CableCache: In-Network Request Deduplication for Key-Value Stores

Jiawei Huang[1], Junru Li[1], Qing Wang[1], Lijie Wen[1], Youyou Lu[1], Erci Xu[2]

[1]Tsinghua University; [2]SJTU

清華大学
Tsinghua University

Contents

The load imbalance problem in key-value stores

- Key-value stores are widely deployed in modern data centers.
- Key-value stores face a key challenge posed by skewed dynamic workloads, which can lead to load imbalances.
    - Facebook: 10% of objects account for 60%~90% of requests.
    - Alibaba: The Zipfian parameter reaches 0.9~0.99 in daily scenarios and 1~1.22 in extreme cases.

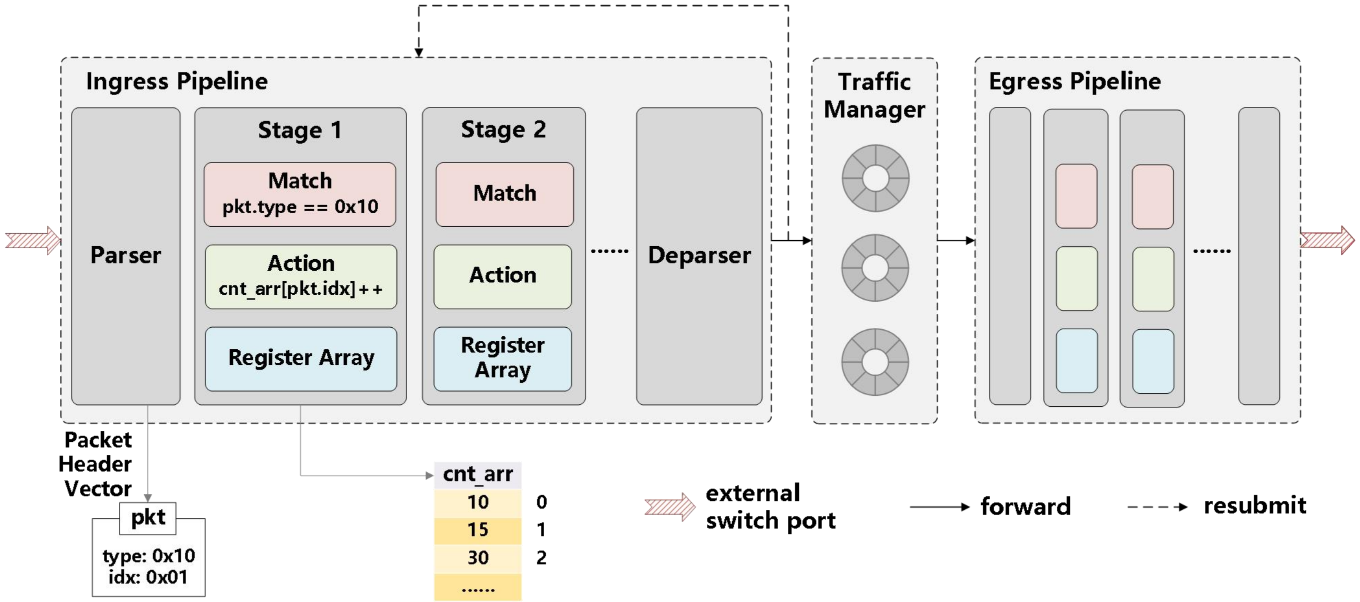
RocksDB

redis

MEMCACHED

A novel network hardware —— programmable switch

- Programmable switches utilize programmable acceleration chips to support user-defined network protocols and packet forwarding logic.
- Current programmable switches contain multiple ingress and egress pipelines.
- A single pipeline consists of multiple stages.



Two advantages of the programmable switch

Position:

processes network packets in the transfer path between clients and storage servers.

Performance:

enables line-rate packet forwarding with throughput on the order of Tbps.

Existing methods' limitations

NetCache / **FarReach**: caches hot objects directly in programmable switches

- The size of object values **cannot exceed 128 B**.
- NetCache does not support write caching.
- FarReach supports write caching but needs complex measures to handle switch failure.

Pegasus: ransfers the directory of selective replication to programmable switches

- The size of object values **cannnot exceed MTU**.
- Directing all requests to storage servers results in longer access paths.
- A complex chain replication protocol is required to prevent data inconsistency caused by switch failure.

Real workloads analysis

- **Twitter**: the top 1% hot objects in 54 workloads of different clusters
  - 35 workloads contain hot objects >128B.
  - 18 workloads contain hot objects >1500 B (Ethernet MTU).
  - 10 workloads contain hot objects >9000 B (jumbo frame max).
  - 25 workloads show median hot object sizes >128 B.
  - 16 workloads include read-modify-write requests (not supported by previous work).
- **Facebook**: the top 1% hot objects in October 2022 sampled workloads
  - 19.35% of top 1% hot objects exceed 128B, yet they generate 86.99% of network traffic.

Request collision analysis

- **processing latency**: refers to the time interval between a switch receiving a request and its corresponding reply.
- **request collision**: occurs when the processing latency periods of requests for the same object overlap.

α	L (μs)				
	10	25	50	75	100
0.99	41.67%	49.56%	55.56%	59.09%	61.59%
1.11	61.66%	67.86%	72.53%	75.12%	76.90%
1.22	72.41%	79.41%	82.81%	84.61%	85.81%

The weighted probability of request collisions for the top 65536 objects.

keyspace: 250 M, **system-wide throughput**: 100 MRPS

α: Zipfian parameter, ***L***: processing latency

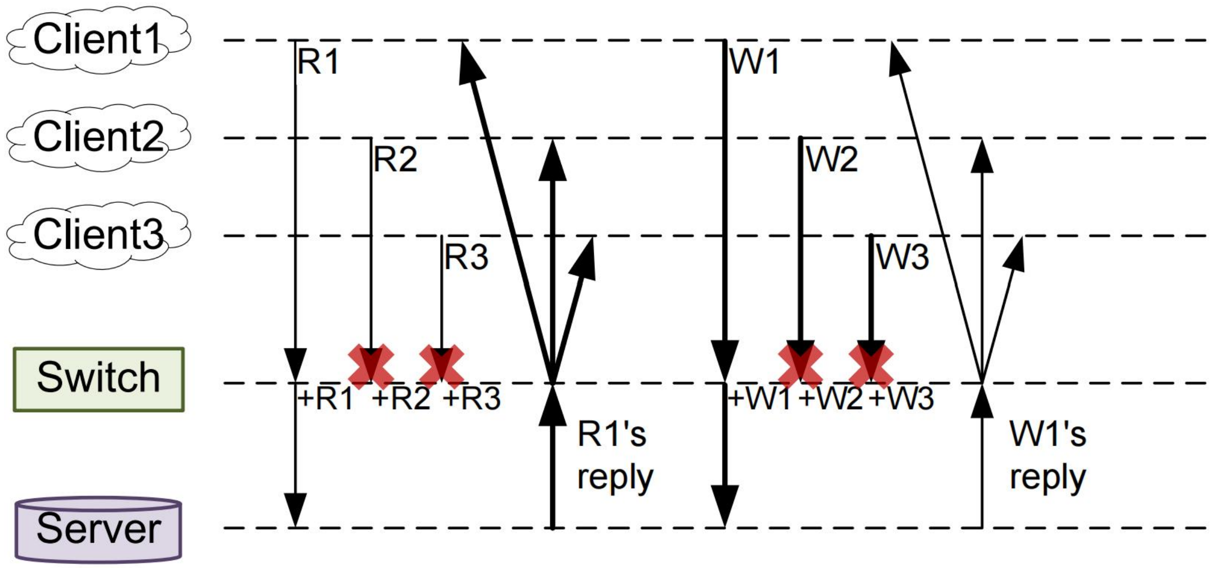
Comparison of Different Methods

Name	Operation	Object Size	Access Path	Switch Failure Handling
NetCache	read	$\leq$ 128 B	short	simple
FarReach	read & write	$\leq$ 128 B	short	complex
Pegasus	read & write	$\leq$ 1 MTU	long	complex
CableCache	read & write & read-modify-write	$\leq$ 32 MTU	medium	simple

CableCache's core idea: Maintaining an object request information directory in the programmable switch to avoid repeated requests for hot objects.

Packet flow

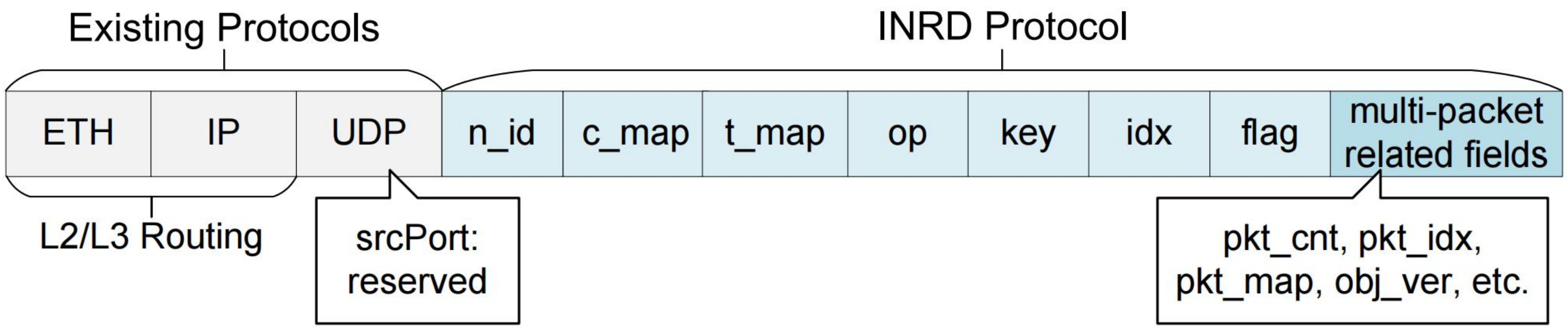
- CableCache separates the deduplication processing of read and write requests.
- The first read request **R1** will be recorded in the switch and sent to the storage server.
- The read request **R2** and **R3** will be recorded and then dropped.
- The write packet flow is similar.



Tsinghua University

Packet header format

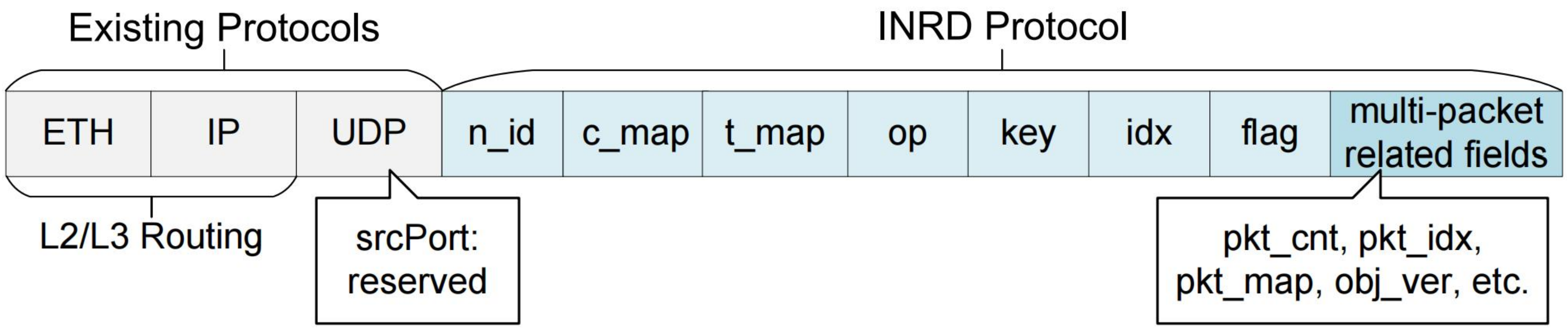
- We propose the **I**n-**N**etwork **R**equest **D**eduplication (INRD) protocol, an application-layer protocol using a specific UDP source port.
  - **n_id**: the target node ID
  - **c_map**: the one-hot encoding of the client ID
  - **t_map**: the one-hot encoding o the thread ID
  - **op**: the operation type (e.g. read request, read reply, write request, write reply)



Tsinghua University

Packet header format

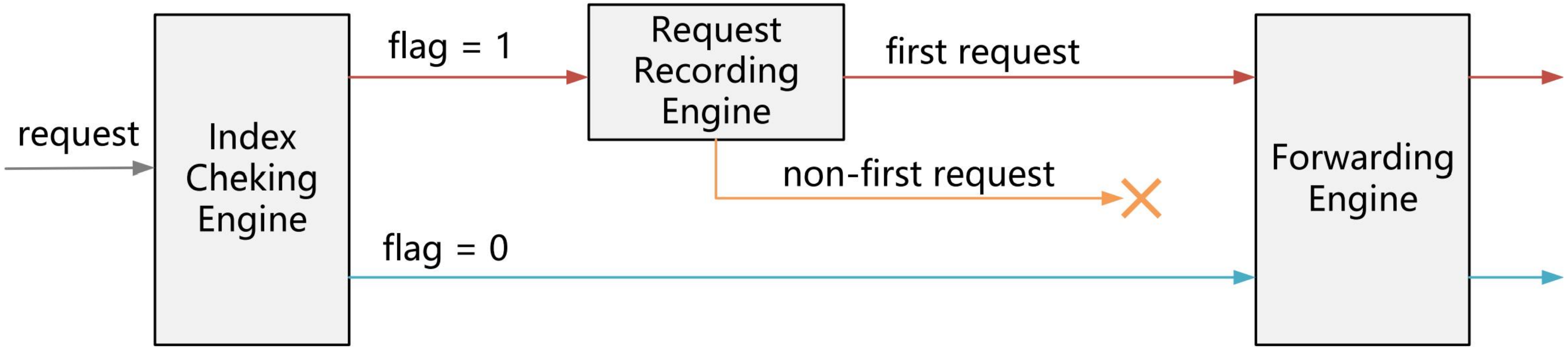
- We propose the **In-Network Request Deduplication** (INRD) protocol, an application-layer protocol using a specific UDP source port.
  - **key**: the target key (fixed length, 32 bits)
  - **idx**: the record index within the switch (hash the 32-bit key to 16 bits)
  - **flag**: “0”: direct forwarding; “1”: executing deduplication logic (client default value)
  - **multi-packet related fields**: to handle multi-packet objects’ requests



Tsinghua University

Switch request processing

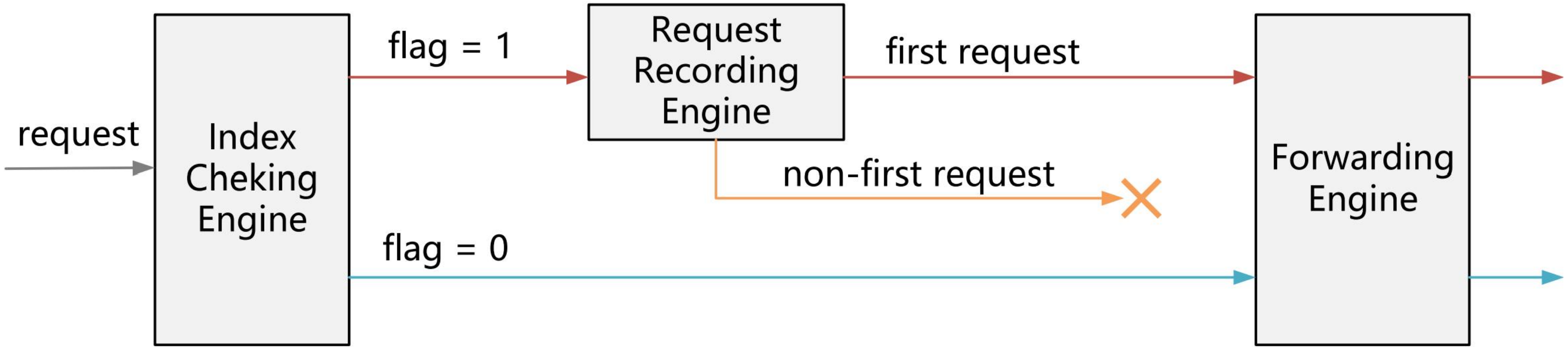
- **index checking engine**: state register array + key register array
  - If the state register value is “0”, set the state register to “1”, set the key register to the key field value, and mark the request as “first request”.
  - If the state register value is “1”, when the key register value mismatches the key field, set the flag field to “0” and forward the request packet directly.



Tsinghua University

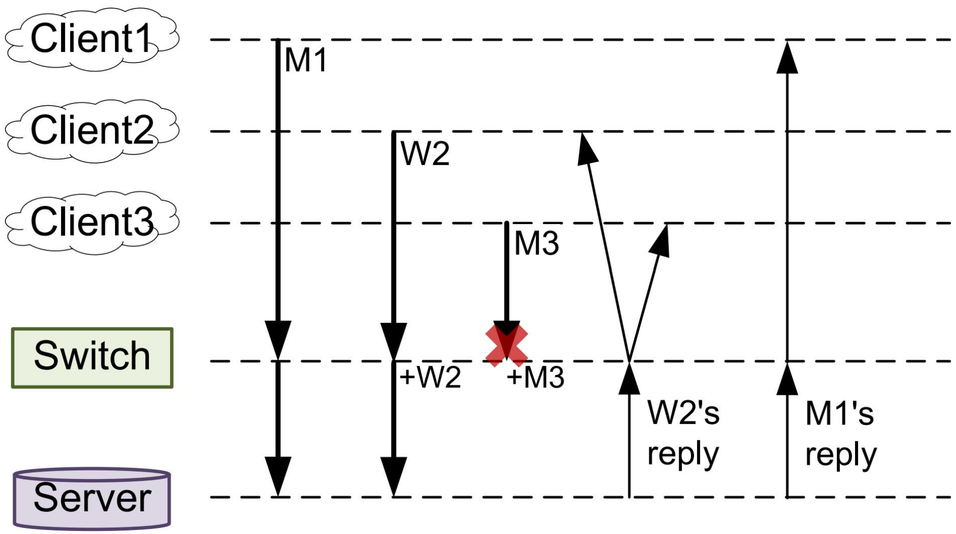
Switch request processing

- **request recording engine**: client bitmap register array + thread bitmap register array
  - The client bitmap register and the corresponding thread bitmap register are updated with the c_map and t_map fields using the bitwise OR operation.
  - If the request is marked as “first request”, forward it to the storage server.
  - If the request is not marked as “first request”, drop it directly.
- **forwarding engine**: set forwarding information for the packet



Complex scenario #1: read-modify-write request

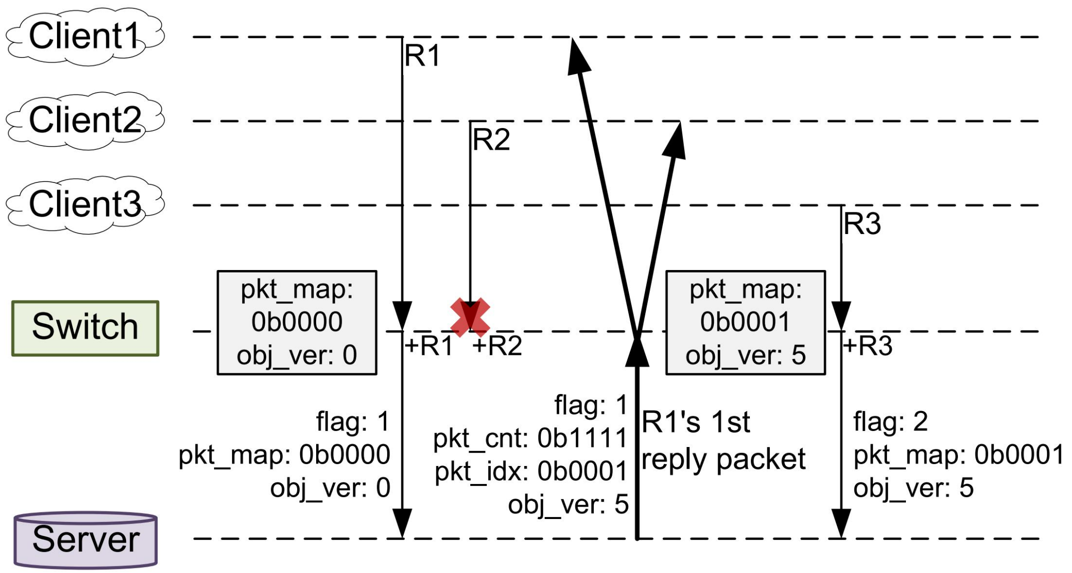
- CableCache integrates RMW requests into the write request deduplication module.
- RMW requests can’t be marked as “first request” to guarantee linearizability.
- The switch will set the flag field to “0” and directly forward the RMW request **M1**.
- The first write request **W2** will be recorded in the switch and sent to the storage server.
- The RMW request **M3** will be recorded and then dropped.



Tsinghua University

Complex scenario #2: multi-packet object

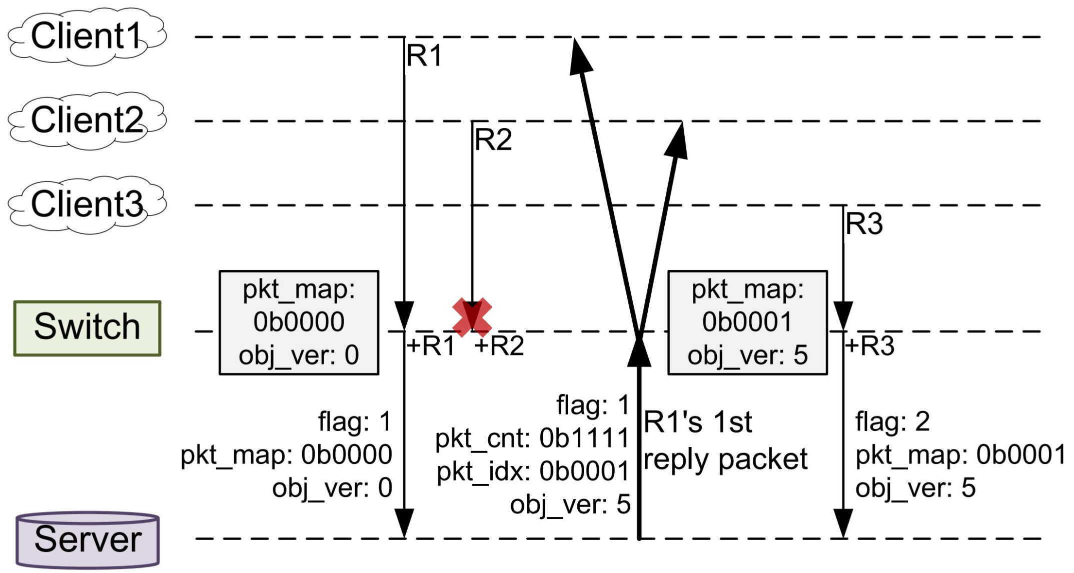
- CableCache employs a **compensatory read mechanism** to handle multi-packet objects' read requests.
- The switch maintains a **pkt_map register array** and an **obj_ver register array**, representing received reply packet IDs' bitmap and the object version number.



Tsinghua University

Complex scenario #2: multi-packet object

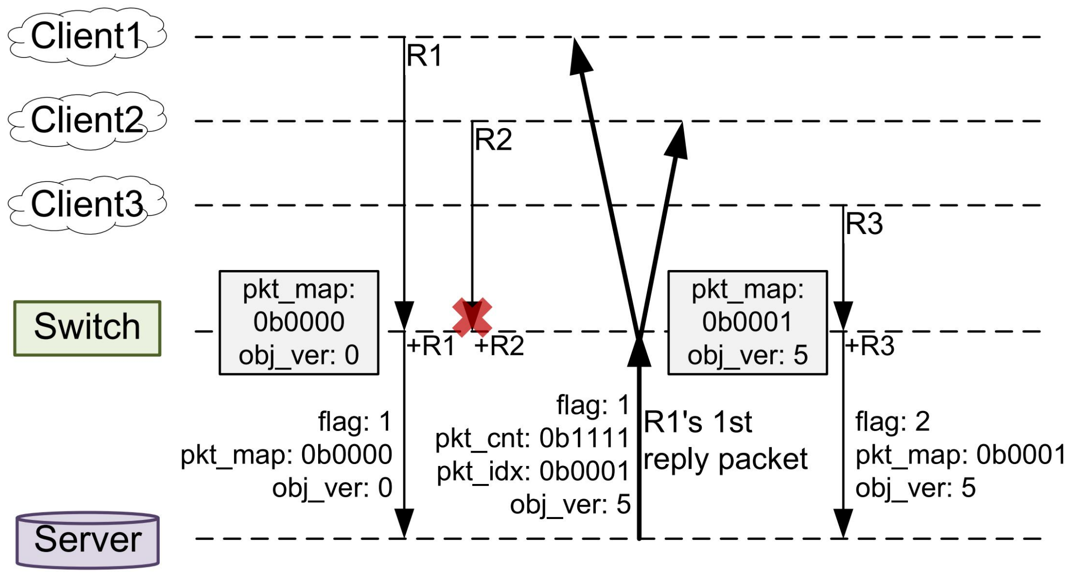
- When the non-first read request arrives, if the pkt_map register value is “0”, the request will be recorded and dropped (e.g., **R2**).
- Otherwise, the pkt_map and obj_ver registers will be set, and the flag field will be set to “2”, which means the compensatory read mechanism should be used (e.g., **R3**).



Tsinghua University

Complex scenario #2: multi-packet object

- The storage server checks whether its recorded object version number is ahead of the packet's obj_ver field.
- If not, the server only returns the reply packets indicated by the pkt_map field.
- Otherwise, the server must return all reply packets.



Discussion #1: variable-length key

- The variable-length key needs to be mapped to 32 bits using a hash function.
- The original key must be included after the header.
- If a client receives the reply packet mismatching its target key, it should resend the request while bypassing the request deduplication logic.

Discussion #2: packet loss

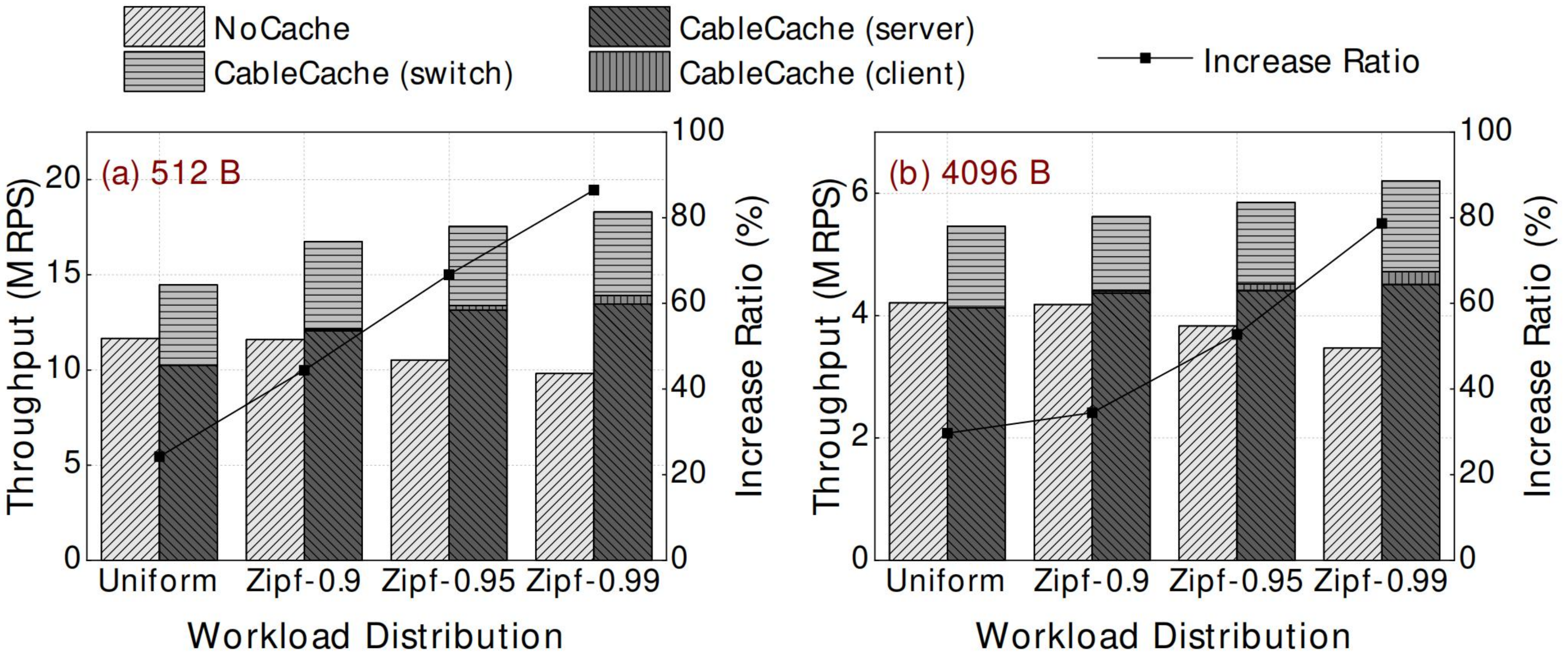
- A timeout mechanism can be used by clients to address packet loss issues.
- A timestamp checking mechanism can be introduced to prevent the issue of record indexes being continuously occupied.

Methodology

- **Testbed**
  - 6 hosts are connected by a Barefoot Tofino Wedge 100BF-32X switch.
  - 4 hosts act as clients, each running 12 threads.
  - 2 hosts each run 12 threads to simulate 24 storage servers.
- **Default workload**
  - **keyspace**: 100 MB, **Zipfian parameter**: 0.99
  - **proportion of request types**: 90% (read) : 8% (write) : 2% (CAS)
- **Default configuration**
  - **MTU**: 1092 B, **storage backend**: Redis, **cache record count**: 65536
  - **object value size**: 512 B (single-packet) and 4096 B (multi-packet)
  - **coroutine count per client thread**: 16

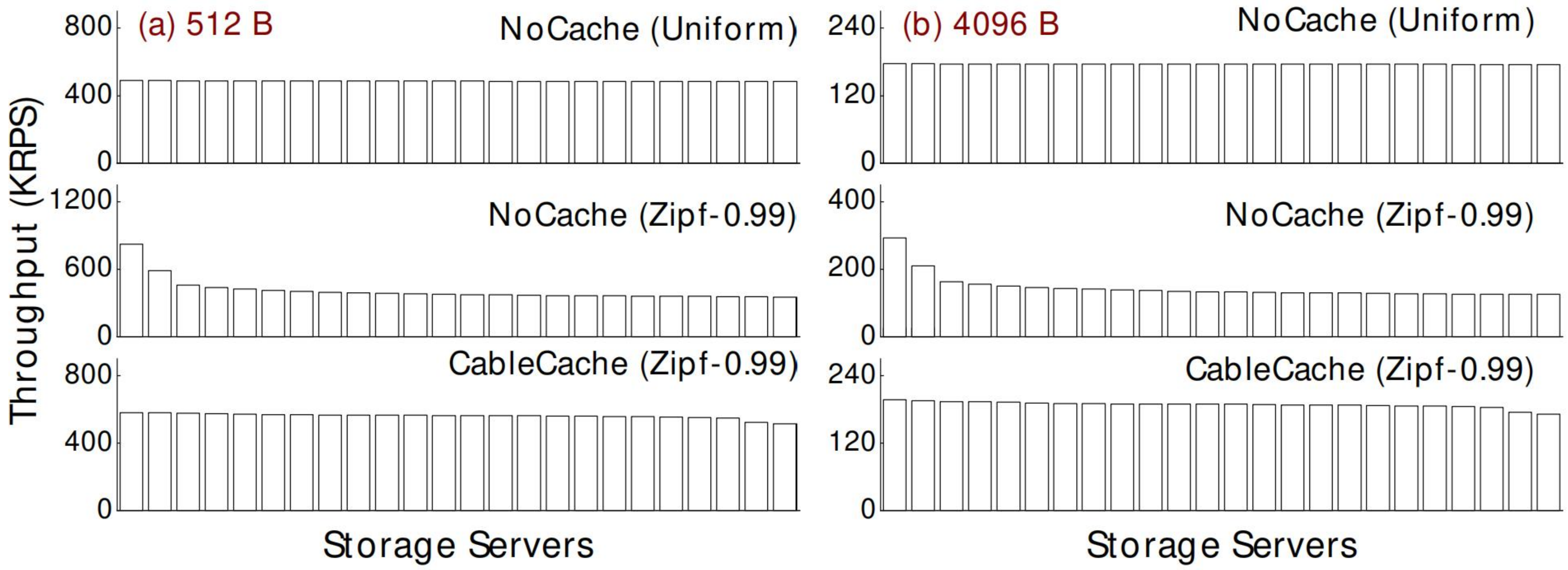
Throughput analysis

- The increase ratio of throughput grows significantly as the workload skew intensifies.
- Higher workload skew exacerbates load imbalance across storage servers, leading to more frequent request collisions.



Throughput analysis

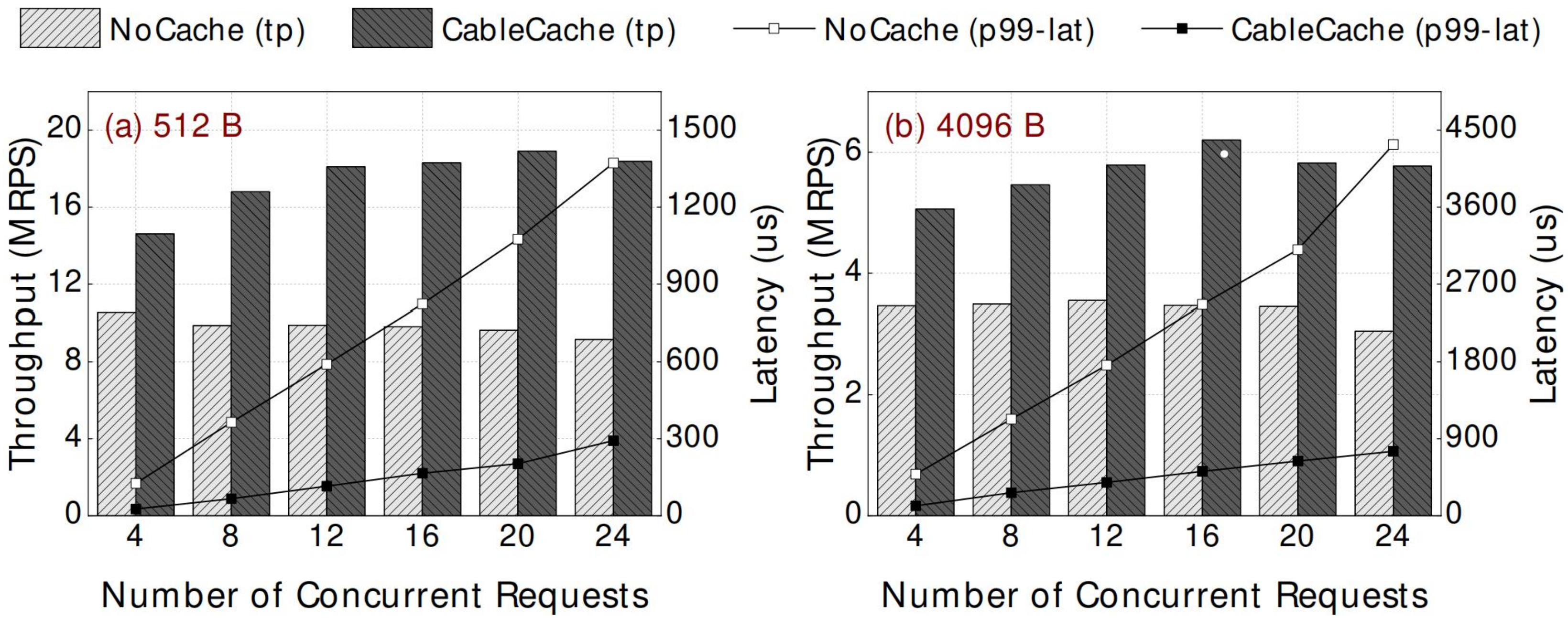
- The figure presents the load distribution across different storage servers, sorted in descending order.
- CableCache significantly improves load balancing across storage servers.



Tsinghua University

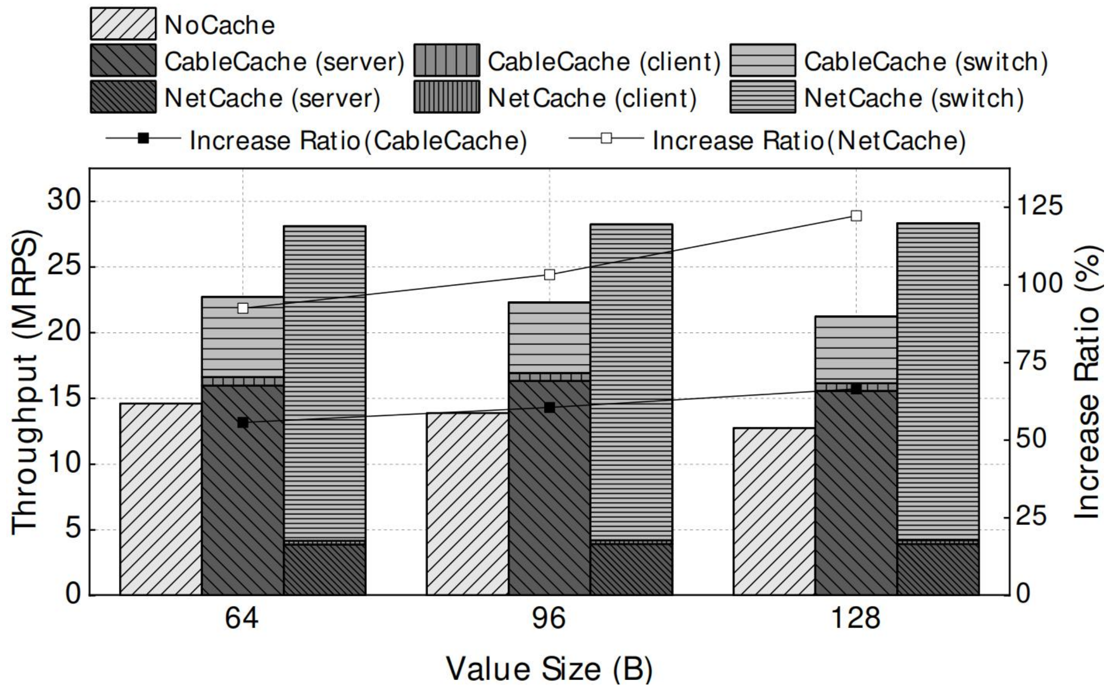
Latency analysis

- The number of concurrent requests equals to the number of coroutines per thread.
- CableCache can mitigate tail latency escalation caused by higher concurrency levels effectively.



Handling small objects

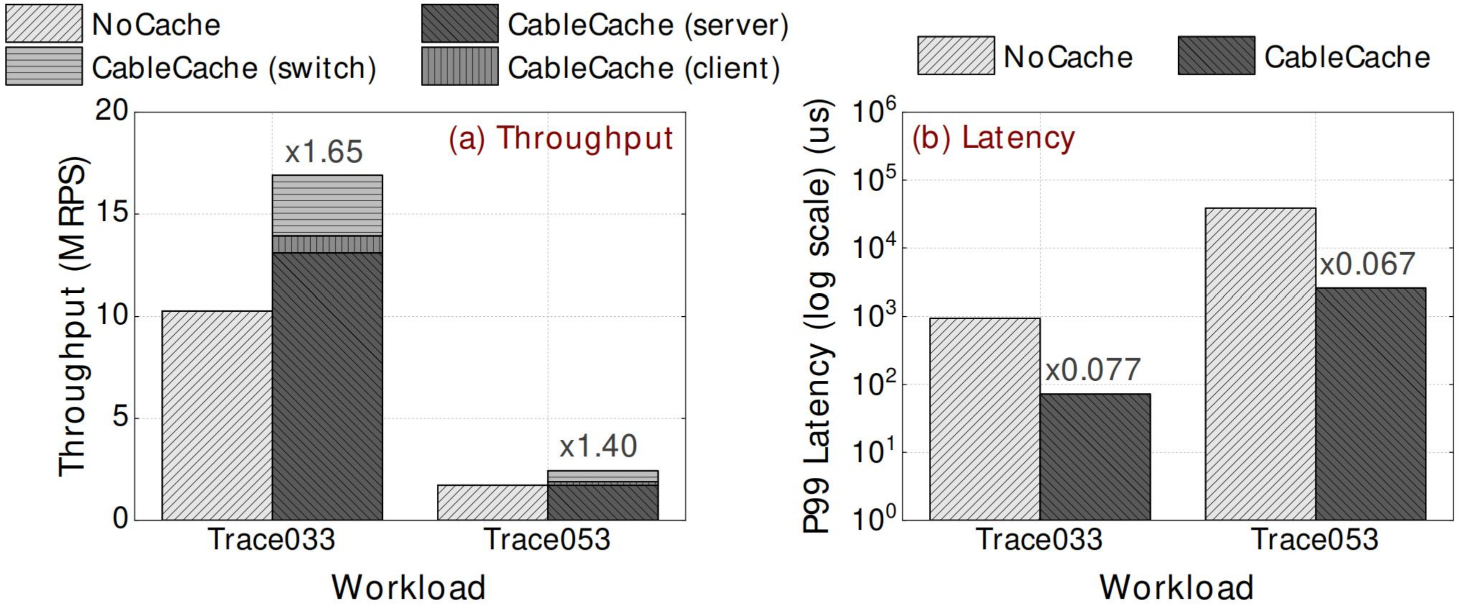
- Employ a read-only workload to conduct a comparison between **CableCache** and **NetCache**.
- NetCache enables the majority of small objects to be cached within switches.
- CableCache still requires forwarding most requests to storage servers, resulting in a lower increase ratio of throughput.



Tsinghua University

Handling real workloads

- Select **Trace033** (average value size: 1118 B) and **Trace053** (average value size: 9213 B) from two different clusters in Twitter's workload.
- CableCache improves system-wide throughput and reduces the P99 latency effectively under real workloads.



CableCache: an in-network request deduplication system

- Leverages programmable switches to maintain a directory of object request information.
- Introduces the INRD protocol to deduplicate hot object requests.
- Effectively handles complex scenarios such as the presence of read-modify-write requests and multi-packet target objects.

Experimental results show CableCache's effectiveness

- Alleviates load imbalance under both synthetic and real workloads.
- Improves system-wide throughput, and reduces tail latency of requests.

Thanks for Listening!
Q & A

Contact: huangjw22@mails.tsinghua.edu.cn