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

Presented BY *Xia, wenniu*

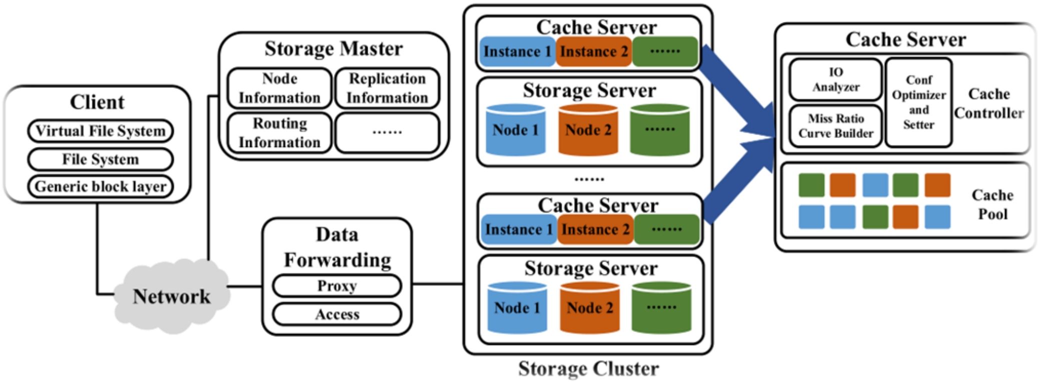
2020/10/16

Main contexts of presentations

- **Background**
- **Technique parts**
- **Experiment and results**
- **comments**

Background

- What is cloud block storage(CBS)?
- a CBS contains multiple components, the client, the storage master, the proxy and access server, and the storage server.
- The client : cloud disk virtualization and presents the view of cloud disks to tenants.
- The storage master (also called the metadata server) : node information, replication information, and data routing information.
- The proxy server : external and internal storage protocol conversion.
- The access server : I/O routing.
- The storage server consists of multiple failure domains to reduce the probability of correlated failures.



Background

- To ensure scalability, there are often multiple cache instances, each associated with one storage node, at the cache server.
- A cache instance is deployed to perform caching for each physical disk and our task is to partition the cache resource among all the cache instances.

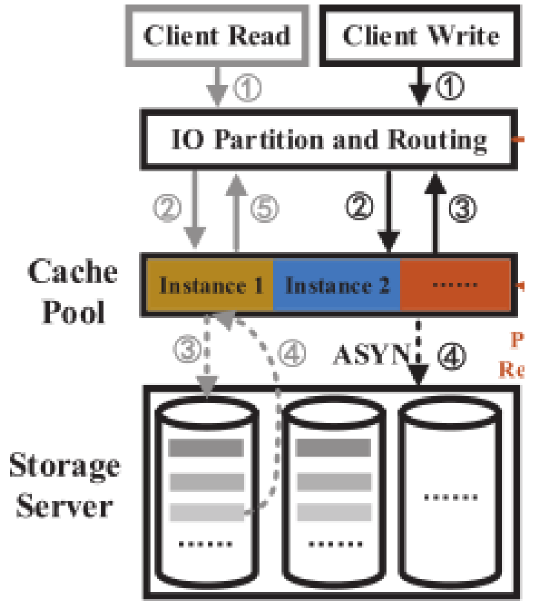
read process:

if find the data in the index map of the corresponding cache instance(i.e. in the cache):

return data to the client;

else:

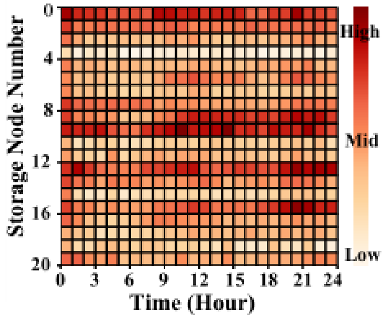
visit back-end HDD and return data;



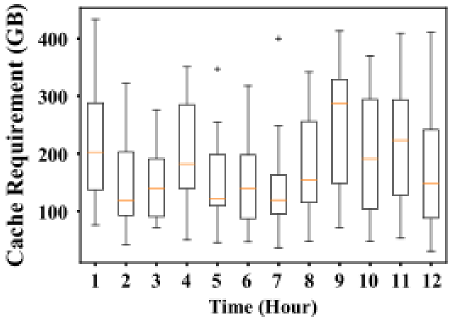
Write process:

first written to the cache, then flush back-end HDD storage asynchronously

Background



- Existing cache Allocation Scheme:
- Even-allocation policy (EAP): low hit-ratio and high traffic to back-end

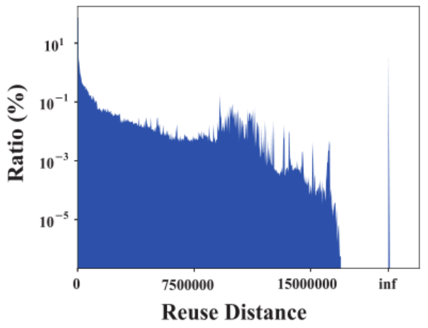


- So there have been proposed two broad categories of solutions:
  - intuition-based policies: TCM categorizes threads as either latency-sensitive or bandwidth-sensitive and correspondingly prioritizes the latency-sensitive threads over the bandwidth-sensitive threads, dependent on prior reliable experiences or workload regularities AND not guaranteed
  - model-based policies :quantitative methods enabled by cache models described by Miss Rate Curves (MRCs), which plot the ratio of cache misses to total references, as a function of cache size, offline analysis and cost-inefficient. (SHARDS/OSCA excluded)

Background

- Existing Cache Modeling Methods：
  - Locality quantization method(most commonly, reuse distance distribution(*RDD*)): like SHARDS, first selects a representative subset of the traces through hashing block addresses. It then inputs the selected traces to a conventional cache model to produce MRCs.
  - Simulation-based cache modeling: miniature simulation based on the idea of SHARDS need to concurrently run multiple simulation instances to determine the cache hit ratio in different cache sizes.

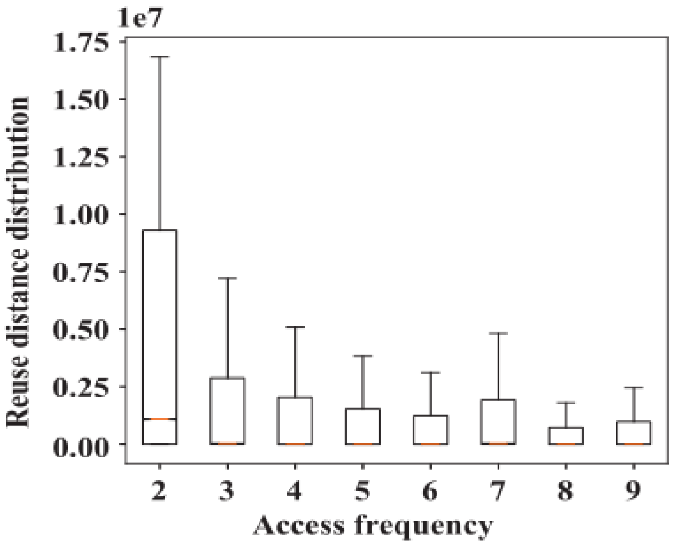
Technique parts



- Some knowledge prepared for OSCA's Technique:
  - 1. reuse distance: the amount of unique data blocks between two consecutive accesses to the same data block. For example, A-B-C-D-B-D-A, the reuse distance of data block A is 3.
  - 2. eviction distance: the amount of unique blocks accessed *from* the time it enters the cache *to* the time it is evicted from the cache, dependent on cache algorithm (LRU, ARC, LIRS……)
  - 3. if reuse distance < eviction distance, it means the data block hits the cache.
  - 4. we can plot reuse distance distribution from reuse distance of each block
  - 5. The LRU algorithm uses one list and always puts the most recently used data block at the head of the list and only evicts the least recently used block at the tail of the list. As a result, the eviction distance of the most recently used block is equal to the cache size.(SET cache size = eviction distance * block_size)

Technique parts

- Some knowledge prepared for OSCA's Technique:
  - 6. Eviction distance dependent on cache algorithm (LRU/ARC/LIRS), and this paper focus on LRU:
    - Reason1: LRU is simple and widely deployed in many real cloud caching systems.
    - Reason2: when the cache size becomes larger than a certain size, the advanced algorithms would degenerate to LRU.



tricky

Explanation: set cache size=229GB(i.e. $0.75*10^7$ blocks and block size is 32KB). So most LRU blocks with access frequency no smaller than 2 can be hit in cache and advanced cache algorithms have bigger access frequency so that they can hit cache more easily but less meaningfully.

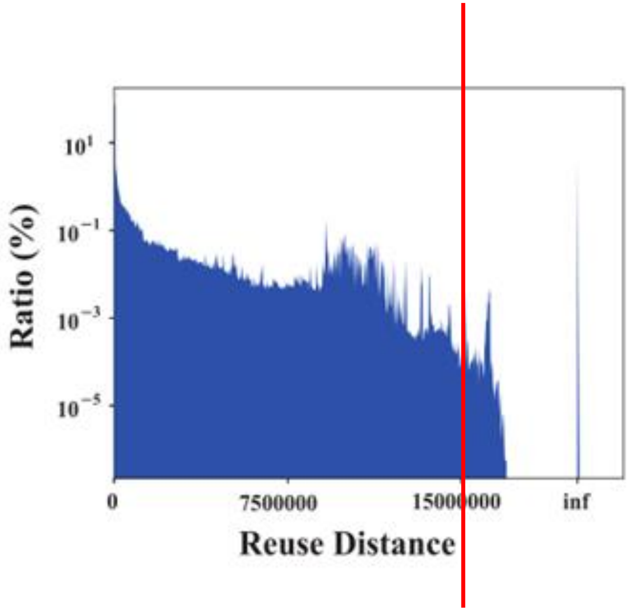
Technique parts

- Some knowledge prepared for OSCA's Technique:
  - 7. the hit ratio of the LRU algorithm as the discrete integral sum of the reuse distance distribution (from zero to the cache size)

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

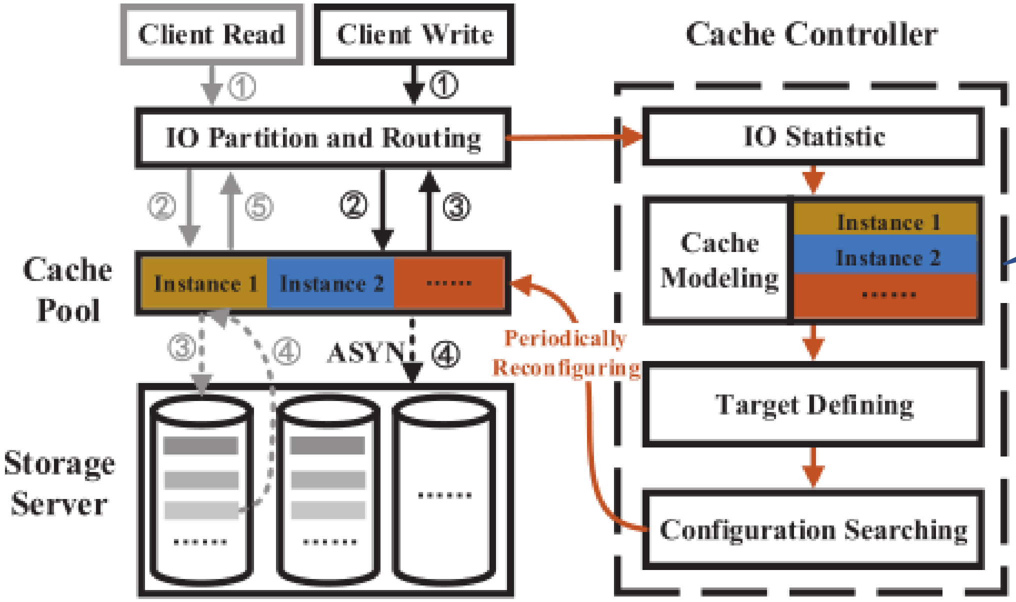
How to plot rdd?
OSCA mainly solve it?

hr(C) is the hit ratio at cache size C and rdd(x) denotes the distribution function of reuse distance.



Technique parts

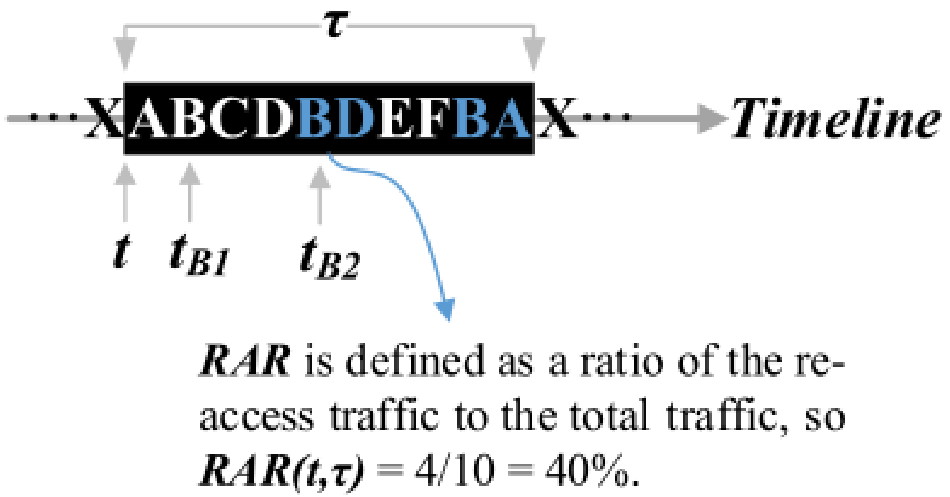
- Design overview:
  - online cache modeling, optimization target defining and the optimal configuration searching.



HOW to build miss ratio curve(MRC)

Technique parts

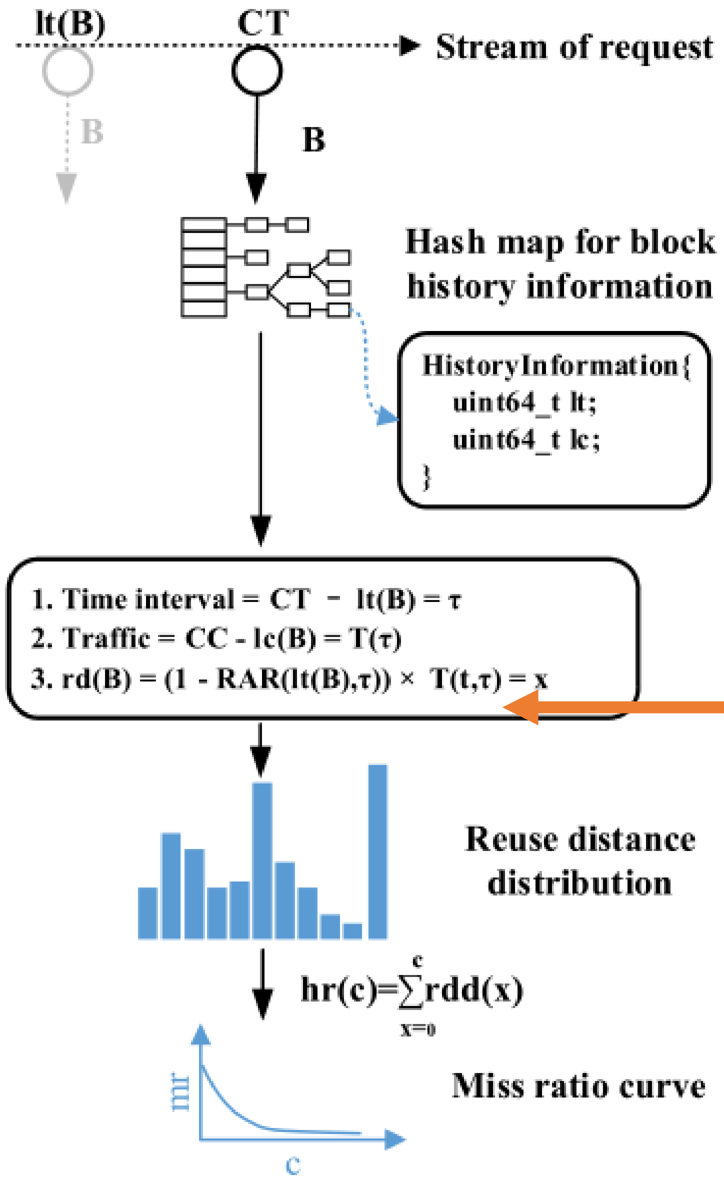
- Part one: online cache modeling
  - Re-access Ratio Based Cache Model (RAR)
    - RAR, which is defined as the ratio of the re-access traffic to the total traffic during a time interval τ after time t, is expressed as RAR(t,τ).



Technique parts

- part one: online cache modeling

Parameter definition



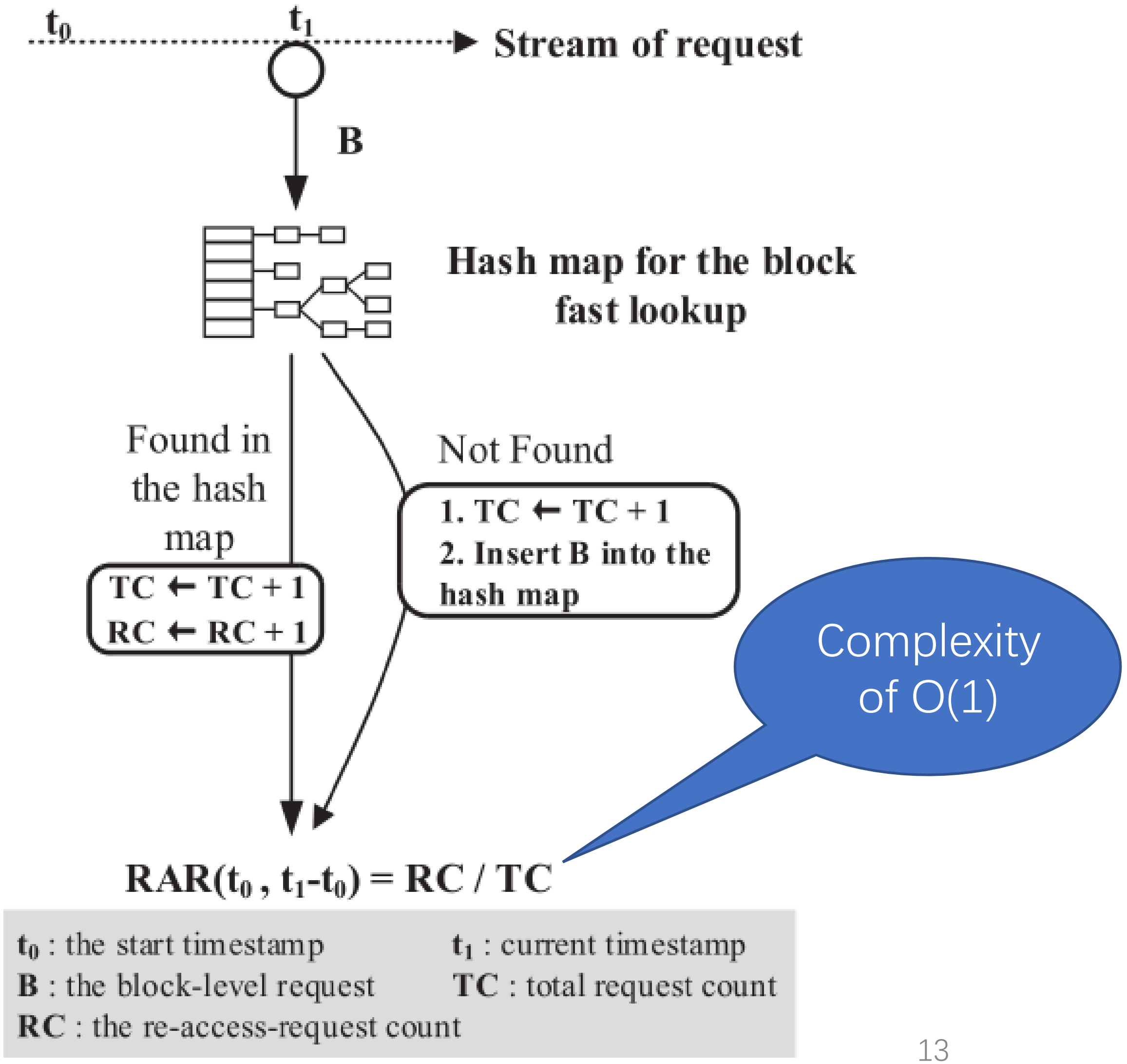
T(t,τ) means total block accesses between the two consecutive references to block B

lt(B) : last access timestamp of block B
B : the block-level request
lc(B) : last access counter at block B
hr(c) : the hit ratio of cache size c
rdd(x) : the ratio of data with the reuse distance x
CT: current timestamp
CC : current request count
rd(B) : reuse distance of block B
mr: miss ratio

Technique parts

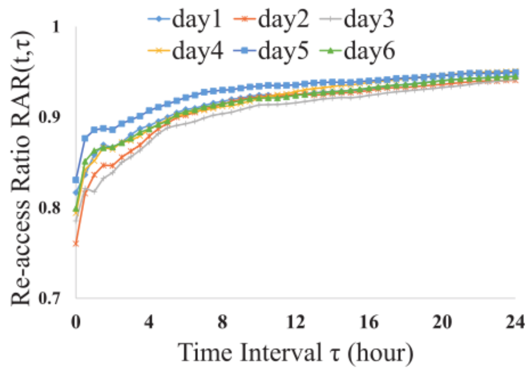
- Part one: online cache modeling
  - how to calculate RAR

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



Technique parts

- Part one: online cache modeling
  - favorable properties of RAR
    - 1. Complexity of O(1) (reason: calculated by dividing the re-access-request count (*RC*) by the total request count (*TC*))
    - 2. easily translated to the locality characteristics (reason: one equation translated to *RD*) $rd(B) = (1 - RAR(t, \tau)) \times T(t, \tau)$
    - 3. low overhead of memory footprint
      - i) can be approximated by logarithmic curves which have the form of RAR(τ) = a*log(τ) +b, where τ is the time variable, need to store two parameters
      - ii) changes of RAR curve are negligible over days



Technique parts

- Part two: Optimization Target

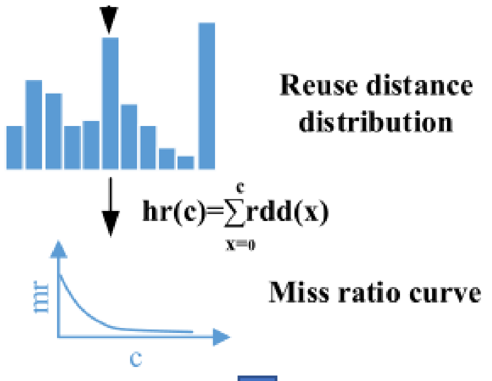
Vague and no show in detail

$$E = \sum_{node=1}^{N} HitRatio_{node} \times Traffic_{node}$$

  - In the equation, $\boldsymbol{HitRatio_{node}}$ represents the hit rate of the node and $\boldsymbol{Traffic_{node}}$ denotes the I/O traffic to this node.

Technique parts

- Part two: Optimization Target
  - Show in detail:
    - $\text{Max E} = \sum_{node=1}^{N} HitRatio_{node} * Traffic_{node}$
    - $Subject\ to \begin{cases} Cache_{node} = G(HitRatio_{node}) \\ total\ cache\ size\ (\text{fixed}) = \sum_{node=1}^{N} Cache_{node} \\ Cache_{node} >= 0\ (\text{node}=1,2,3\ \ldots\ldots) \\ Traffic_{node}\ \text{is statistic result} \end{cases}$



$HitRatio_{node} = hr(Cache_{node})$

Existing Inverse function G makes $Cache_{node} = G(HitRatio_{node})$ reasonable

Technique parts

- Part three: Searching for Optimal Configuration
  - dynamic programming (DP) is used

3.4 Searching for Optimal Configuration

Based on the cache modeling and defined target mentioned above, our *OSCA* searches for the optimal configuration scheme. More specifically, the configuration searching process tries to find the optimal combination of cache sizes of each cache instance to get the highest efficiency E.

To speed up the search process, we use dynamic programming (DP), since a large part of calculations are repetitive. A DP method can avoid repeated calculations using a table to store intermediate results and thus reduce the exponential computational complexity to a linear level.

Technique parts

- algorithm details

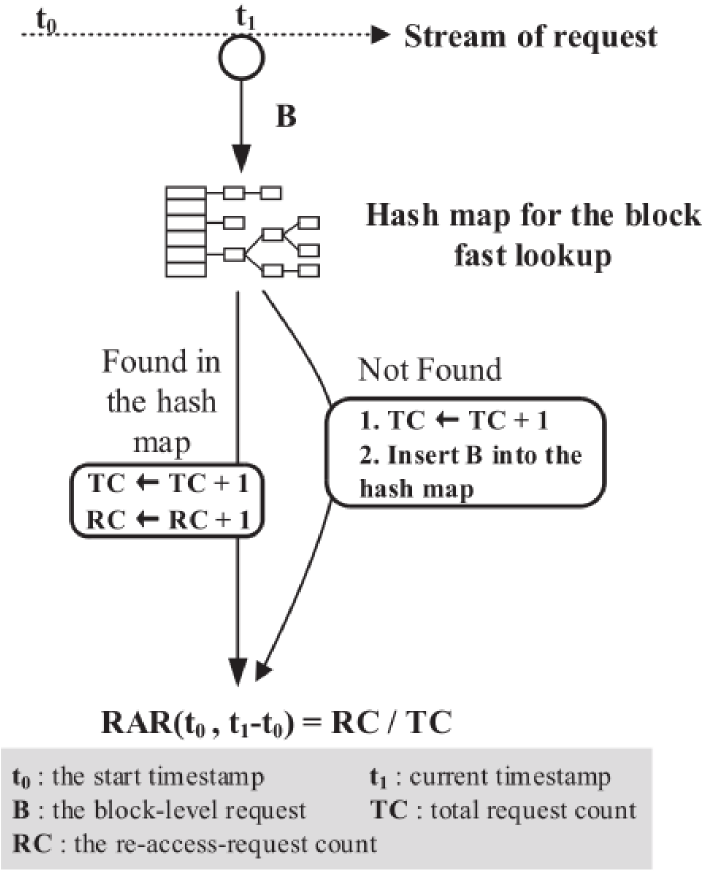
Algorithm 1: The pseudocode of the *RAR-CM* process

Data: Initialize the global variable: hash map for block history information H, current timestamp CT, current block sequence number CC, and the re-reference count RC. The re-access ratio curve RAR. The reuse distance distribution RD

Input: a sequence of block accesses

Output: output the miss ratio curve

while *has unprocessed block access* **do**
    $B \leftarrow$ *next block*
    $CC \leftarrow CC+1$
    $CT \leftarrow$ *current timestamp*
    if B *in* H **then**
        $RC \leftarrow RC+1$
        $RAR(H(B).lt, CT-H(B).lt) = RC/CC$
        $H(B).lc \leftarrow CC$
        $H(B).lt \leftarrow CT$
    end
    else
        Initialize H(B)
        $H(B).lc \leftarrow CC$
        $H(B).lt \leftarrow CT$
        Insert $H(B)$ into H
    end
    update_reuse_distance(B)
end
return *get_miss_ratio_curve*(*RD*)



Algorithm 2: Subroutine *update_reuse_distance*

Input: currently accessed block B

if B *in* H **then**
    $time\_interval = CT - H(B).lt$
    $traffic = CC - H(B).lc$
    $rd(B) = (1 - RAR(H(B).lt, time\_interval)) * traffic$
    $RD(rd(B)) \leftarrow RD(rd(B)) + 1$
end

Algorithm 3: Subroutine *get_miss_ratio_curve*

Input: the reuse distance distribution *RD*

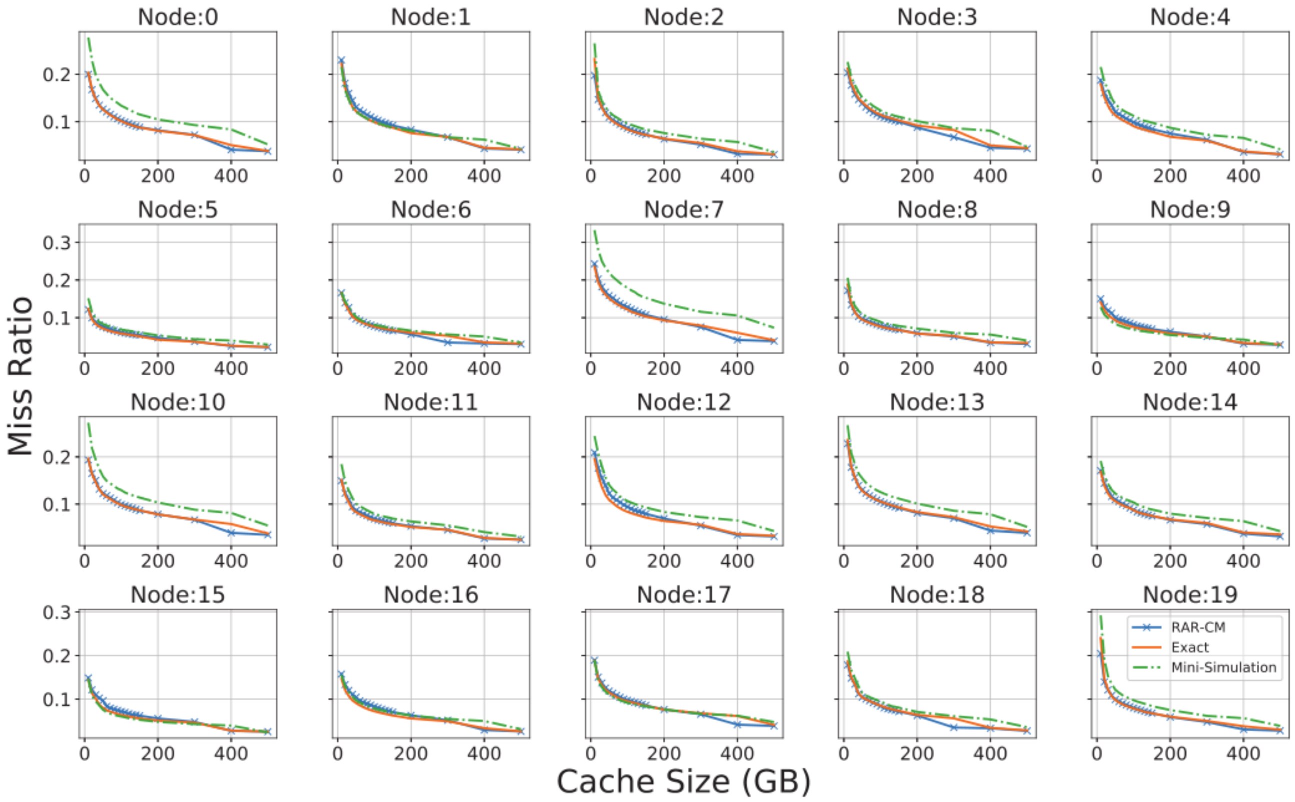
$total = sum(RD)$
$tmp = 0$
for *element in RD* **do**
    $tmp \leftarrow tmp + element$
    $MRC.append(1 - tmp/total)$
end
return *MRC*

Experiment and results

- collect 6 day long I/O trace from 20 nodes
- Language: C++, CPU: 12-core (Intel Xeon CPU E5-2670v3)
- Methods of Comparison:
  - This paper’s model(RAR-CM)
  - even-allocation method (Original)
  - miniature simulation with the sampling idea from SHARDS (Mini Simulation)
  - ideal case (Ideal) where exact miss ratio curves are used in placement of constructed cache models.

Experiment and results

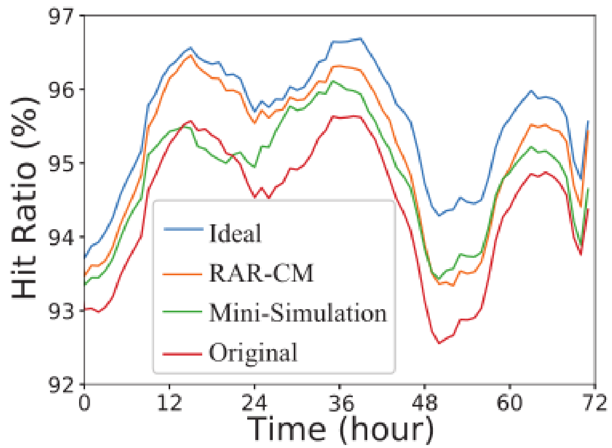
- Miss Ratio Curves(*RAR-CM*: blue, *Mini-Simulation*: green, *Ideal*: orange)

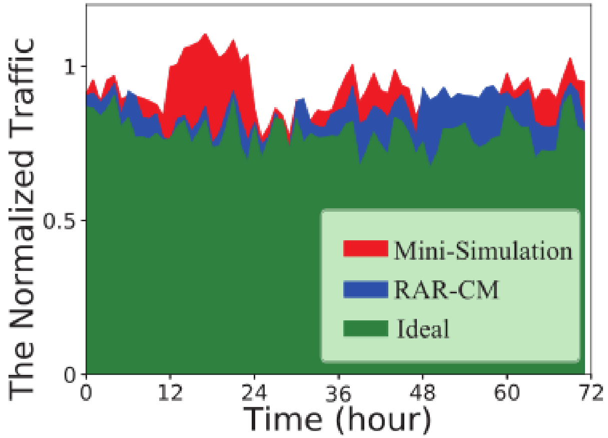


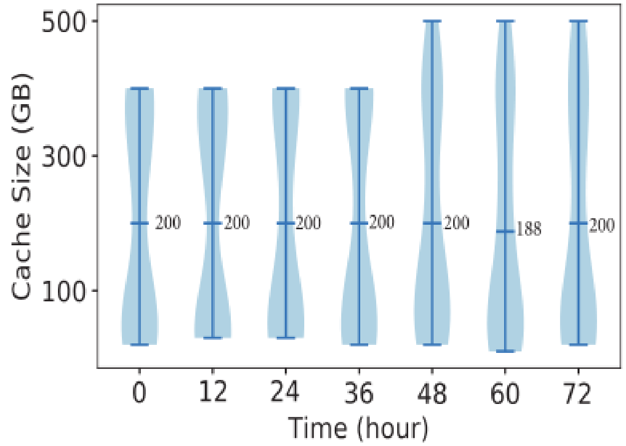
Result: the curves of RAR-CM are closer to the curves of the exact simulation than that of Mini-Simulation in most cases.

Experiment and results

- Overall Efficacy of OSCA(last three days)







1.result: RAR-CM has a higher hit ratio

2..result: RAR-CM has a lower traffic to backend

3.result: RAR-CM can adjust dynamically in response to cache requirements.

comments

- Advantages:
  - 1. higher cache hit ratio and lower traffic to back-end
  - 2. lower computation complexity and memory footprint
  - 3. online
  - 4. new idea
- Disadvantages:
  - 1. tricky in some details
  - 2. not show in detail at some key places
  - 3. parts of optimization target and configuration searching is vague and redundant