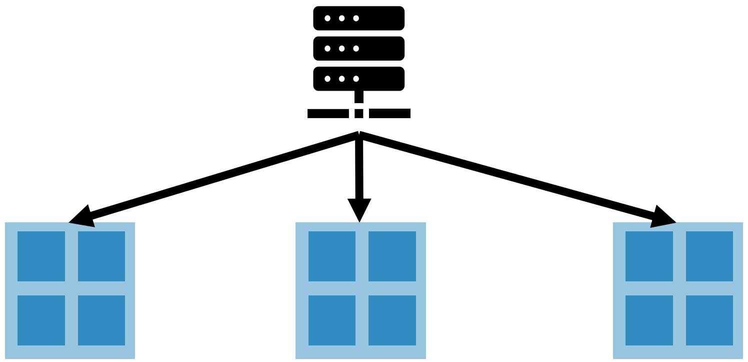
DistCache: Provable Load Balancing for Large-Scale Storage Systems with Distributed Caching

Zaoxing Liu and Zhihao Bai, Johns Hopkins University; Zhenming Liu, College of William and Mary; Xiaozhou Li, Celer Network; Changhoon Kim, Barefoot Networks; Vladimir Braverman and Xin Jin, Johns Hopkins University; Ion Stoica, UC Berkeley

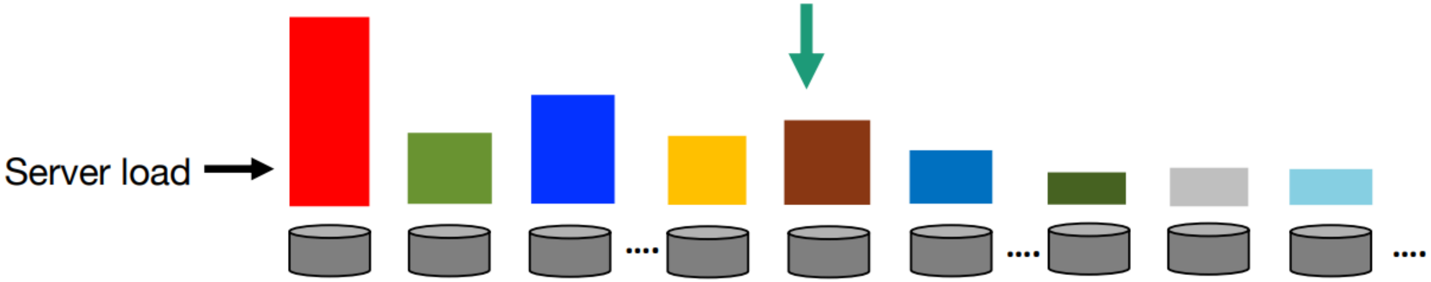
FAST 2019

Background

- Large-scale storage system consists of storage clusters, cluster consists of storage server/node

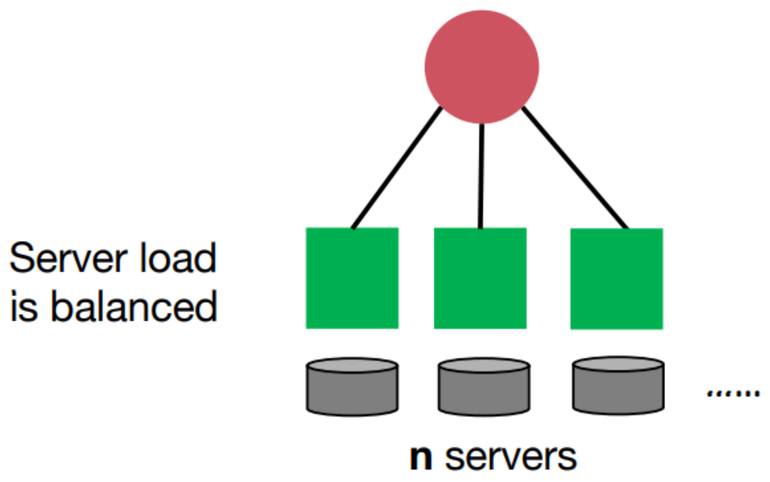


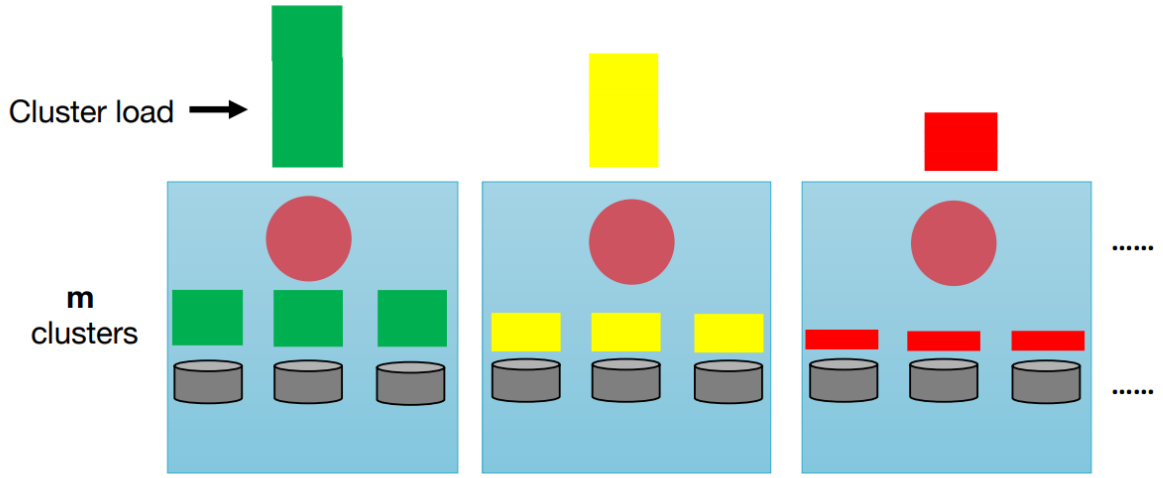
- Storage node have load imbalance issue Due to uneven distribution of visits



Background

- Prior work has proved that set up a cache node in cluster to cache O(nlogn) hot objects inside cluster can solve intra-cluster imbalance

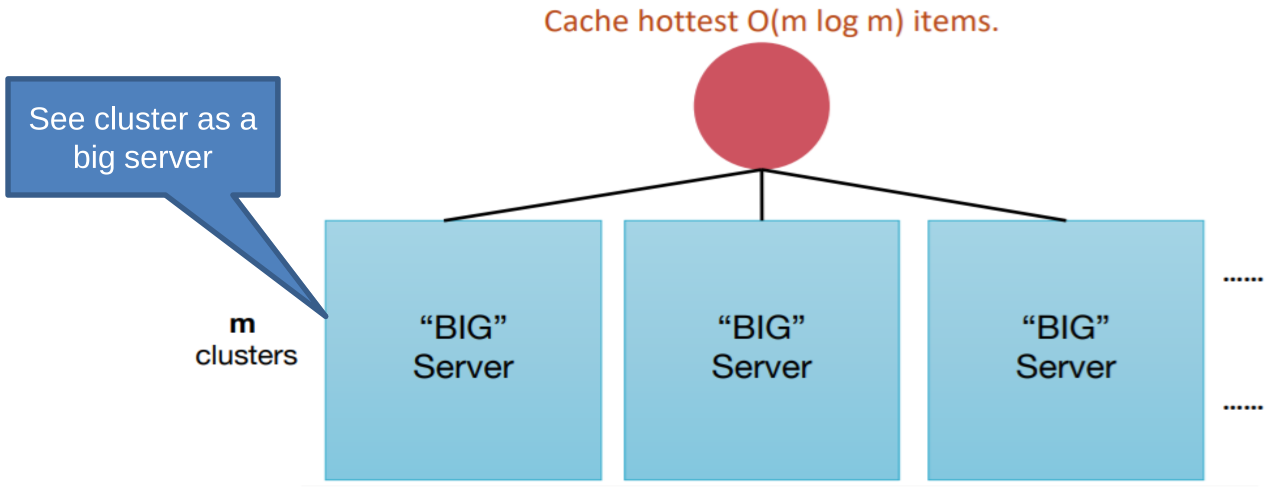




- **The load is still unbalanced between the clusters**

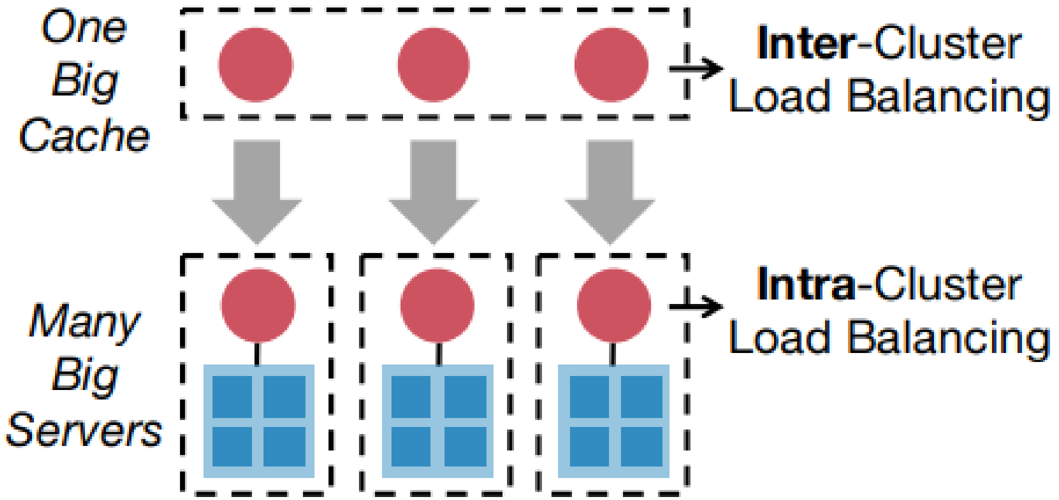
Background

- How to solve load imbalance between clusters?
  - **Set up a cache node to balance load between clusters**



- However, big node should be avoid using(High load)
  - Split a large node into multiple small nodes.

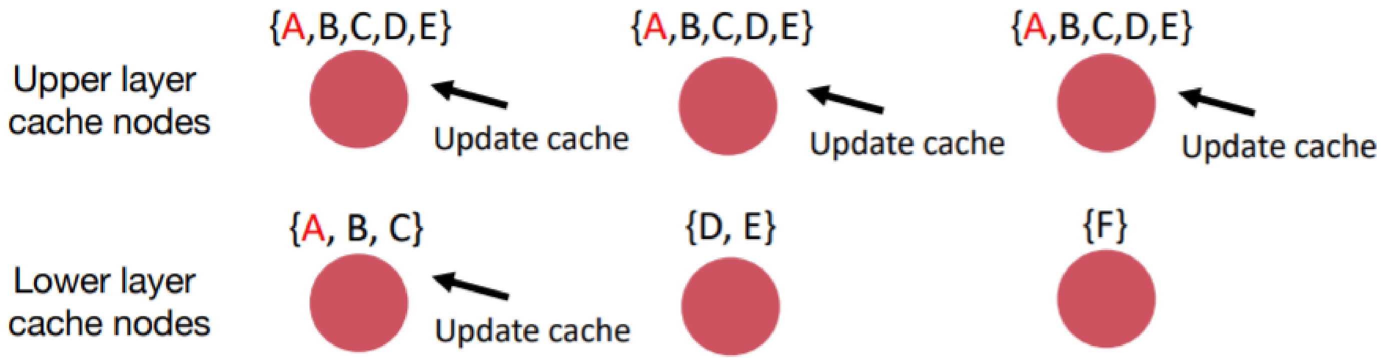
Goal

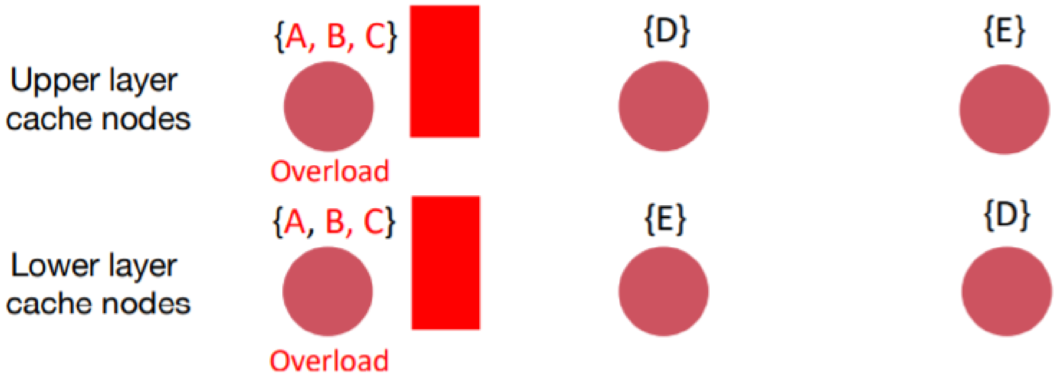


- **The up cache layer should achieve the same effect as a big cache node**
  - Support ANY query workload to hottest O(m log m) items
  - Each cache node is NOT overloaded
  - Keep cache coherence with MINIMAL cost

Challenge

- How to allocate cache for up layer?
  - ***Replication***
    - *Copy objects from lower layer to upper layer*
    - *Cache throughput can grow linearly with number of cache nodes, but high coherence overhead*
  - ***Partition***
    - *Each up node corresponds to the lower node*
    - *Can't solve the load imbalance*
- How to update object items?
- How to query objects?



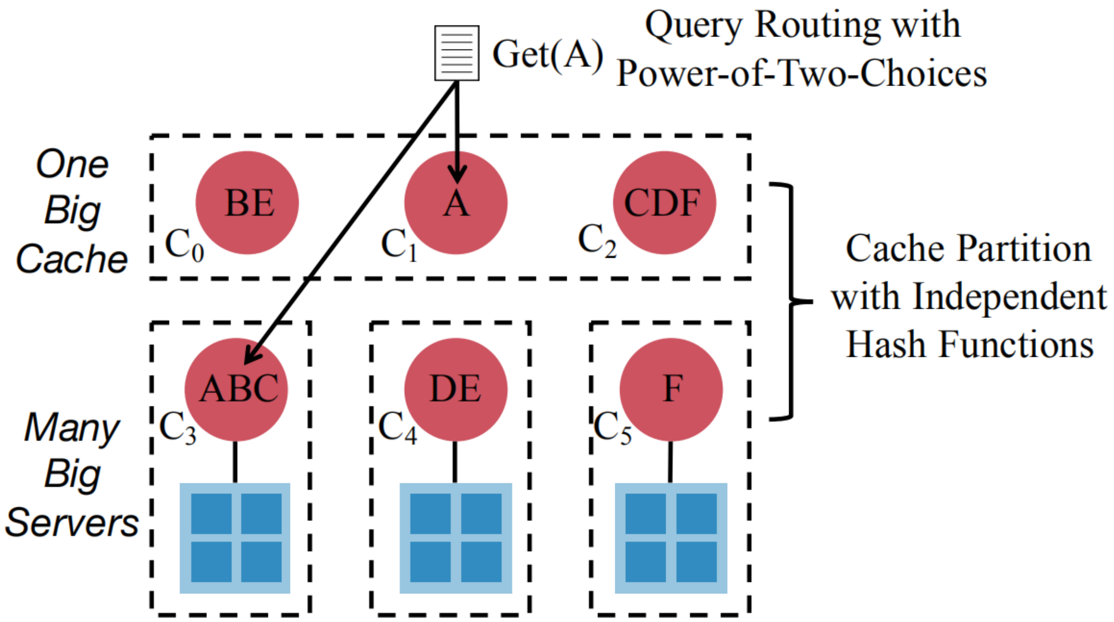


Idea

- Using independent hash functions for cache allocation
- Query routing with the power-of-two-choices
  - Use the same hash functions as cache allocation to find cache node
  - Chose the lower load of two cache node
- Update using off-the-shelf technology.(two phases update)
  - First, mark server and cache nodes which has date to be update invalid and update primary cache
  - Update all cache.

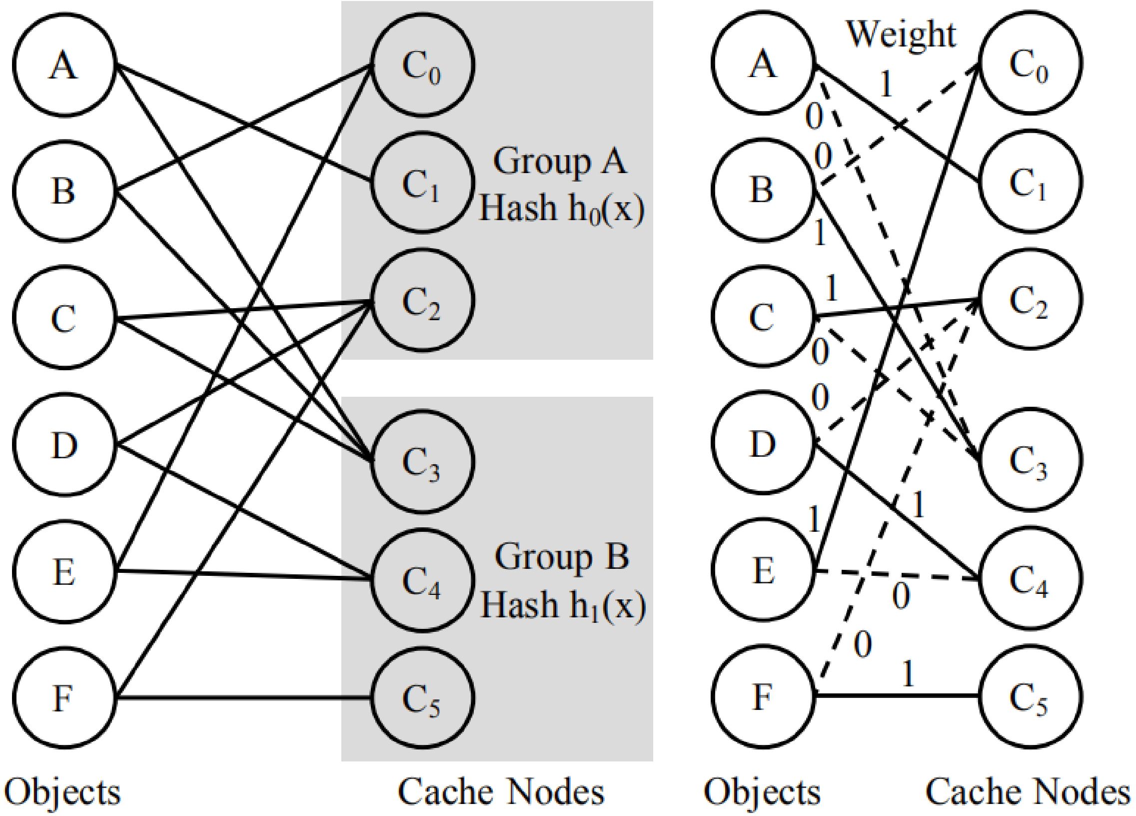
Note:

- number of two layers' cache node can be different.



Rationality

- Goal: cache nodes can absorb all queries to hottest O(mlogm) objects, despite query distribution



(a) Bipartite graph.

(b) Perfect matching.

- all objects have a query rate of 1, and all cache nodes have a throughput of 1
- Each group represents a layer
- If we construct bipartite graph as left, it can be proved that there always exists perfect match. That's to say, goal can be satisfied

Architecture

- ***DistCache***, a switch-based caching system

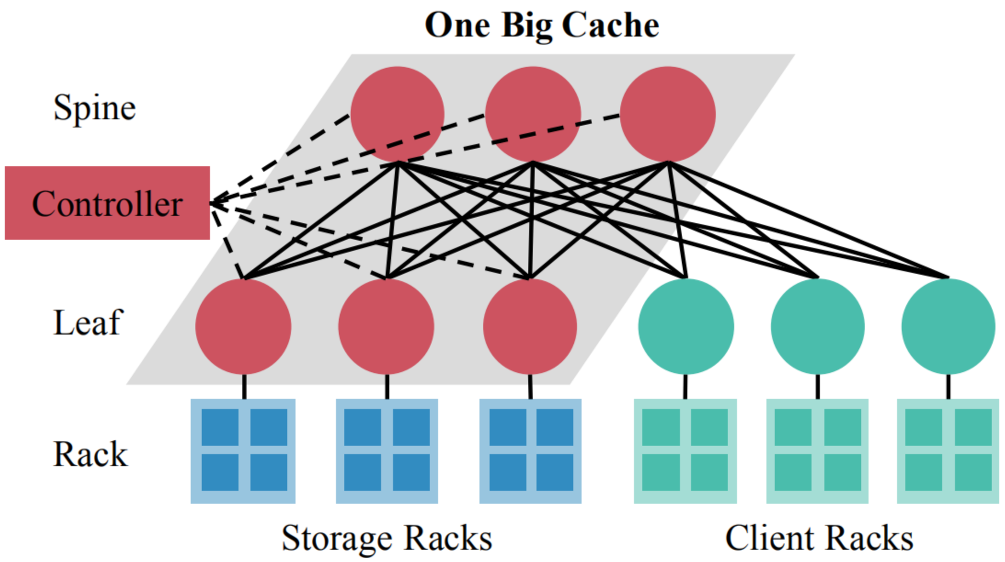
- Controller:
  - computes cache partitions, and notifies cache switches
  - updates cache allocation under system reconfigurations

- Cache switches:
  - caching hot key-value objects
  - distributing switch load information for query routing

- ToR(top of rack) switches at client racks:
  - use power-of-two-choices to decide which cache switch to send query to

- Storage servers

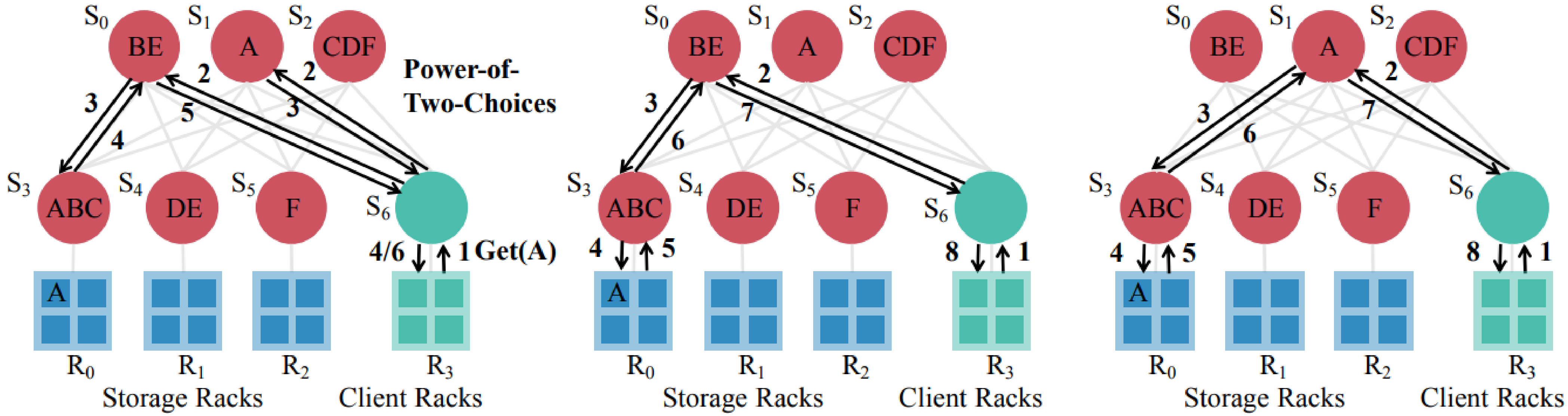
- Client Racks:
  - provides client library for applications to access key-value store



System design

➢ Query Handling

- Client ToR switches chose lower load of two server to access
  - ToR switch stores cache nodes’ load information
- Routing mechanism choose the least loaded path to the cache switch
- In-network telemetry for cache load distribution
- If cache miss generates, the query is sent to storage server



(a) Cache hit. (b) Cache miss (leaf cache S_3). (c) Cache miss (spine cache S_1).

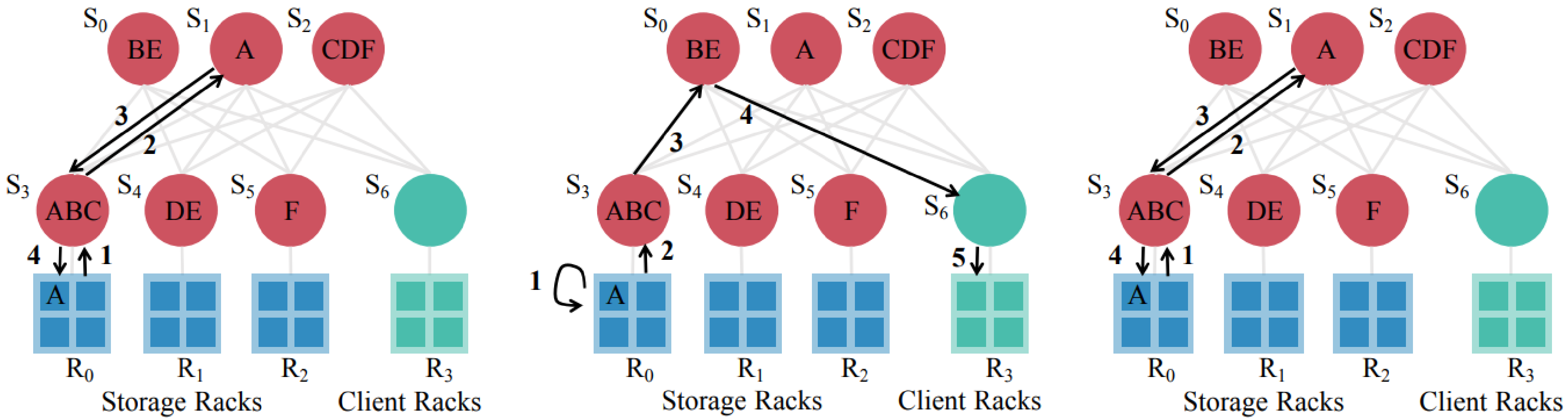
System design

Cache Coherence

- Storage server notifies all cache nodes that has data to be update are invalid.
- After the first phase, update its primary copy, and send an acknowledgment to the client
- Update all cache nodes

Cache Update

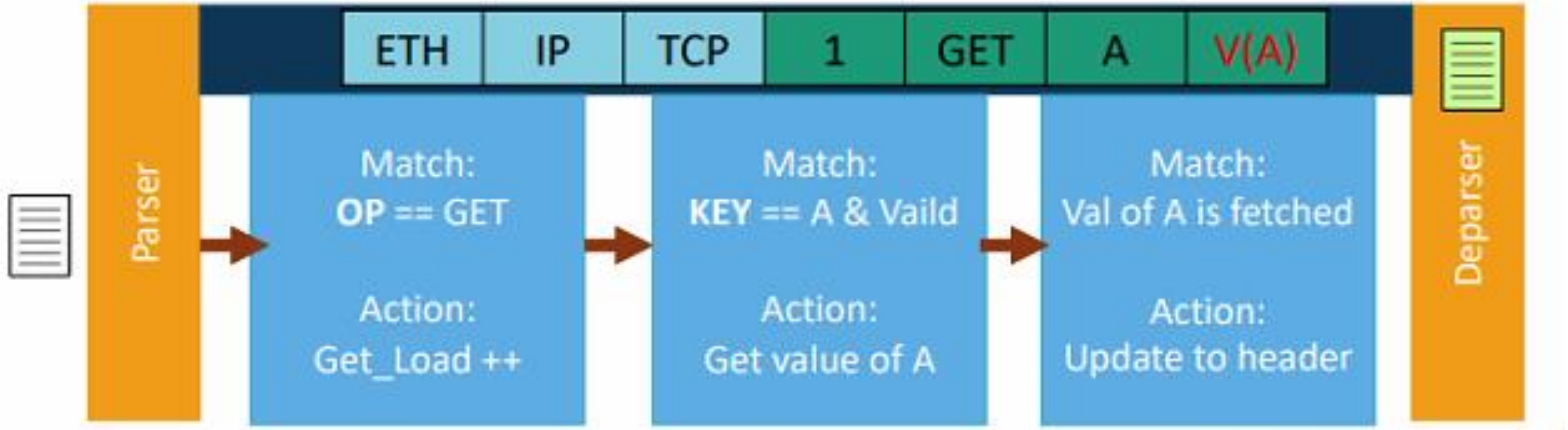
- Switch detect hot objects in its own partition, and decides cache
- Cache eviction is finished by switch directly.
- Cache insertion need contact with storage server. (Distributed coordination technology)



(a) Phase 1: invalidate all caches. (b) Update primary copy, and acknowledge client. (c) Phase 2: update all caches.

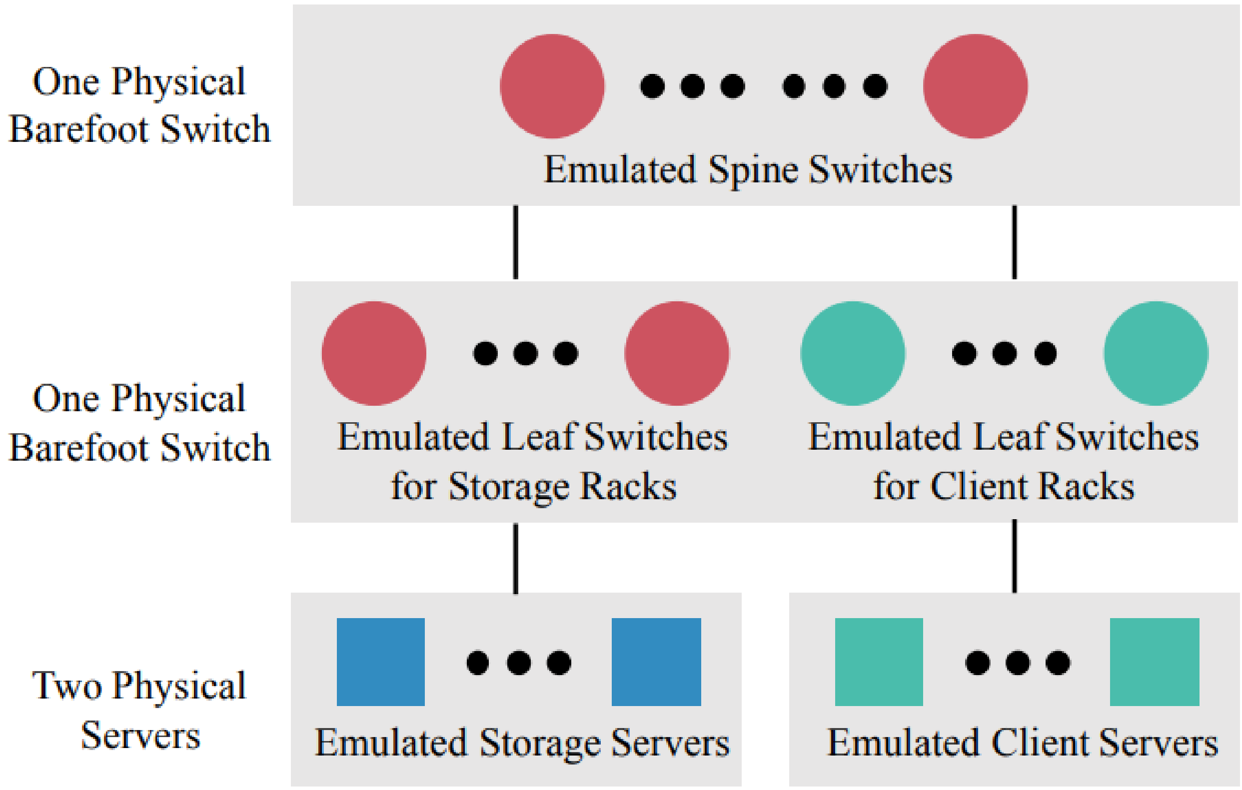
Implementation

- Client and storage: Python
- Switch: P4 for define packet format



Evaluation setup

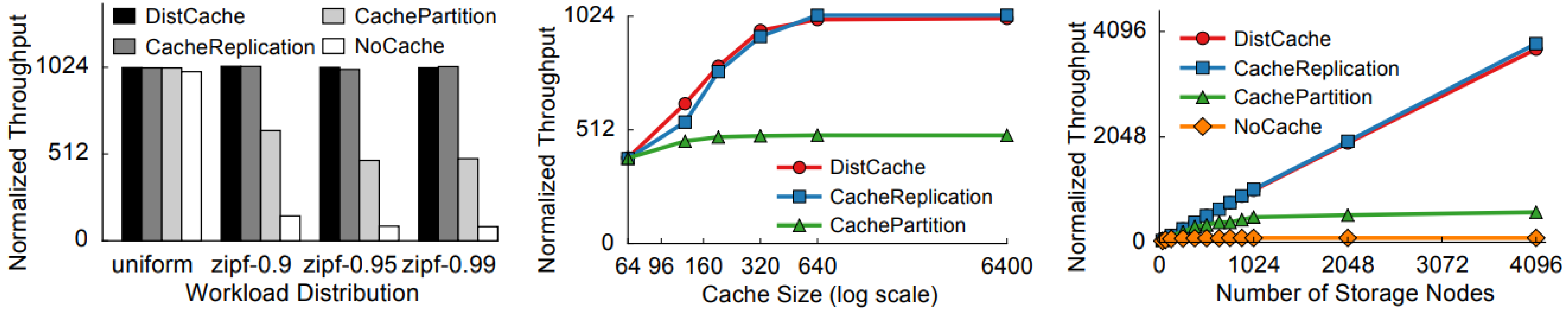
- **Methodology**
  - two 6.5Tbps Barefoot Tofino switches and two server machines
  - server machine is equipped with a 16 core-CPU (Intel Xeon E5-2630), 128 GB total memory (four Samsung 32GB DDR4-2133 memory), and an Intel XL710 40G NIC
- **Workloads**
  - use both uniform and skewed workloads
  - Size: 100 million objects
  - uniform workload generates queries to each object with the same probability
  - skewed workload follows Zipf distribution with skewness parameter (e.g., 0.9, 0.95, 0.99)
    - **P(r) = C / r^α**
- **Comparison**
  - DistCache
  - CacheReplication, CachePartition, and NoCache.(only on up layer)



- Up layer cache node size: O(mlogm)
- Bottom layer cache node size: O(nlogn)

Evaluation

- Performance for Read-Only Workloads
  - **Impact of workload skew**
  - **Impact of cache size**
  - **Scalability**

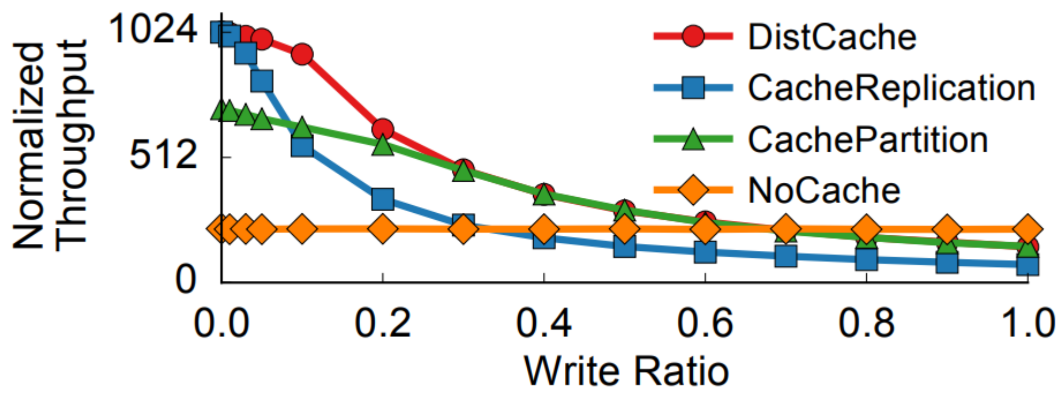


(a) Throughput vs. skewness. (b) Impact of cache size. (c) Scalability.

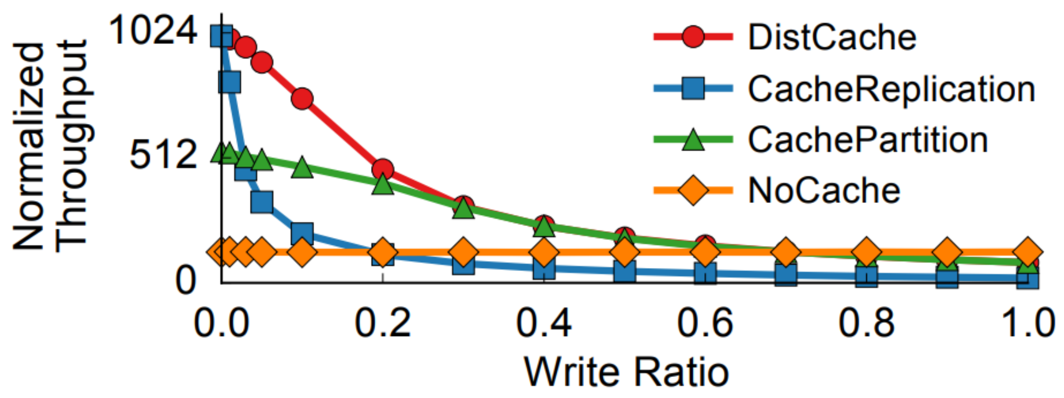
- **Regardless of skewness, cache size, and scalability, A provides approximately the same throughput as CacheReplication.**
- **CachePartition's performance is not as good as DistCache due to the hot spots between its own node Cache points**

Evaluation

- **Cache Coherence**



(a) Throughput vs. write ratio under Zipf-0.9 and cache size 640.



(b) Throughput vs. write ratio under Zipf-0.99 and cache size 6400.

- **DistCache can ensure high throughput when the write account is relatively low**
- **The throughput of CacheReplication drops quickly because of the consistency of the guarantee**
- **CachePartition's performance is not as good as DistCache due to the hot spots between its own node Cache points**

Evaluation

➢ Failure Handling

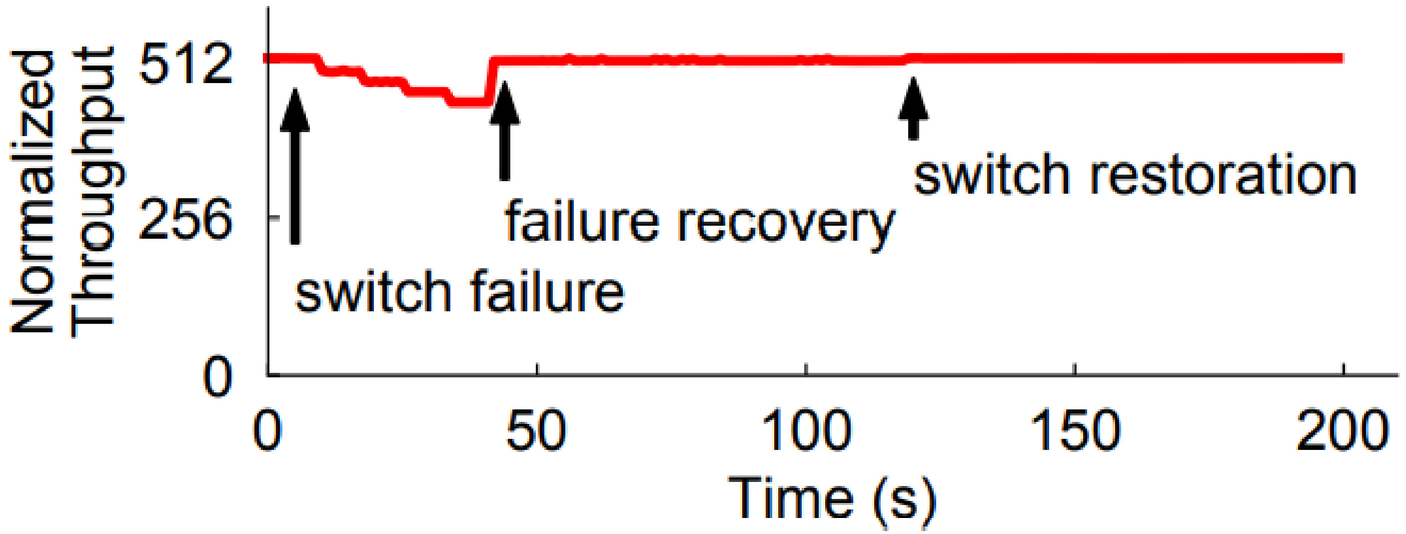


Figure 11: Time series for failure handling.

- ➢ **Even if an error occurs, the performance degradation of the system will not be particularly noticeable**

Conclusion

- Experiments proved that power of two choice s makes a “life-or-death” improvement in this problem, instead of a “shaving off a log n” improvement.

- **Disadvantage:**
  - Mathematical proof about bipartite graph is obscure
  - Lack of support from actual data in experiments and problem