OSCA: An Online-Model Based Cache Allocation Scheme in Cloud Block Storage Systems

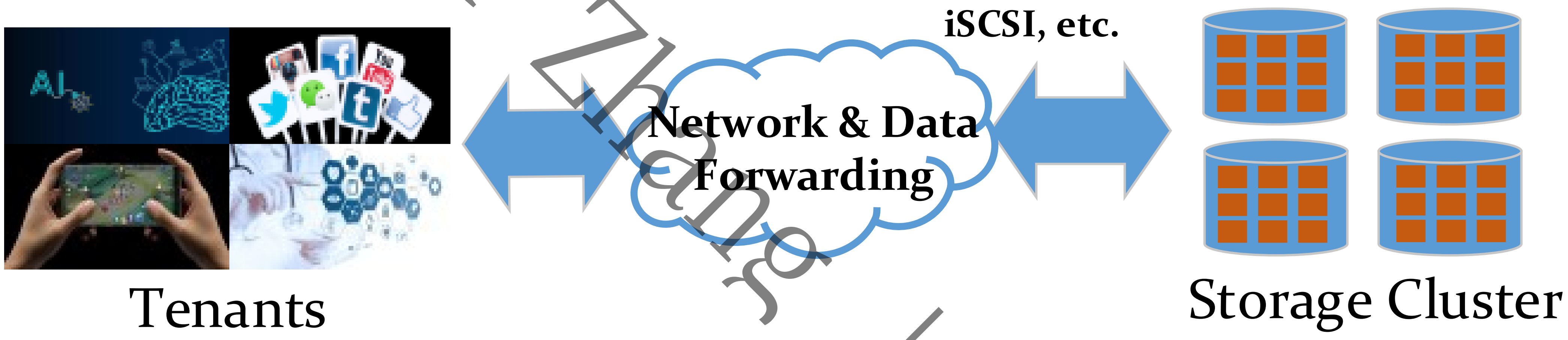
Yu Zhang[†], Ping Huang[†§], Ke Zhou[†], Hua Wang[†], Jianying Hu[‡], Yongguang Ji[‡], Bin Cheng[‡]
[†]Huazhong University of Science and Technology
[†]Intelligent Cloud Storage Joint Research center of HUST and Tencent
[§]Temple University
[‡]Tencent Technology (Shenzhen) Co., Ltd.

USENIX Annual Technical Conference 2020

Agenda

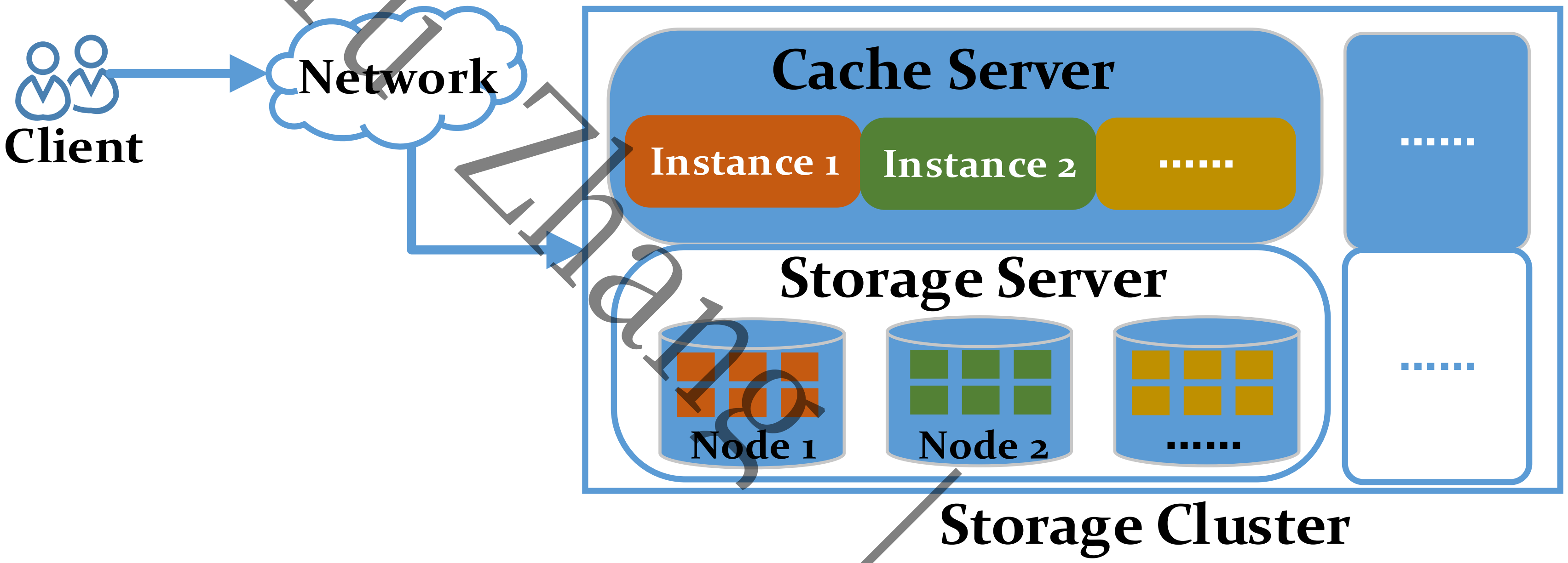
- **Research Background**
  - ➢ Cloud Block storage (**CBS**)
- **Motivation**
- **OSCA System Design**
  - ➢ Online Cache modeling
  - ➢ Search for the optimal solution
- **Evaluation Results**
- **Conclusion**

Background



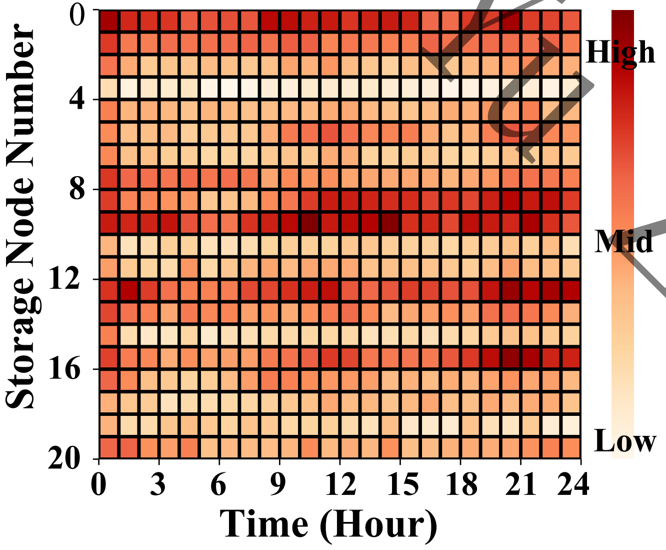
- To satisfy the rigorous performance and availability requirements of different tenants, **cloud block storage (CBS) systems** have been widely deployed by cloud providers.

Background

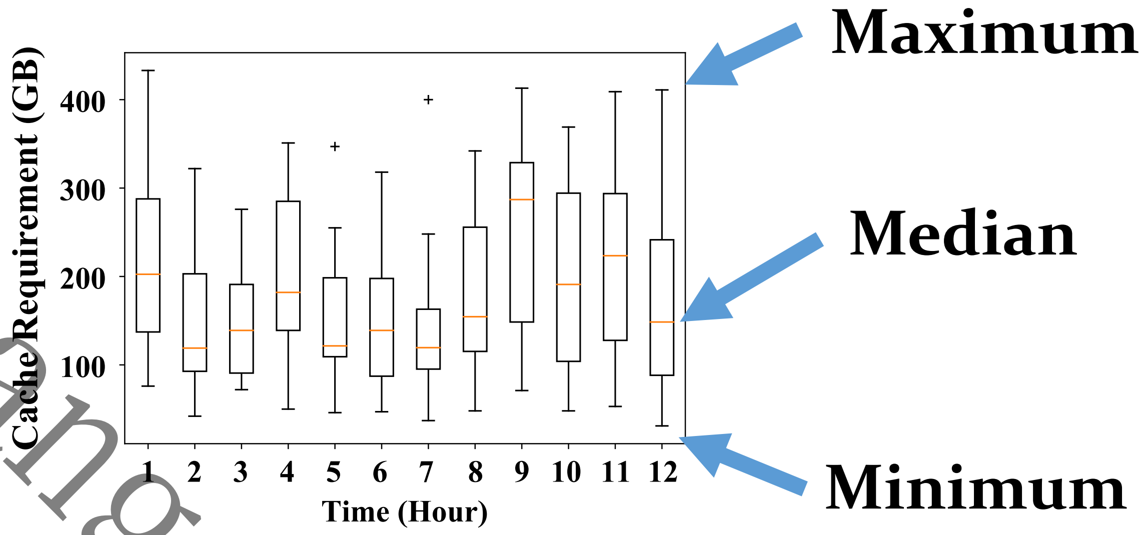


- Cache servers, consisting of multiple cache instances competing for the same pool of resources.
- **Cache allocation scheme** plays an important role.

Motivation



(a)



(b)

- The highly-skewed cloud workloads cause uneven distribution of hot spots in nodes. → figure (a)
- The currently used even-allocation policy is inappropriate for the cloud environment and induces resource wastage. → figure (b)

Motivation

To improve this policy via ensuring more appropriate cache allocations, there have been proposed two broad categories of solutions.

- **Qualitative methods** based on intuition or experience.
- **Quantitative methods** enabled by cache models typically described by Miss Ratio Curves (MRC).

Motivation

To improve this policy via ensuring more appropriate cache allocations, there have been proposed two broad categories of solutions.

- **Qualitative methods** based on intuition or experience.
- **Quantitative methods** enabled by cache models typically described by Miss Rate Curves (MRC).

We propose OSCA, an Online-Model based Scheme for Cache Allocation

Main Ideas

Online Cache Modeling

- Obtain the *miss ratio curve*, which indicates the miss ratio corresponding to different cache sizes.

Optimization Target Defining

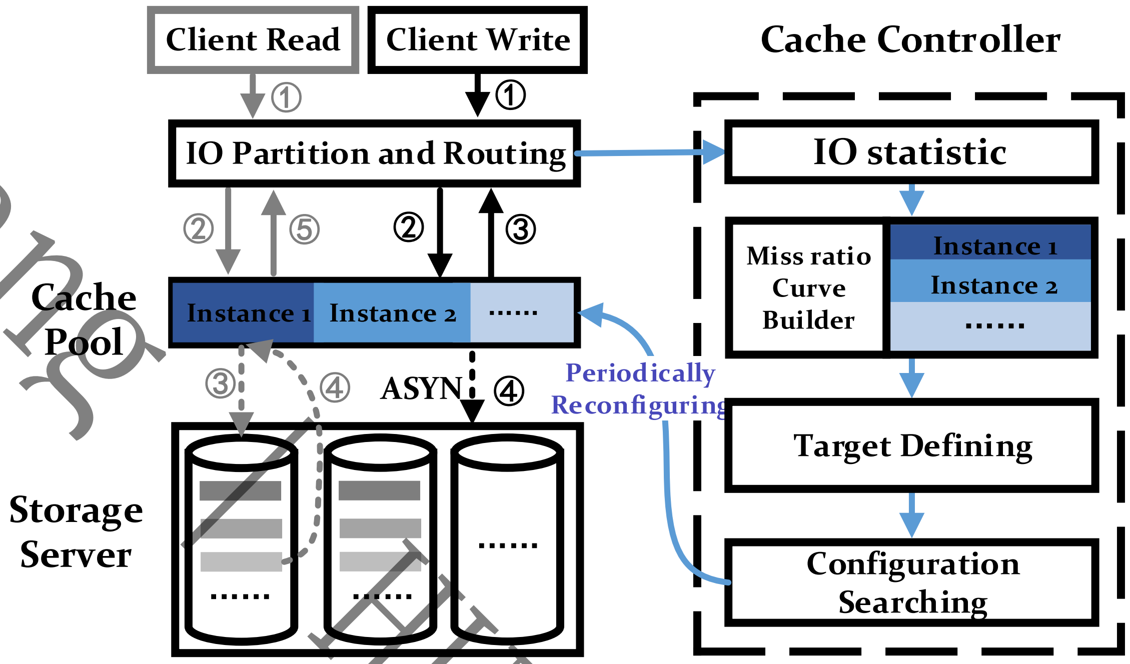
- Define an optimization target.

Searching for Optimal Configuration

- Based on the cache modeling and defined target mentioned above, our OSCA searches for the optimal configuration scheme.

Cache Modeling

- **Cache Controller**
  - IO processing & Obtain Miss Ratio Curve.
  - Optimization Target.
  - Configuration Searching.
- **Periodically Reconfigure.**

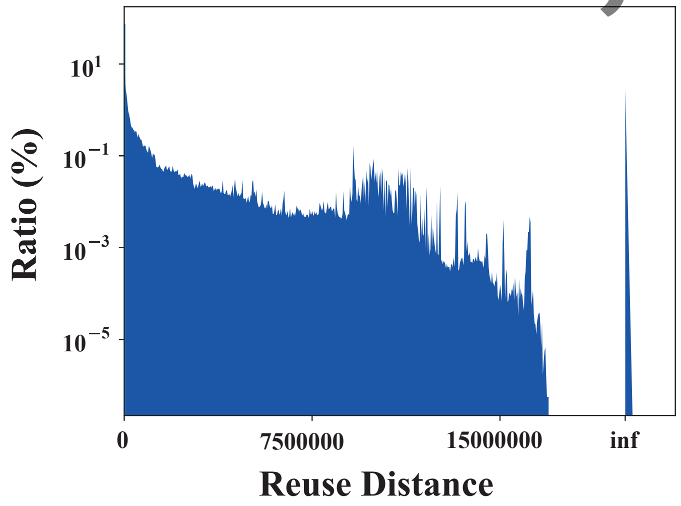


Cache Modeling (cont.)

Online Cache Modeling

- **Obtain the miss *ratio curve*, which describes the relationship between hit ratio and cache size.**

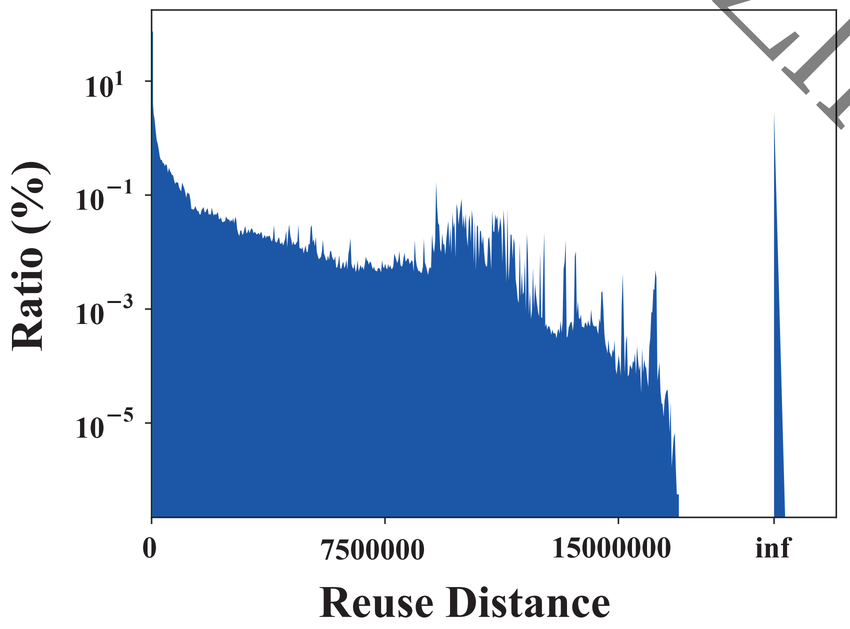
- The hit ratio of the LRU algorithm can be calculated from the **discrete integral sum** of the reuse distance distribution (from zero to the cache size).



$$hr(C) = \sum_{x=0}^{C} rdd(x)$$

Cache Modeling (cont.)

- **Reuse Distance**



- The reuse distance is the number of **unique data blocks** between two consecutive accesses to the same data block.
  - A**BCD**BDA
  - Reuse Distance of block **A** = 3
- A data block can be hit in the cache only when its reuse distance is **smaller than** the cache size.
- The hit ratio of the LRU algorithm can be calculated from the **discrete integral sum** of the reuse distance distribution (from zero to the cache size).

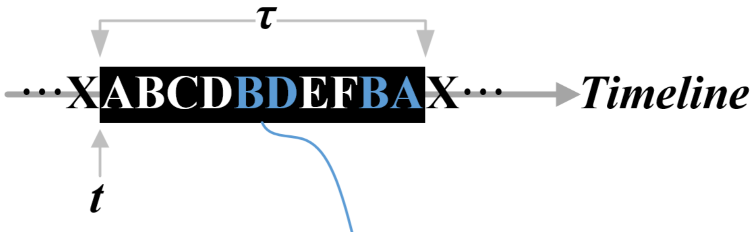
$$hr(C) = \sum_{x=0}^{C} rdd(x)$$

Reuse Distance

- However, obtaining the reuse distance distribution has an **O(N ∗ M)** complexity.
- Recent studies have proposed various ways to decrease the computation complexity to **O(N ∗ log(n))**. SHARDS further decreases the computation complexity by sampling method.
- We propose **Re-access Ratio based Cache Model (RAR-CM),** which does not need to collect and process traces, which can be expensive in many scenarios. RAR-CM has an O(1) complexity.

Re-access Ratio

- Re-access ratio **(RAR)** is defined as the ratio of the re-access traffic to the total traffic during a time interval τ after time t.
- RAR can be transferred to Reuse distance.
  - ABCD**BD**EF**BA** → RAR(t,τ) = 4 / 10 = 40%
  - Reuse Distance of Block **X** = Traffic(t,τ) * (1 - RAR(t,τ)) = 6
- So we can get the reuse distance distribution by obtaining the RAR.

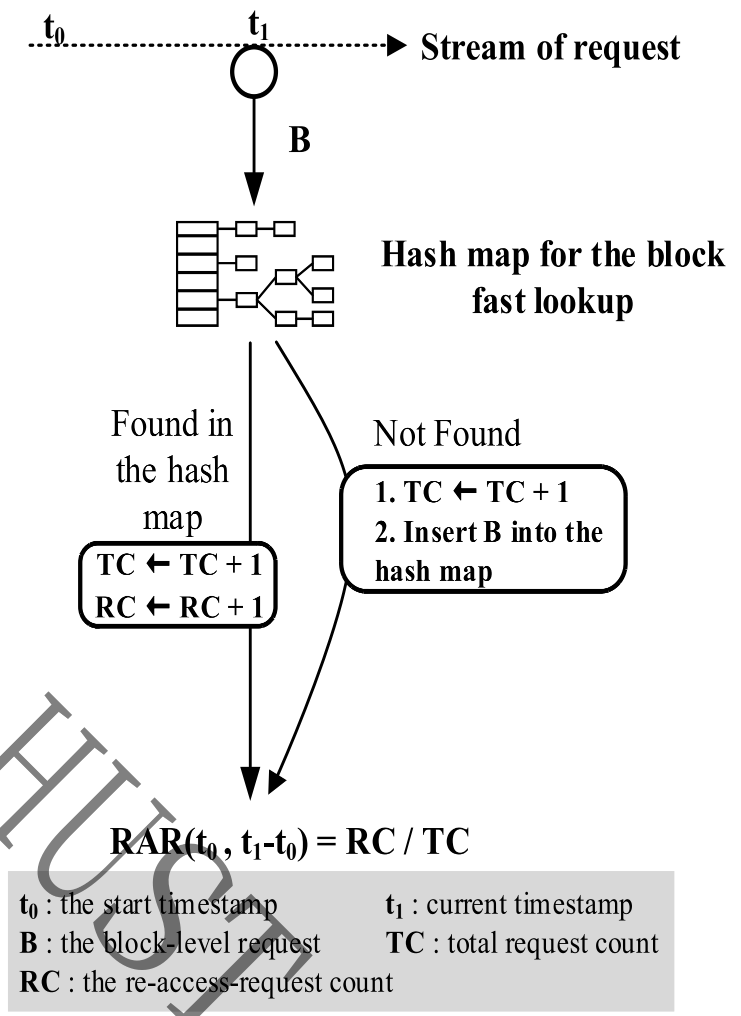


RAR is defined as a ratio of the re-access traffic to the total traffic, so ***RAR(t,τ)*** = 4/10 = 40%.

Obtain Re-access Ratio

- RAR(t_0,t_1-t_0) is calculated by dividing the re-access request count (RC) by the total request count (TC) during [t_0,t_1].

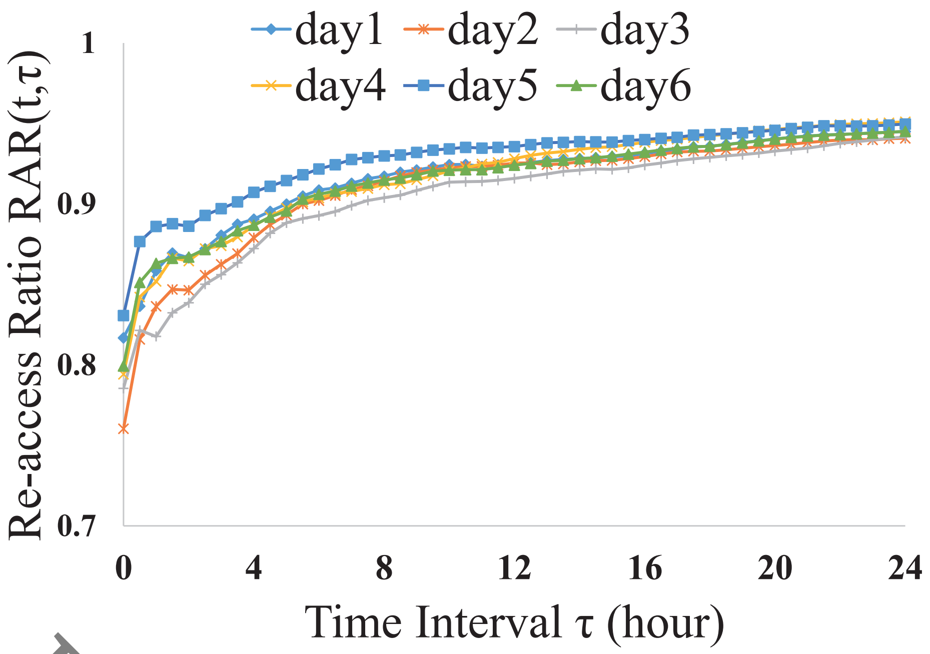
- To update RC and TC, we first lookup the block request in a hash map to determine whether it is a re-access request.



Re-access Ratio Curve

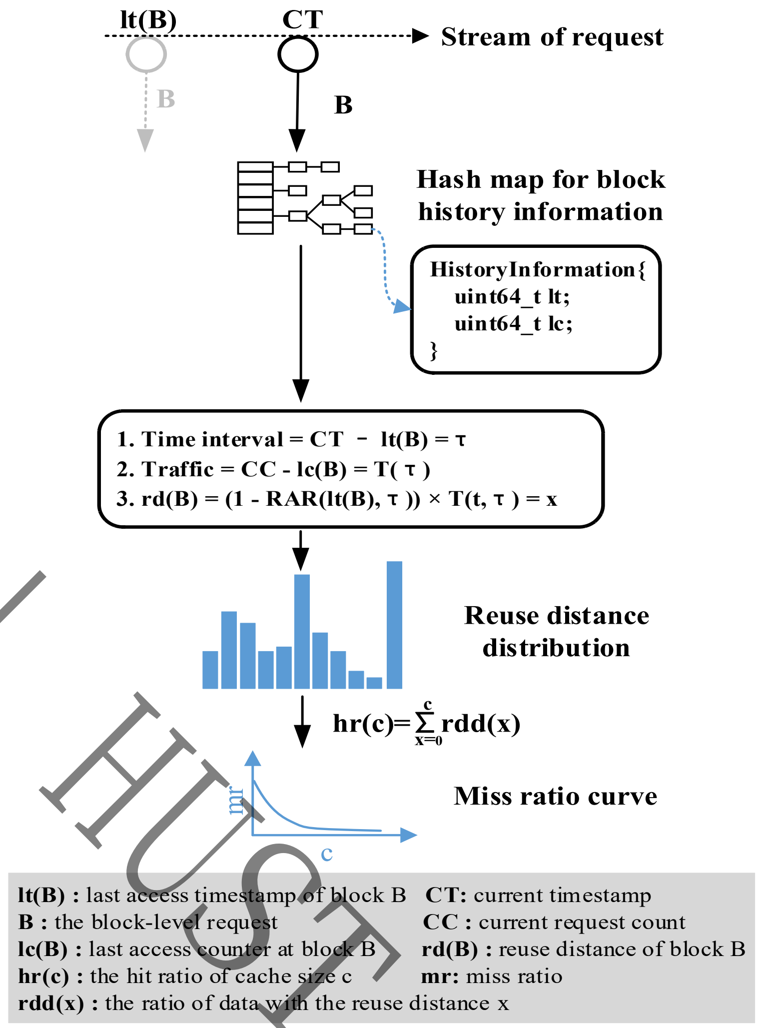
- The re-access ratio curve is a function relative to time interval τ and timestamp t, **denoted as RAR(t, τ).**

- Although cloud workloads are highly dynamic, we observe that the RAR curves are stable over a couple of days.



Construct MRC from RAR

- For a request to block B, we first check its history information in a hash map and obtain its last access timestamp (lt) and last access counter (lc, a 64-bit number denoting the block sequence number of the last reference to block B).
- We then use lt, lc and RAR curve to calculate the reuse distance of block B.
- Finally, the resultant reuse distance is used to calculate the miss ratio curve.



Define the Optimization Target

- Considering our case being cloud server-end caches, in this work we use the **overall hit traffic** among all nodes as our optimization target.
- The greater the value of E is, the less traffic is sent to the backend HDD storage.

Search for the Optimal Solution

Searching for Optimal Configuration

- **Based on the cache modeling and defined target mentioned above, our OSCA searches for the optimal configuration scheme.**

- Configuration searching process tries to find the optimal combination of cache sizes of each cache instance to get the highest overall hit traffic.

$$[CacheSize_0, CacheSize_1, \ldots\ldots, CacheSize_N]$$

Dynamical Programming

- The simplest method is the time-consuming exhaustive searching, which will calculate all possible cases.

- To speed up the search process, we use **dynamical programming** (DP).

System Evaluations

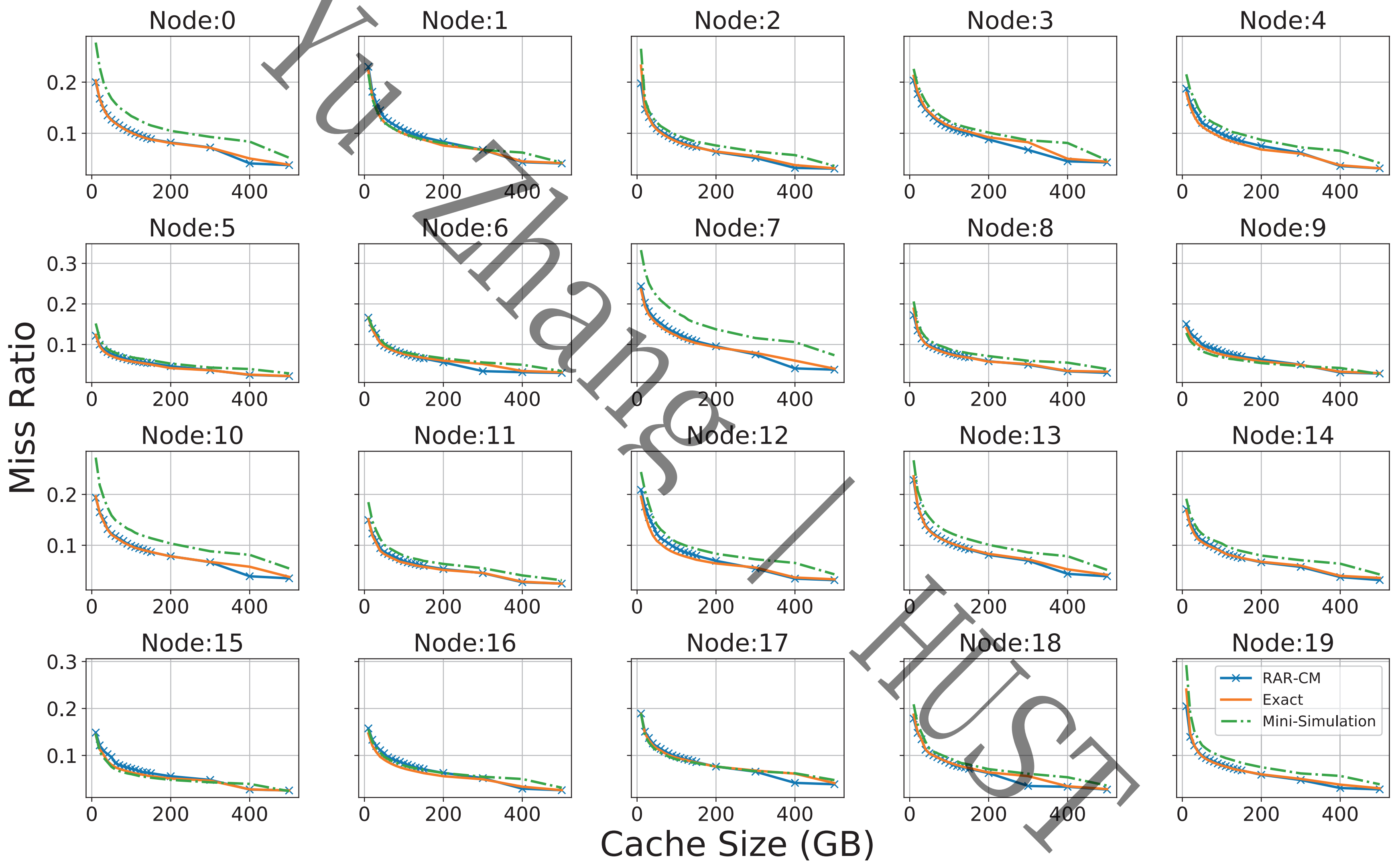
- **Trace Collection**
  - We have collected I/O traces from a production cloud block storage system. We are in the process of making it publicly available via the SNIA IOTTA repository.

- **Trace Storage**
  - The traces are stored in a storage server and each thread accesses the traces via the network file system (i.e., **Tencent CFS**).

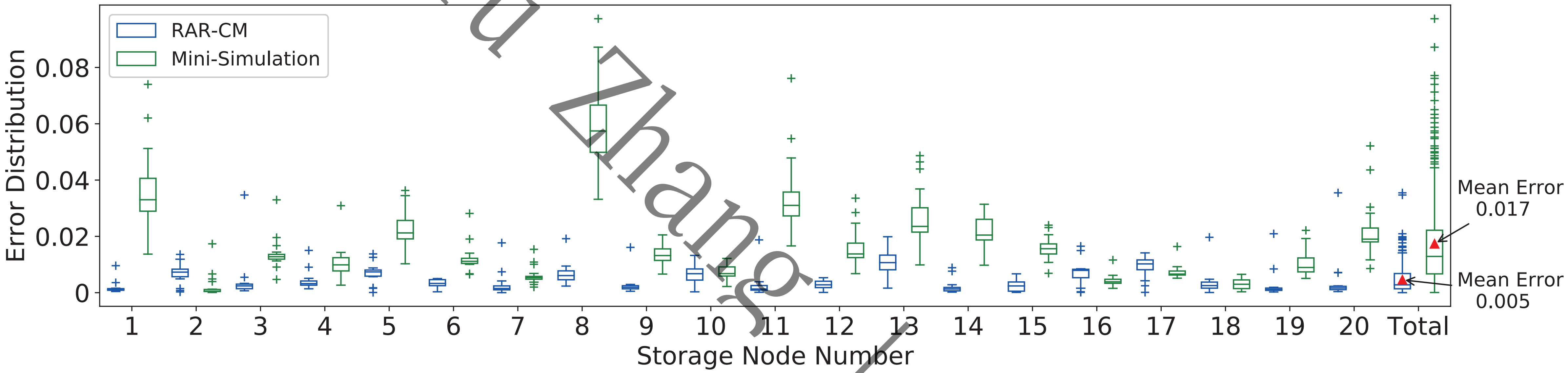
- **Simulation**
  - We have implemented a trace-driven simulator in C++ language for the rapid verification of the optimization strategy.

- **Counterpart**
  - Even-allocation Policy
  - Exact MRC Construction
  - Miniature-Simulation (FAST'15, USENIX'17)

Miss Ratio Curves

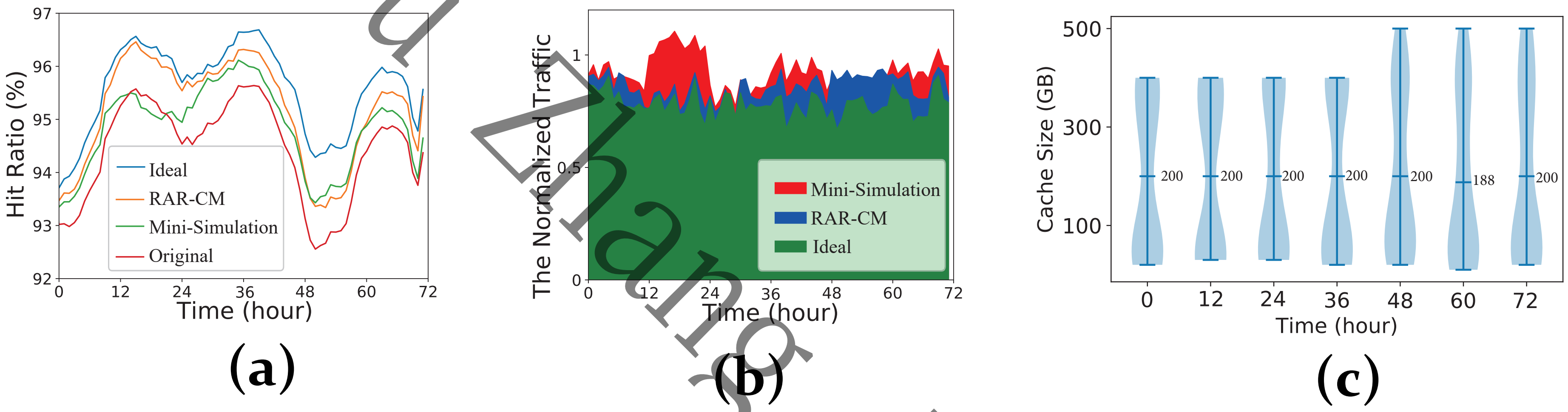


Mean Absolute Error (MAE)



- The MAE averaged across all 20 storage nodes (labeled "Total") for RAR-CM is smaller than for Mini-Simulation: 0.005 vs 0.017, in addition to being smaller for each of the 17 out of the 20 nodes.

Overall Efficacy



- We compare the efficacy of OSCA in terms of ***hit ratio*** and ***backend traffic***.
- The backend traffic is normalized to that of original method.
- On average, OSCA based on RAR-CM can reduce IO traffic to back-end storage server by 13.2%.
- OCSA adjusts the cache space for 20 storage nodes dynamically in response to their respective cache requirements decided by our cache modeling.

Conclusion

- Propose an online cache model-based cache allocation scheme for CBS systems
- Our approach complements the SHARDS method which adopts sampling but requires much less memory
- We have demonstrated its efficacy via perform simulating experiments with real-world CBS traces
- Publicize the traces to the storage research community

Q&A
Thanks！

Contact me :

Yu Zhang
Homepage: yuzhang.pro
E-mail: mail[at]yuzhang.pro