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'20

Background

- **In-memory key-value stores(KVSes)**
  - A storage system with keys and values in memory, the following are the operations it supports:
    - Insert(key, value)
    - Delete(key)
    - Find(key)
  - Samples:
    - **Hash table**
    - Red-black tree
    - …

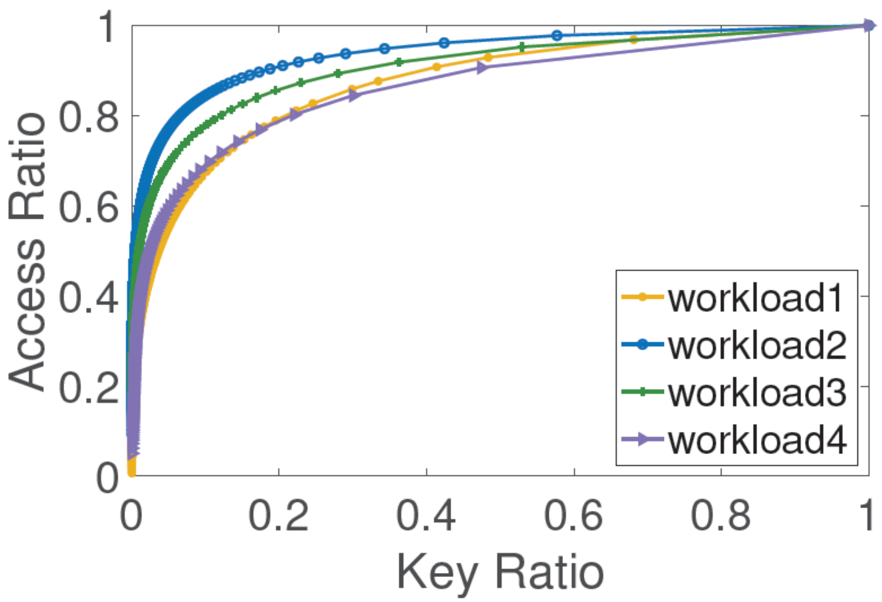
Background

➢ Hotspot

- Observation
  - The top 10% of the most frequently accessed keys account for 80% of the total;
  - The last 60% accounted for only about 10% of the total number of accesses.

- Definition
  - A small portion of items that are frequently accessed in a highly-skewed workload;

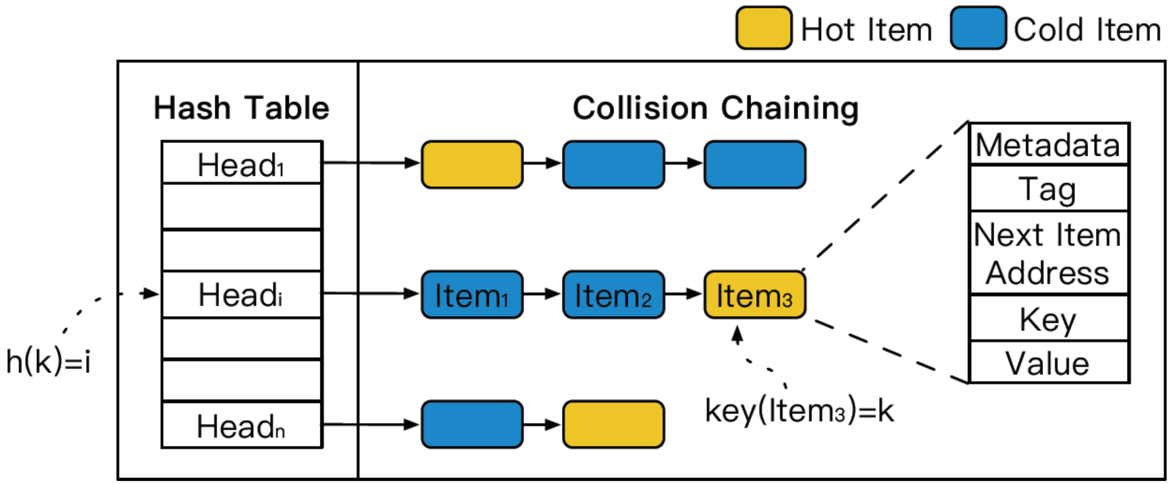
  - High-frequency access key → Hot item.
  - Low-frequency access key → Cold item.



Problem

➢ Hash table

- Head table;
- Collision Chaining(open hash).



➢ Access time

- The righter the position of the item in the collision chain, the more memory accesses.

➢ Extreme case

-  → Longer average access time;
- 

→ Shorter average access time.

Problem

- **Average access time**

$$\bar{T} = \Sigma t_i f_i$$

- **Problems of existing solutions**
  - Common structure: Unable to detect hot items;
  - Cache: capacity limit.

Idea

- **Shorten the access time of hot items**

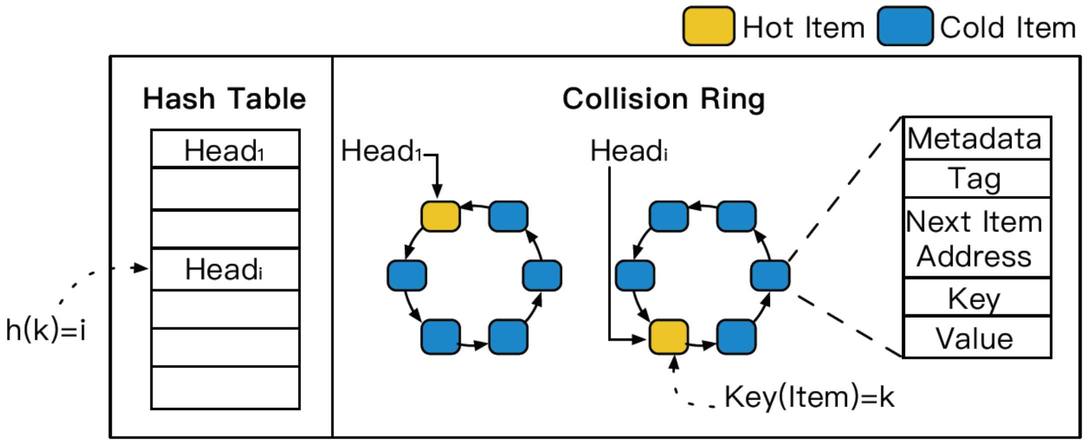
  - **HotRing**: Make hot items closer to the head table
    - Hot item location detection;
    - Dynamically head pointer movement.

  - **Lock-free structure**: improve concurrency performance

HotRing

➢ Structure

- Chain → Ordered ring;
- Head pointer always points to hot item;



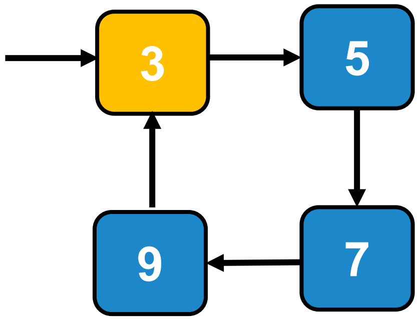
➢ Why

- Head pointer can freely move to another item without modifying the collision chain;
- Compared with the previous design (**straight chain**), there is no obvious performance loss (Doubtful).

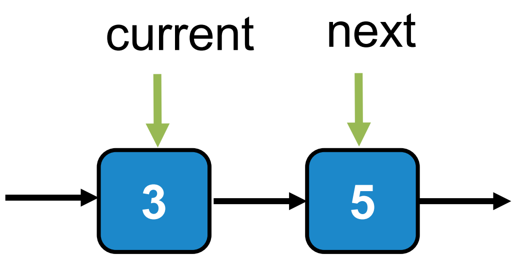
HotRing

- **Avoid infinite traverses in the ring**
  - Next item is null → **Not available under ring structure**
  - Pointed by header →**Not available under multi-threading**

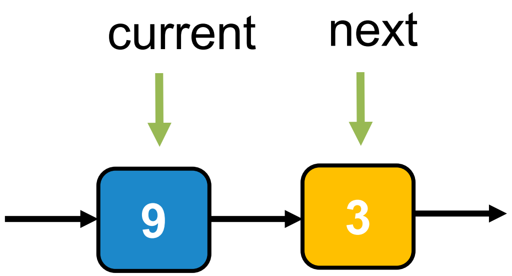
  - **Comparing adjacent keys(x is the number to be searched)**



Sample Ring
Min = 3,max = 9



X = 4(min < x < max)
Cur < x < next



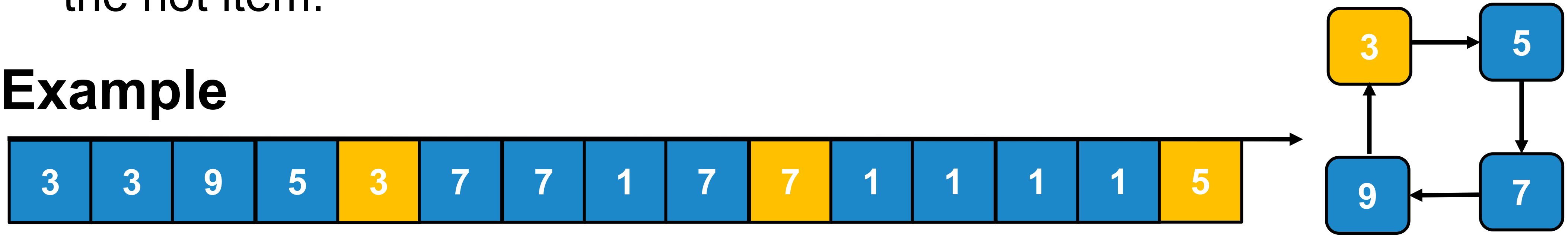
X = 11 or x = 1(X > max or x < min)
Cur, next < x or cur, next > x

Hotspot detection

- **Overview**
  - Triggered under certain conditions (for example, after receiving R requests);
  - Calculate and find new hot item;
  - Move head pointer to new hot item.
- **Tips**
  - The scope of hot spot detection **is based on the ring**, not the entire hash table;
  - The item currently pointed to by head is regarded as a hot item.

Random Movement

- **Triggering condition**
  - After every R requests.
- **Calculation**
  - The **last key requested to be accessed** in the previous round is used as the hot item.
- **Example**



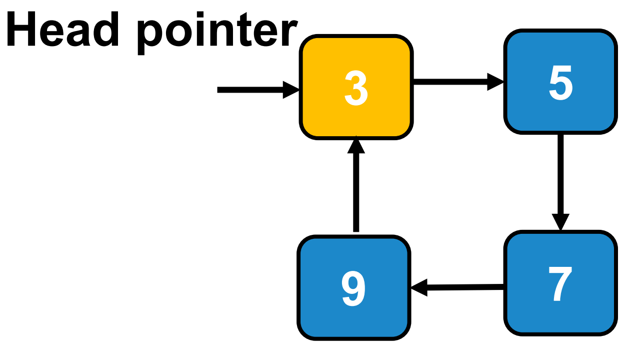
- **Evaluation**
  - Low latency, low accuracy
  - The effect depends on the workload
  - Data fluctuations will cause frequent movement of the head pointer

Statistical Sampling

- **Triggering condition**
  - After every R requests(a round).
- **Calculation**
  1. Get the **last key requested to be accessed** in the previous round;
  2. If the key is same with current hot item, nothing to do;
  3. Else, start a sampling (Count the next N items requested for access, N is the size of collision ring);
  4. Calculate hot items based on statistical data.
  5. Move head to current hot item.

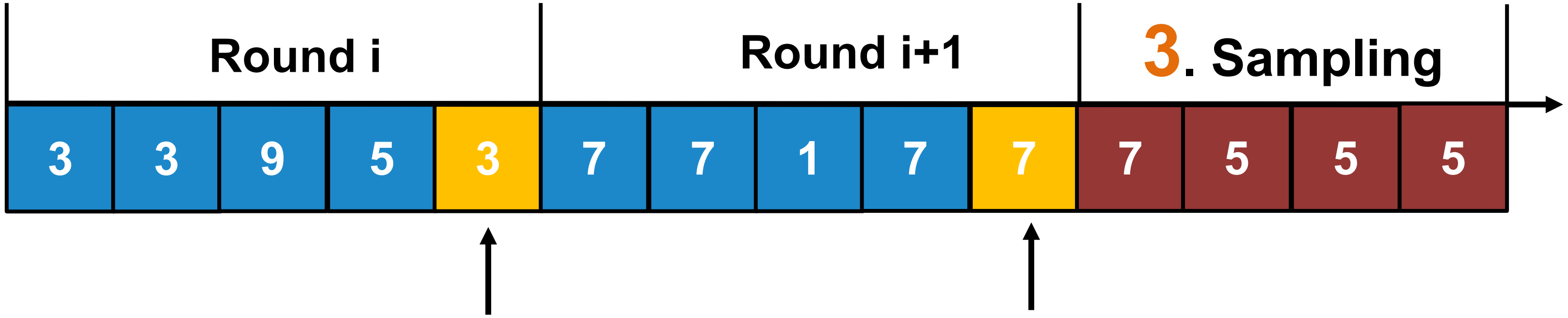
Statistical Sampling

➢Workflow



Current collision ring

Request sequence



1. Same with current→ nothing to do

2. Hot item changed

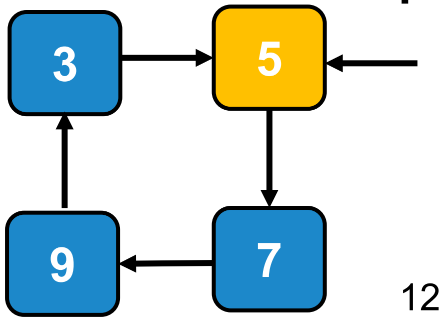
4. Calculate each item's cost(Take item 3 as an example):

Total cost = 0 * 0 + 3 * 3 + 1 * 3 + 0 * 0 = 6

Item	Distance(to 3)	Frequency	Cost
3	0	0	0
5	1	3	3
7	3	1	3
9	3	0	0

Item	Total cost
3	6
5	1
7	9
9	9

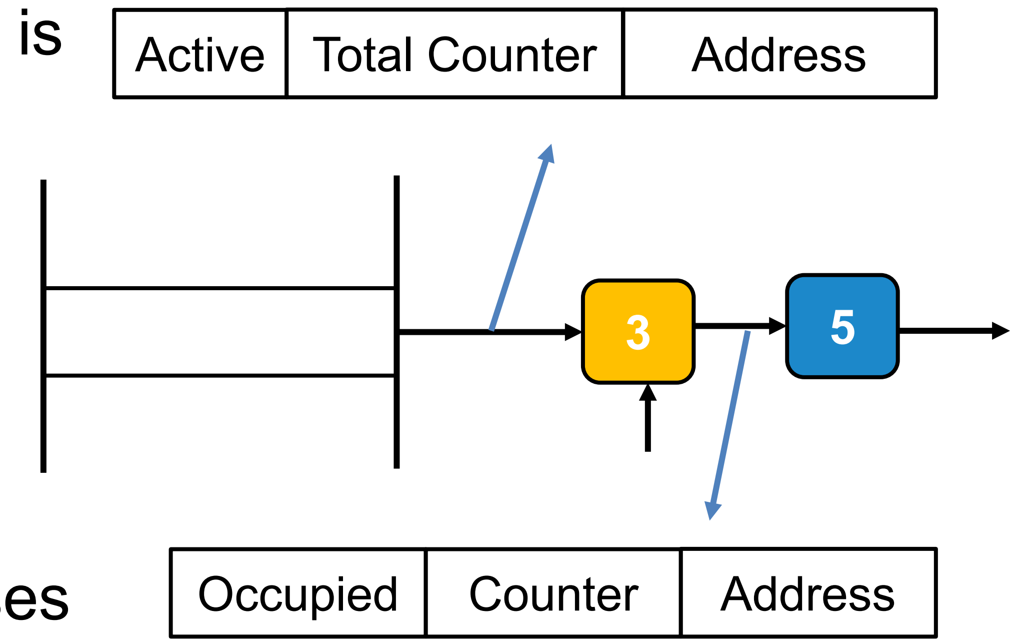
5. Move Head pointer



Data structure

➢ Header pointer

- **Active** → Determine if a ring is in sampling;
- **Total Counter** → Determine if the sampling is over.
- **Address** → Address of hot item.



➢ Item Pointer

- **Occupied** → Check if it is occupied by other threads;
- **Counter** → Record the frequency of accesses during the sampling process.
- **Address** → Address of next item.

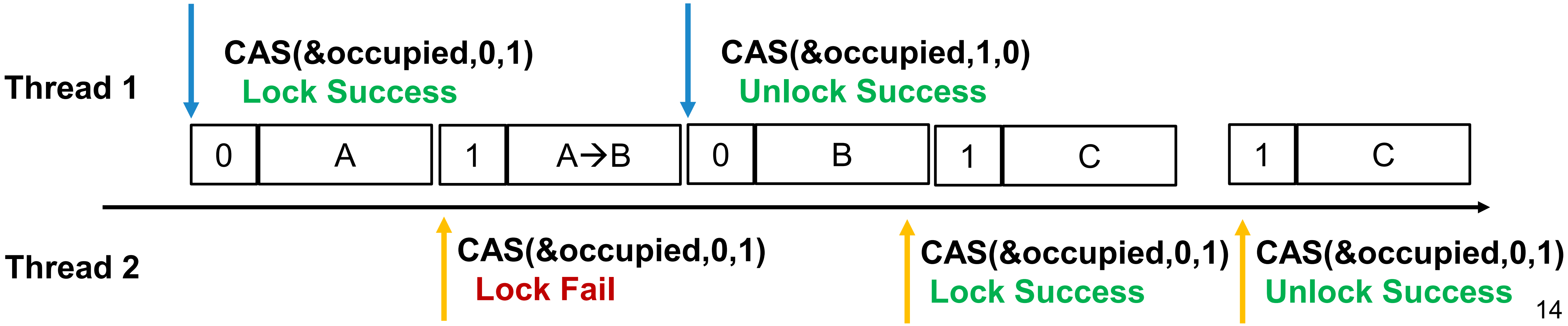
Concurrency support

➢ Problem

- Pointer invalidation occurs when updating the ring structure;

➢ Solution

- **CAS**(compare and swap, atomic) and the occupied bit form a simple mutex;
- This simple mutex is used to **ensure data consistency** when changing Pointers.



Evaluation

➢ Test and comparison items

- HotRing-r (use random movement);
- HotRing-s (use statistical sampling);
- Chaining Hash(lock-free);
- FASTER (designed for point lookups and heavy updates.);
- Masstree;
- Memcached(with lock).

➢ Dataset

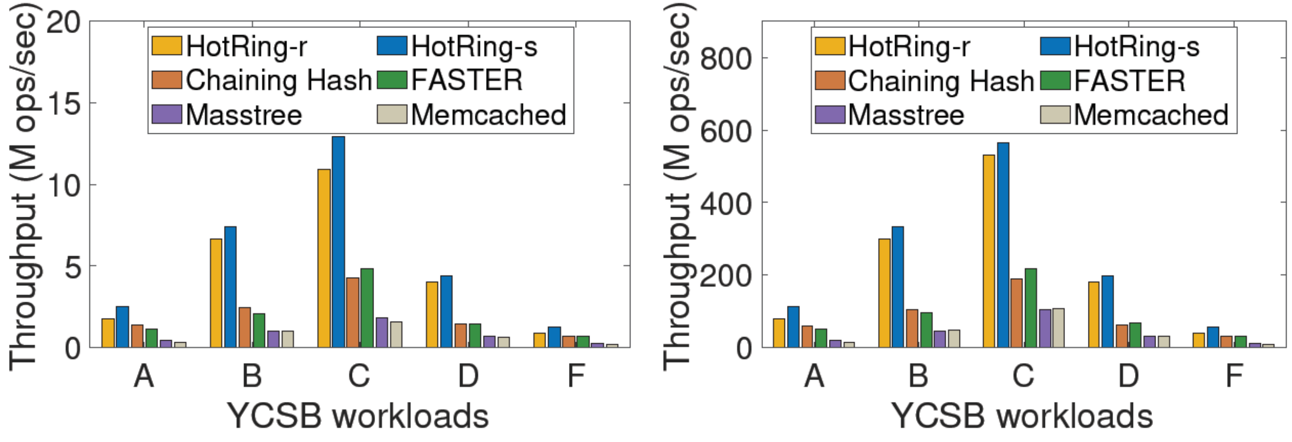
- YCSB(zipfian distribution).

The larger the θ, the larger the hot issues in the data set.

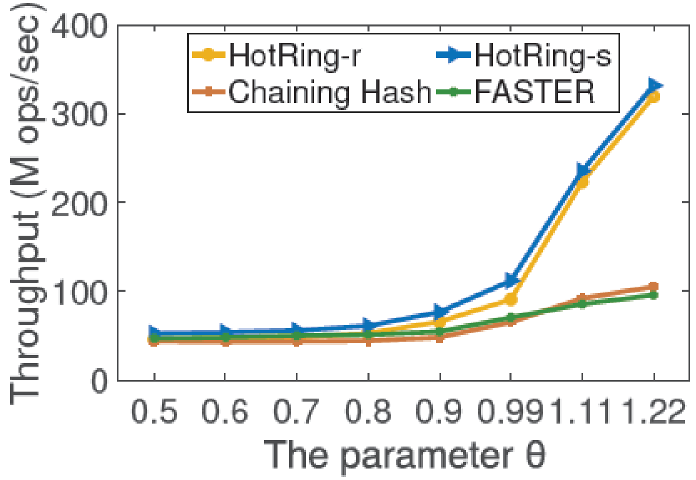
θ \ α	1%	10%	20%	30%	40%	50%
0.5	9.9%	31.6%	44.7%	57.7%	63.2%	70.7%
0.7	24.9%	50.0%	61.6%	71.9%	75.9%	81.2%
0.9	57.3%	76.2%	82.2%	86.9%	89.9%	92.2%
0.99	75.1%	87.4%	91.2%	93.4%	94.9%	96.2%
1.11	91.7%	96.4%	97.6%	98.2%	98.7%	99.0%
1.22	97.8%	99.2%	99.5%	99.6%	99.7%	99.8%

Evaluation

- **Throughput**



(a) Single thread (b) 64 threads

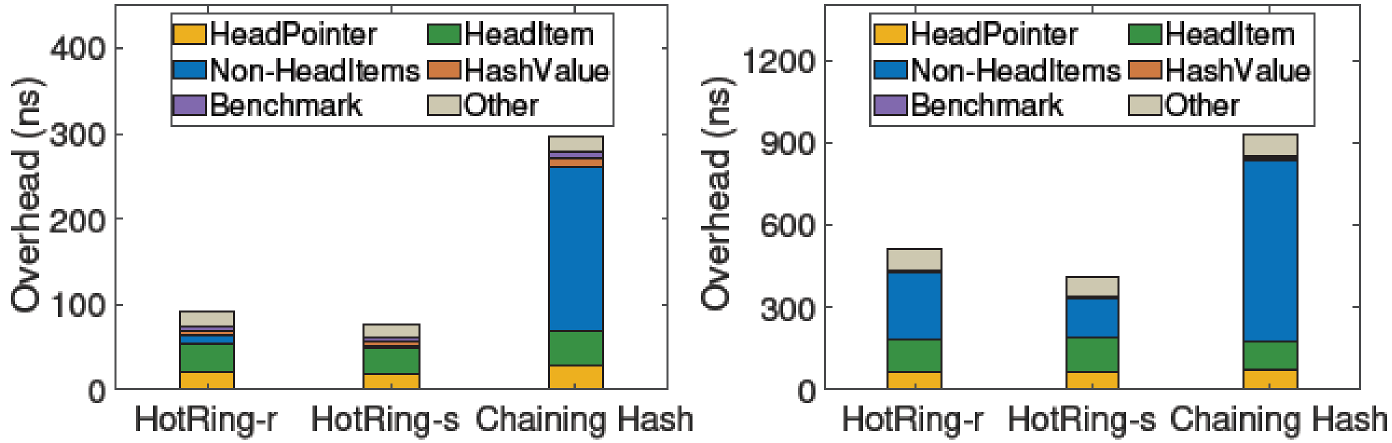


Throughput of single thread and 64 threads

Throughput under different θ

Evaluation

➢ **Break-down cost**



(a) $\theta = 1.22$ (b) $\theta = 0.99$

The average collision chain length is 8

Analysis

- HotRing's advantage is in accessing **non-head items;**
- The rest of the performance is essentially the same.

Conclusion

- **Problem**
  - Existing In-memory kV does not solve hotspot issues.
- **Idea**
  - Minimize access time for hot items.
- **Design**
  - HotRing: The head pointer of the collision chain always points to the hot item;
  - Lock Free operation: CAS and flag bit.