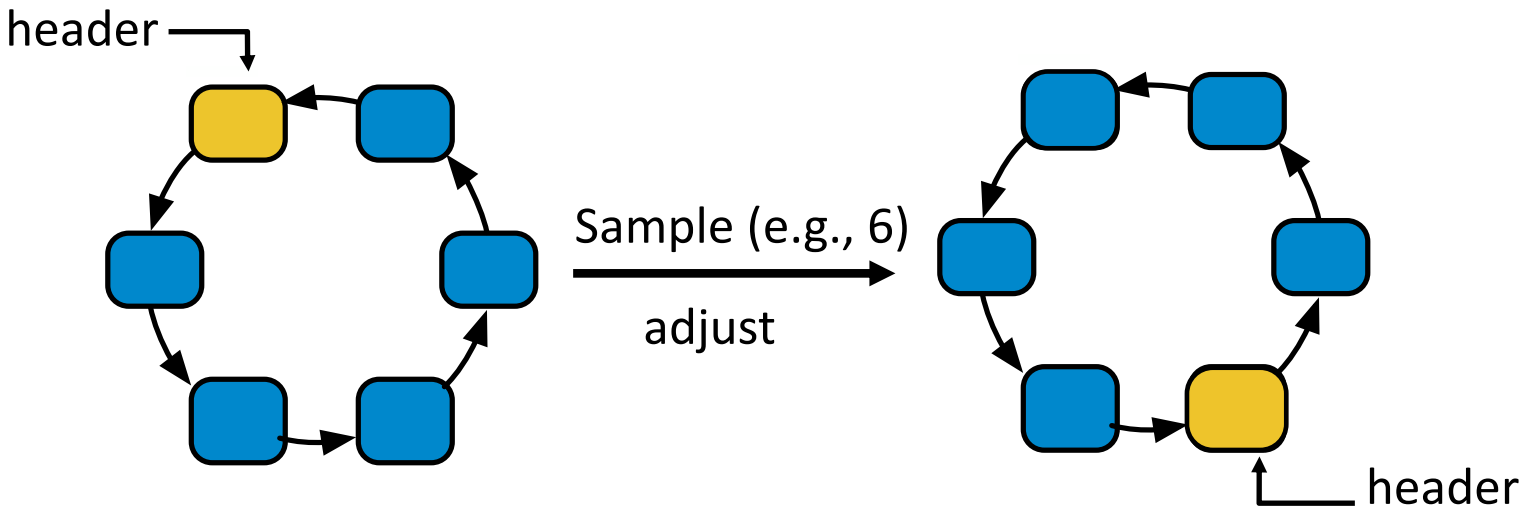
- **Based on random requests**
  - Adjust head pointer to the hotspot after every R requests (e.g., R = 5)
  - Work well for a single hotspot
  - Low accuracy for multiple hotspots but faster reaction

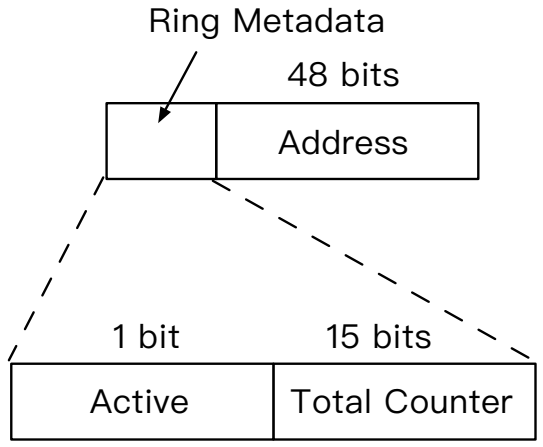


Reaction delay:
the time we detect the hotspot after its occurrence

Statistical Sampling Strategy

- **Based on sampling**
  - Launch a new sampling after every R requests
  - Record counts in the head pointer as well as in the item
  - (The average memory access cost = count * distance)$_{min}$



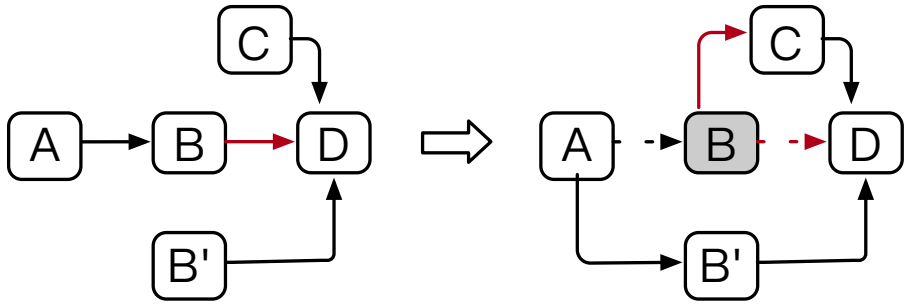


Head pointer format

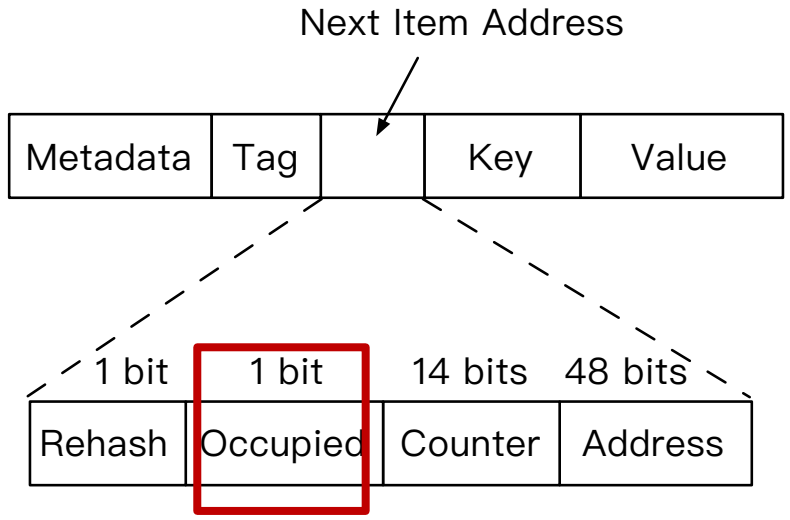
Lock-free Operations

➢ Adopt lock-free to support concurrency

- HotRing supports hotspot shift detection, head pointer movement and ordered-ring rehash
- Utilize occupied bit



Concurrency issue

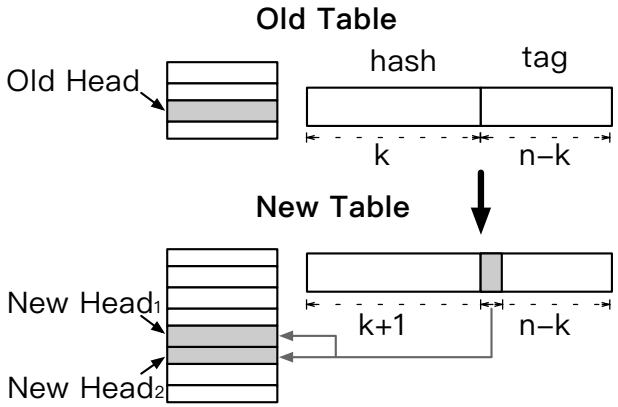


Item format

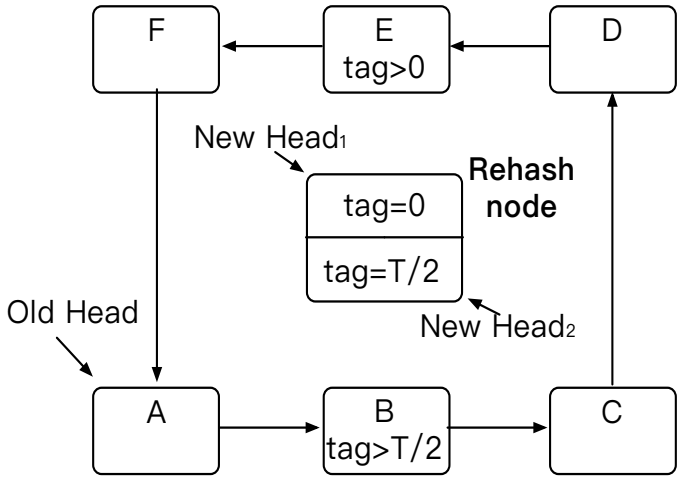
Lock-free Rehash

➢ Rehash to increase data volume

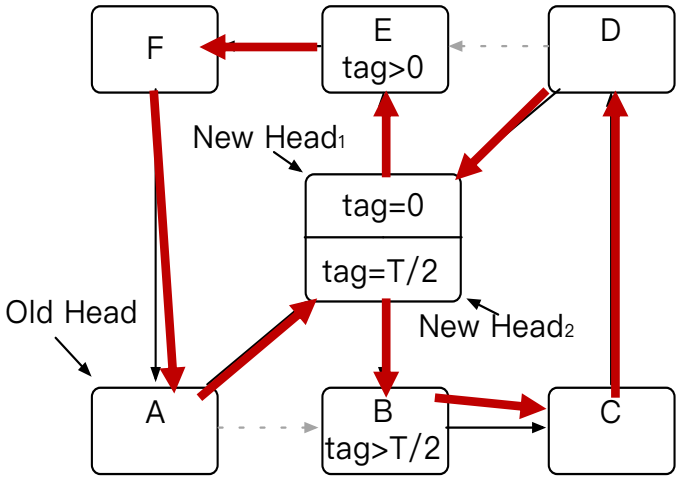
- Access overhead is used to trigger rehashing instead of length for hotspot-awareness
- Tag range: [0, T/2) and [T/2, T) , $T = 2^{(n - k)}$



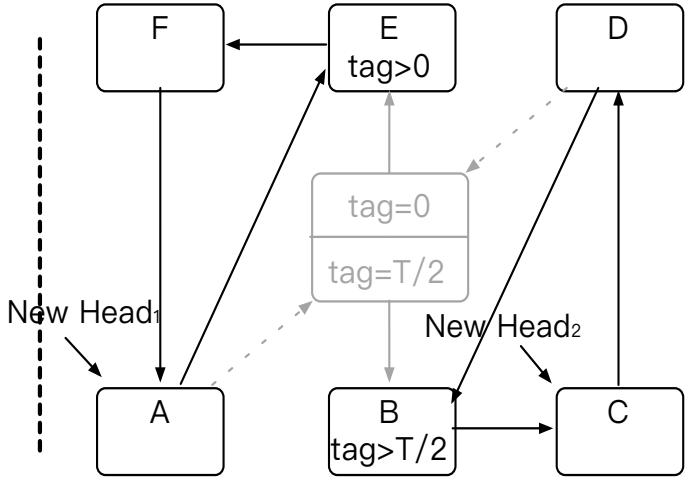
Share highest bit of tag



Initialization



Split



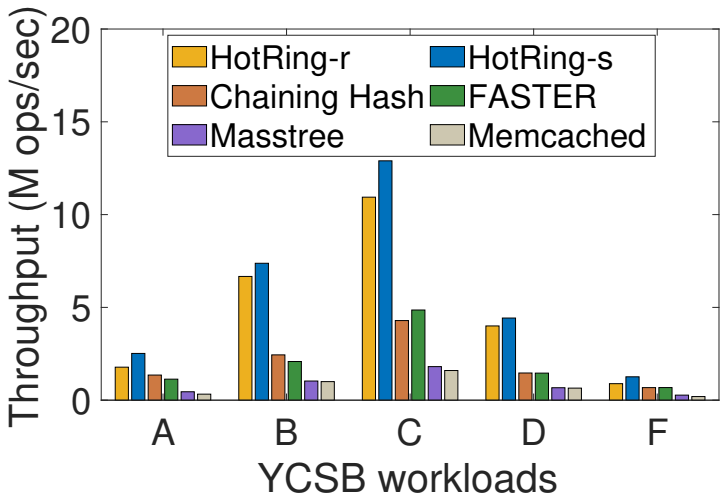
Deletion

Experimental Setup

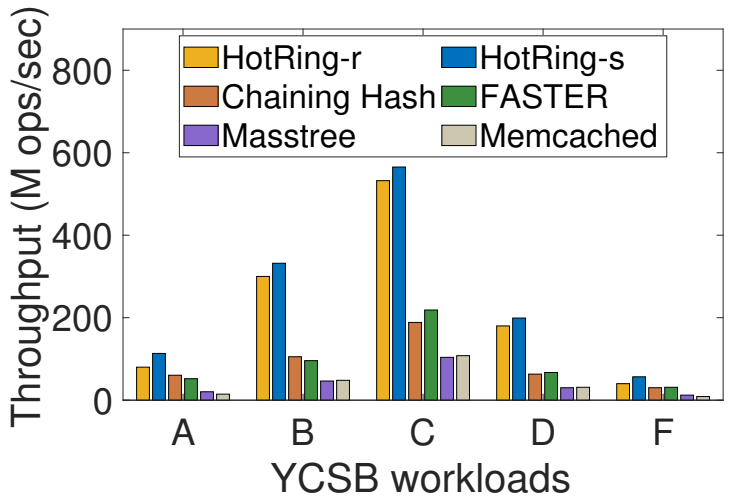
- Datasets:
  - YCSB workloads (Except for E)
  - Key-value size: 8 bytes
  - 250 millions of loaded keys
- Comparison:
  - **Baseline**:
    - Lock-free chaining hash & FASTER
    - Masstree & Memcached
  - **HotRing-r**
  - **HotRing-s**

System	CentOS 7.4 OS with Linux 3.10 kernel
CPU	2.50GHz Intel Xeon(R) E5-2682 v4 * 2 (64 thread in total)
Cache Alignment	64B
Main Memory	32GB 2133MHz DDR4 DRAM * 8

Throughput



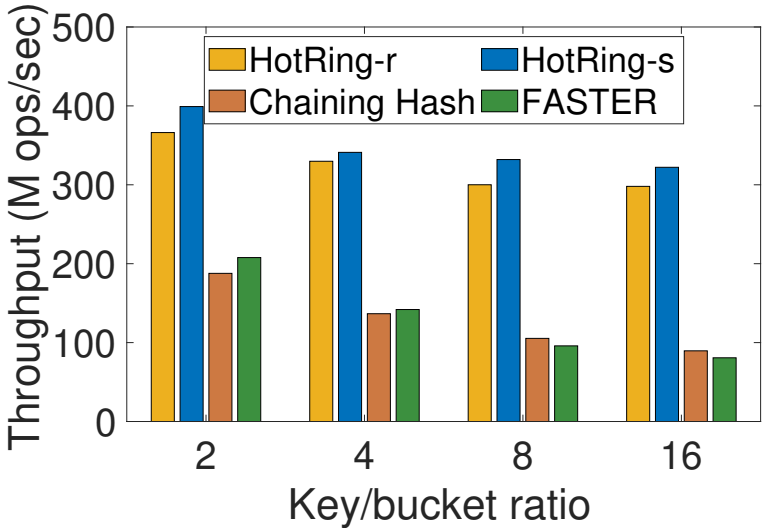
Single thread



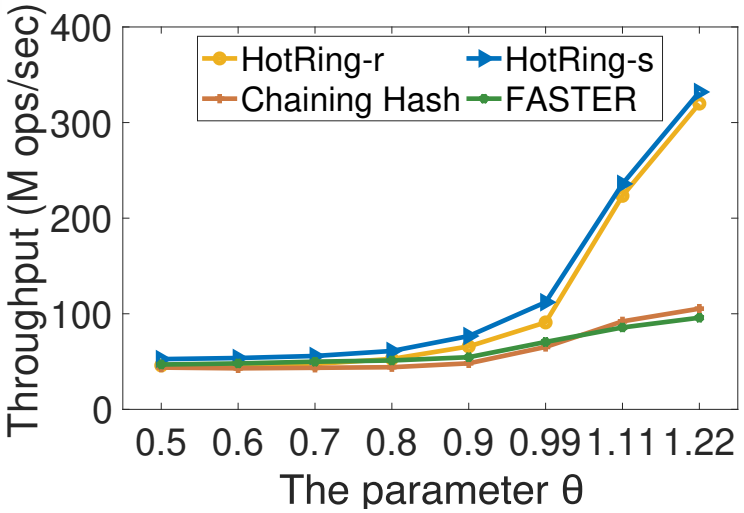
64 threads

- HotRing outperforms other approaches by **2.10X-7.75X**
  - Especially for B(95% read) and C(100% read)
  - HotRing-r is about 7% worse than HotRing-s

Overall Performance



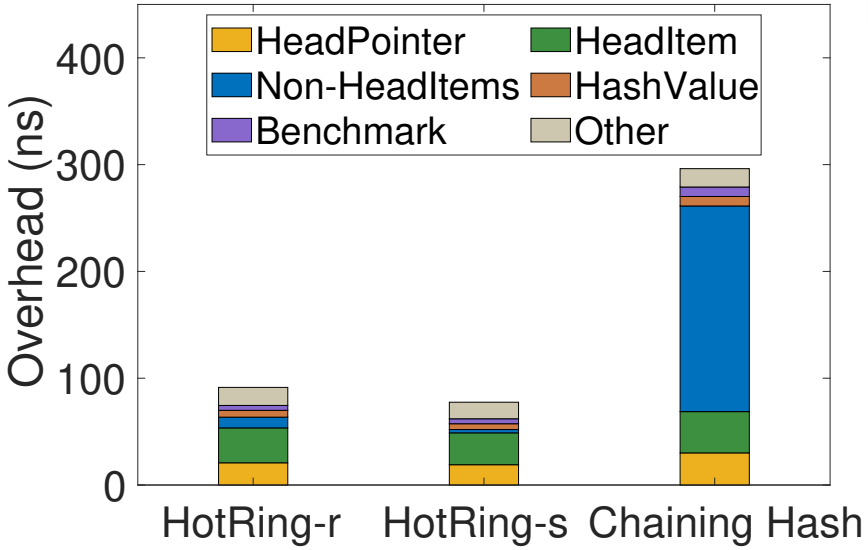
Impact of chain length



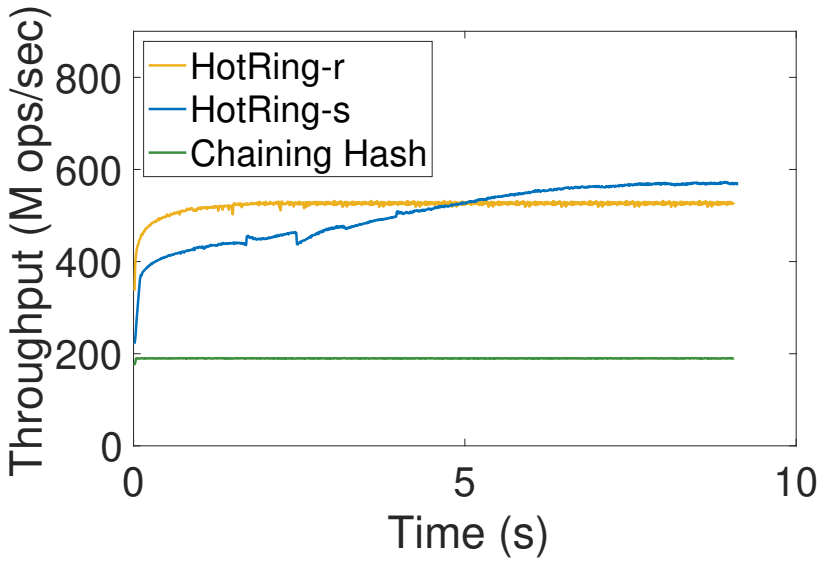
Hotspot awareness

- HotRing's throughput increases from 1.3X to 3.91X compared with FASTER since it puts hot items close to the head pointer (*less memory overhead*)
- HotRing improves up to 5X when the workload is more skewed

Micro-benchmark



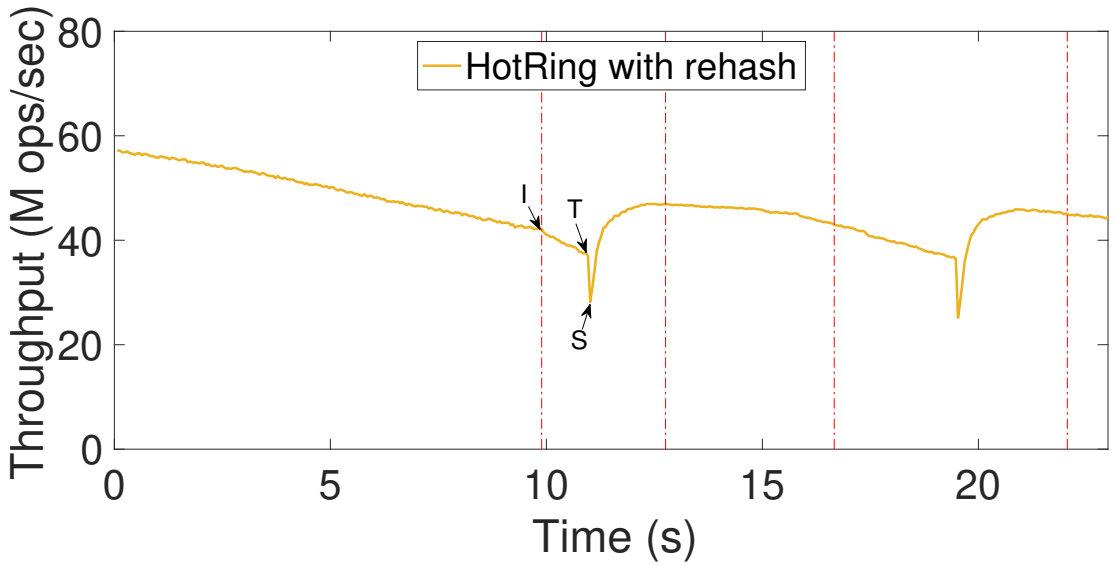
Breakdown cost (Θ = 1.22)



Reaction delay(Θ = 1.22)

- Non-HeadItems: HotRing-s has higher hotspot identification accuracy since more hotspot items are detected
- HotRing-r has faster reaction than HotRing-s (2 seconds to reach stable state)

Rehash Performance



Two consecutive rehash process (Θ = 1.22)

I : Initialization

T: Transition (ensure that all accesses from the old table have finished)

S: Split

- Split stage sets active flag to false, which blocks any requests for this collision ring.
- The short-term drops during rehash are because of the temporary lack of hotspot awareness when the new hash table starts to work.

Conclusion

- HotRing
  - Hash Index with the ordered collision ring
  - Utilize the pair of tag and key for ordering and fast lookup
  - Solve the hotspot shift issue with hotspot-awareness strategy
- Concurrent support for HotRing
  - Adopt lock-free method
  - Support lock-free rehash operation