ELECT: Enabling Erasure Coding Tiering for LSM-tree-based Storage

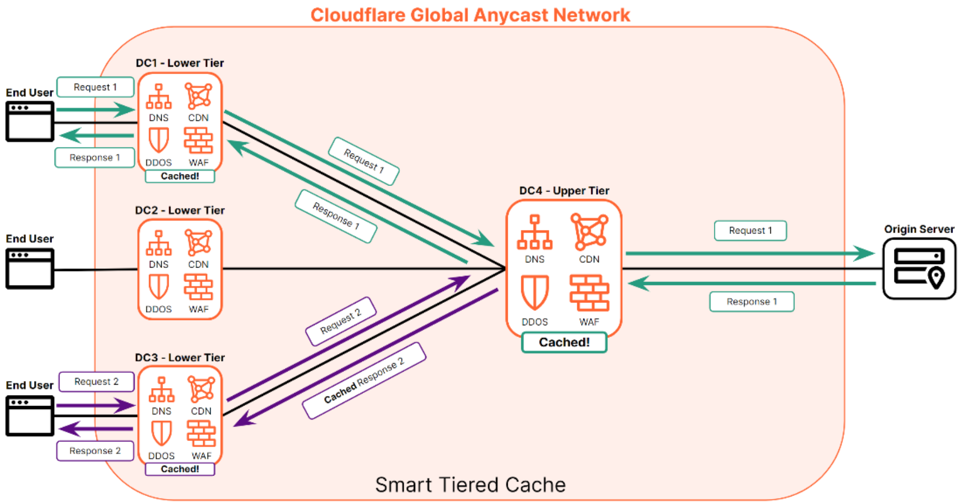
Yanjing Ren[1], Yuanming Ren[1], Xiaolu Li[2], Yuchong Hu[2], Jingwei Li[3], **Patrick P. C. Lee**[1]

[1]The Chinese University of Hong Kong
[2]Huazhong University of Science and Technology
[3]University of Electronic Science and Technology of China

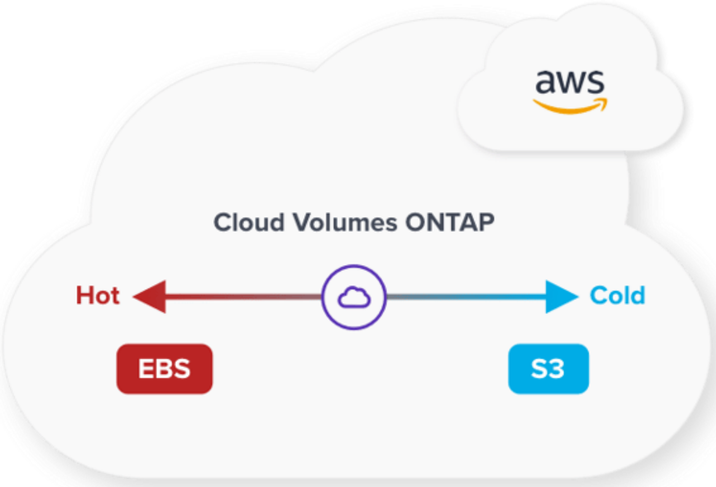
USENIX FAST 2024

Storage Tiering

- Storage tiering balances the trade-off between access performance and storage persistence
  - **Hot tier**: high performance but limited storage space
  - **Cold tier**: abundant storage space but lower performance



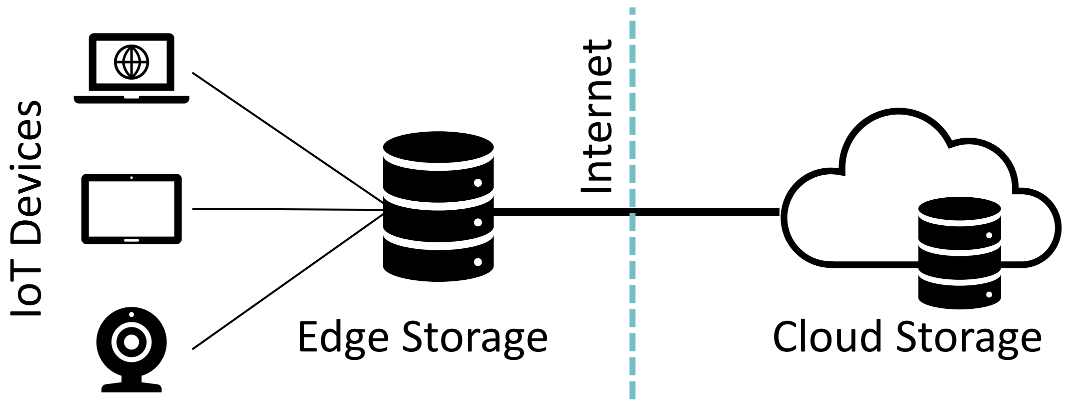
Cloudflare CDN



AWS Cloud Volumes

Storage Tiering

- ➢ A primary use case of storage tiering: **edge-cloud storage**
  - IoT applications will generate over 79.4 ZB data in 2025 [*]



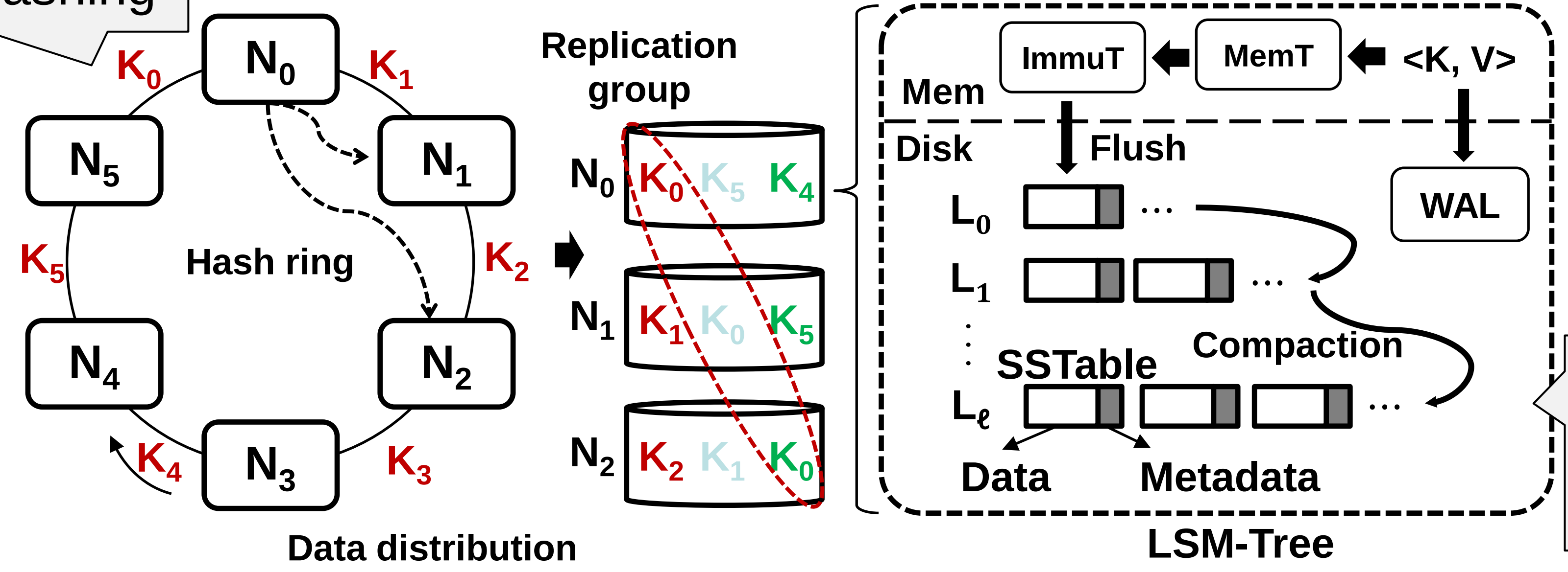
- ➢ Edge storage mainly builds on **distributed key-value (KV) stores**
- ➢ Our goal: Extend a distributed KV store with storage tiering for edge-cloud and general tiered storage environments

[*] https://blogs.idc.com/2019/11/04/how-you-contribute-to-todays-growing-datasphere-and-its-enterprise-impact/

Distributed KV Store

- Use Cassandra as an example for edge storage
  - Cassandra is decentralized, high-performance, and fault-tolerant

KV partitioning via consistent hashing



Internal KV storage via Log-structured Merge tree (LSM-tree)

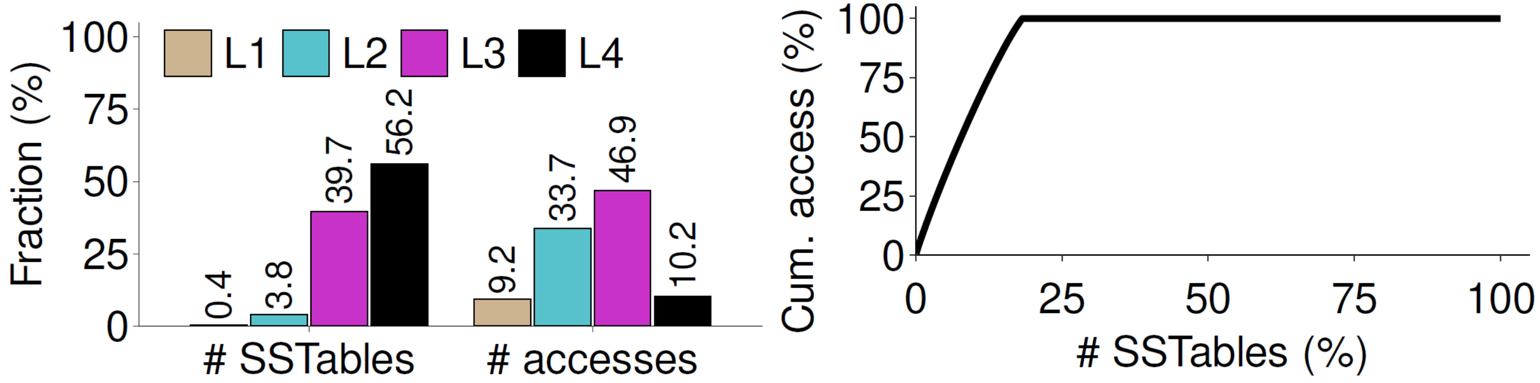
- **Drawback: Replication has high storage overhead**

Erasure Coding

- *(n, k)* Reed-Solomon (RS) codes:
  - Encode *k* data chunks to *n-k* parity chunks
  - Each collection of *n* data/parity chunks forms a coding group
  - With a redundancy of *n/k*, any *k* out of *n* chunks can recover lost data
- Compared with replication, erasure coding
  - ✓ Saves significant storage overhead
  - ✗ Incurs higher degraded read and full-node recovery overhead
- **Can we maintain storage efficiency via erasure coding, while preserving high performance, in tiered storage?**

Skewed Access Patterns

- Practical KV workloads have skewed access patterns



(a) Statistics across levels (b) Access distributions in L_4

10-node Cassandra cluster
- Load 100M 1-KiB KV pairs
- Issue 1M reads
- Keys accessed under Zipf (0.99)

- 56.2% of SSTables are stored in L_4, but only have 10.2% of accesses
- Only 18.2% of SSTables in L_4 are accessed

Our Contributions

ELECT: a distributed LSM-tree-based KV store that <u>e</u>nab<u>l</u>es <u>e</u>rasure <u>c</u>oding <u>t</u>iering

- Extends LSM-tree with hybrid redundancy
  - Replicates hot KV pairs and erasure-codes cold KV pairs in hot tier
  - Offloads (selectively) cold KV pairs to cold tier for further hot-tier storage savings
- Key techniques
  - LSM-tree-based redundancy transitioning
  - Hotness-aware redundancy transitioning and cold-data offloading
  - Tunable configuration for balancing storage-performance trade-off
- Results: **56.1%** less hot-tier storage overhead than replication, with similar normal read/write performance

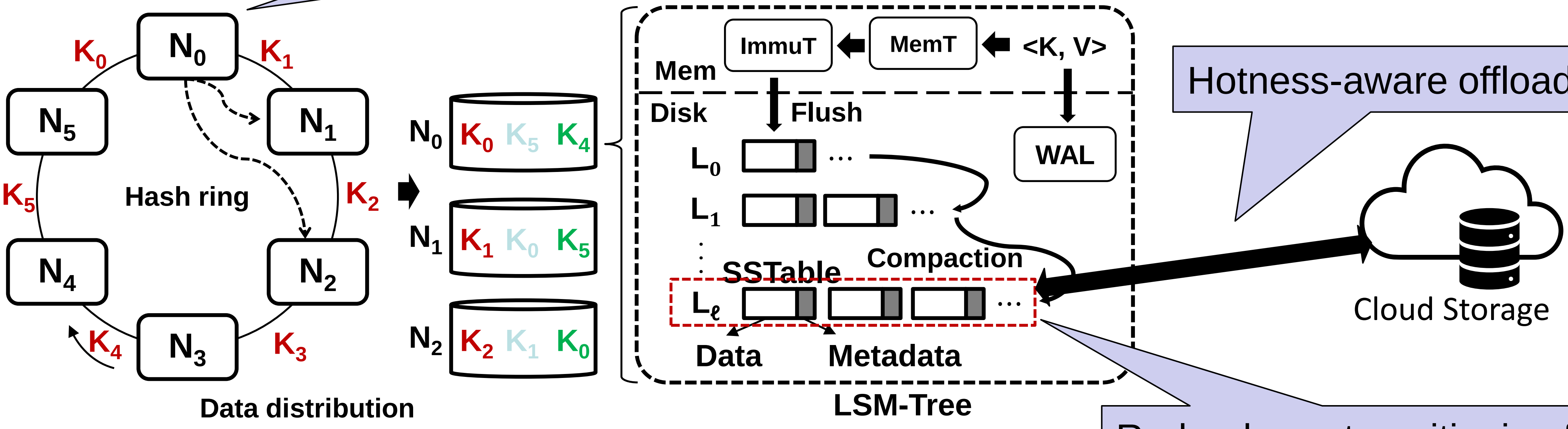
Design Considerations

- **Q1:** At what granularity should KV pairs be encoded?
- **Q2:** Should erasure coding be performed on or off the write path?
- **Q3:** How should skewed access patterns be addressed?
- **Q4:** How should the access overhead in the cold tier be mitigated?
- **Q5:** How should ELECT address the trade-off between storage savings and access performance?

Design Overview

Tunable configuration of storage-performance trade-off

Hotness-aware offloading

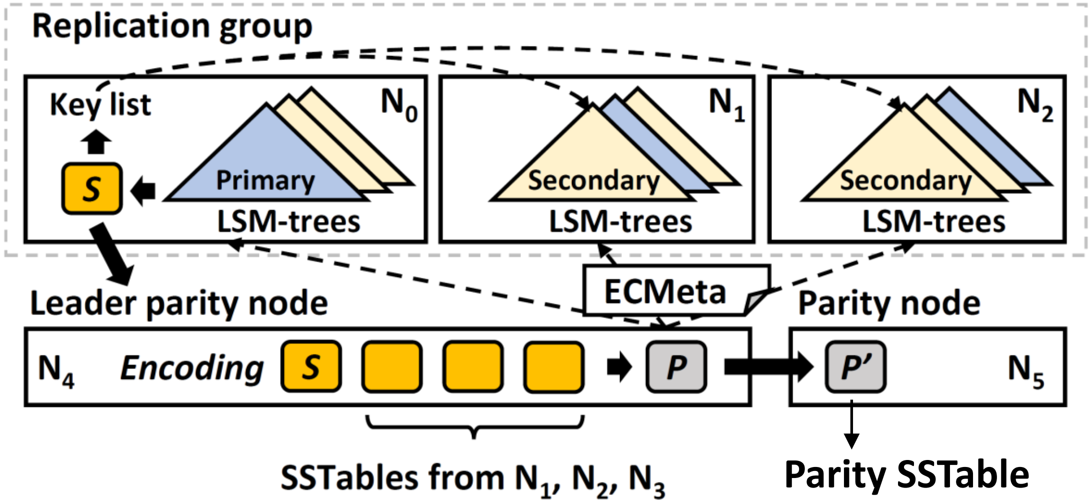


Redundancy transitioning for last LSM-tree level

LSM-tree-based Redundancy Transitioning

- Q1: At what granularity should KV pairs be encoded?

Apply cross-encoding to SSTables in last LSM-tree level



- **Decoupled replication management** [DEPART, FAST'22; Tebis, EuroSys'22]:
  - Each node separates *R* replicas into a primary LSM-tree and *R-1* secondary LSM-trees → facilitates secondary replica removal

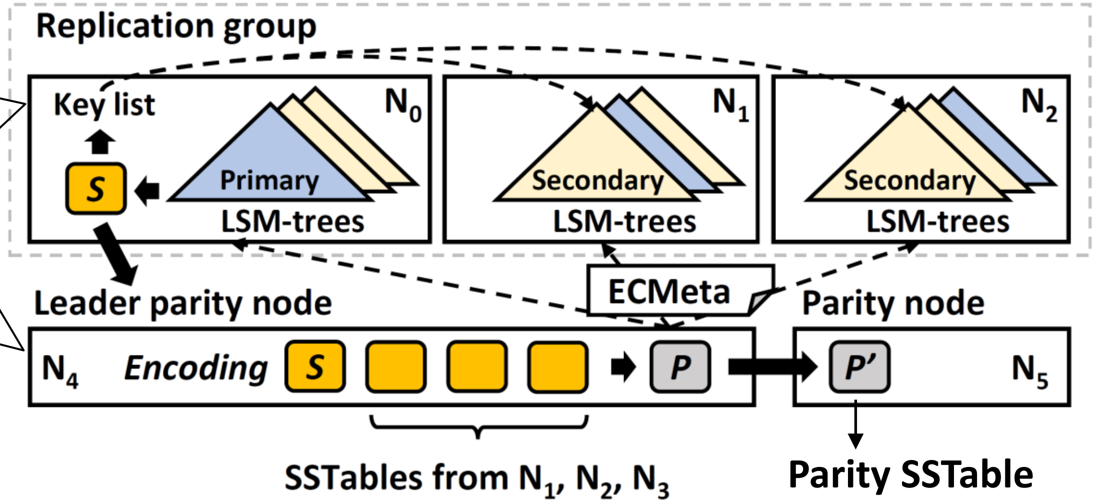
LSM-tree-based Redundancy Transitioning

- Q2: Should erasure coding be performed on or off the write path?

Apply erasure coding offline

Secondary replica removal based on key list

Leader parity node encodes *k* data SSTables into *m* parity SSTables



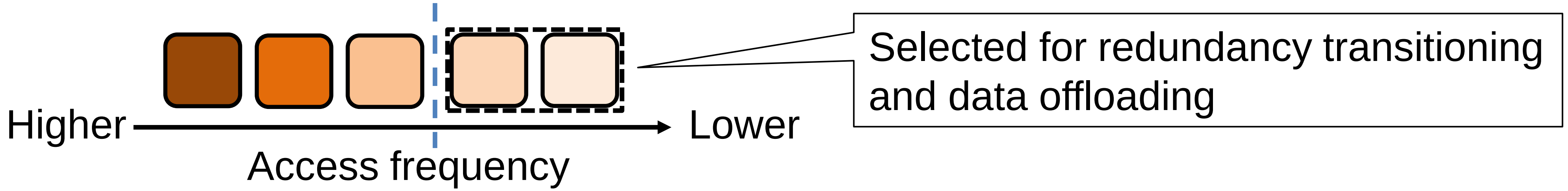
- **Decentralized parity node selection**:
  - Maintains load balancing and fault tolerance for parity SSTables with decentralized placement decisions

Hotness Awareness

- Q3: How should skewed access patterns be addressed?

Hotness-aware redundancy transitioning

  - Sort last-level SSTables by access frequency



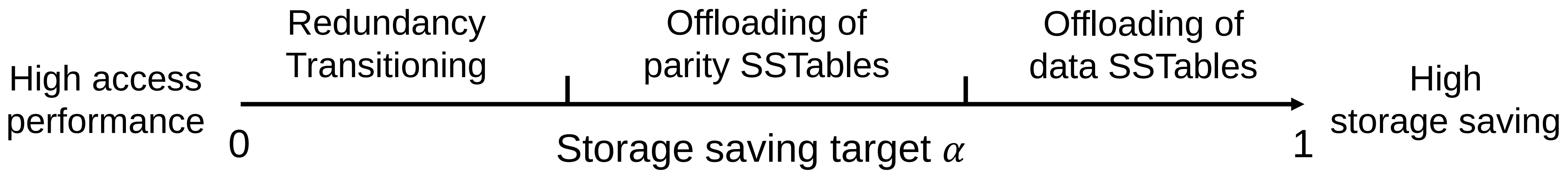
- Q4: How should the access overhead in the cold tier be mitigated?
  - Offload parity SSTables with long lifetime
  - Offload data SSTables with low access frequency and long lifetime

Balancing Storage-Performance Trade-off

- Q5: How should ELECT address the trade-off between storage savings and access performance?

Control redundancy transitioning and cold-data offloading with a user-specified storage saving target α

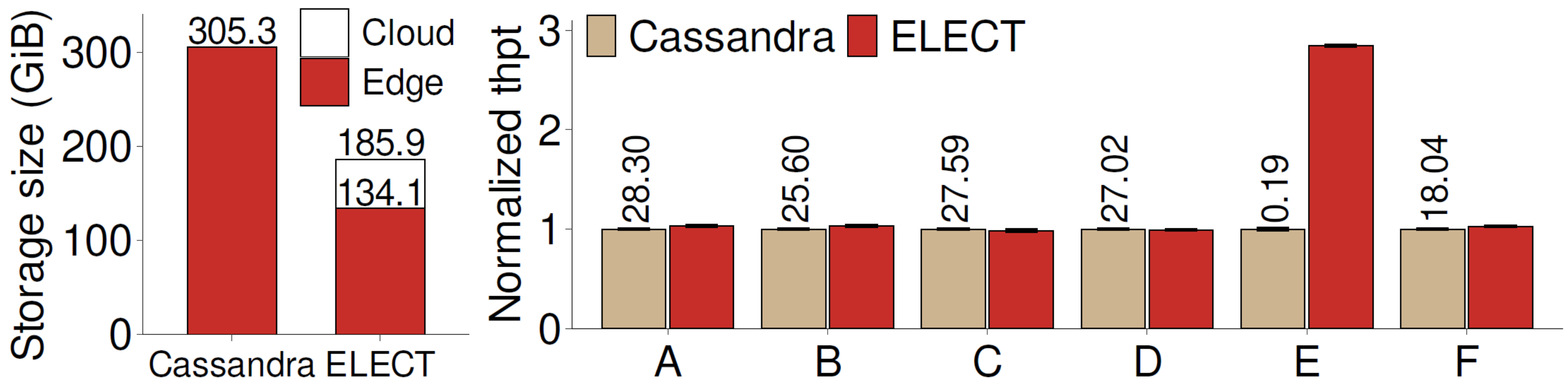
- Quantify storage overhead and control how many SSTables are encoded and offloaded in a step-by-step manner



Experimental Setup

- Alibaba Cloud in an edge-cloud setting:
  - 12 instances (10 edge nodes + 2 client nodes) with 3 Gb/s connectivity
  - Alibaba Object Storage in a different region as cold-tier
    - Edge-to-edge: 1ms; edge-to-cloud: 45ms
  - **Workloads:** YCSB 0.17.0, 1-KiB KV pairs, Zipfian distribution (0.99)
  - **Default settings:** 3-way replication, (6,4) encoding, storage saving target $\alpha = 0.6$, write consistency level ALL, read consistency level ONE
- Compare Cassandra (v4.1.0) and ELECT
  - Average over 5 runs and 95% confidence interval under Student's t-dist
- **Summary of results**: ELECT saves storage overhead of Cassandra while maintaining high performance in normal mode

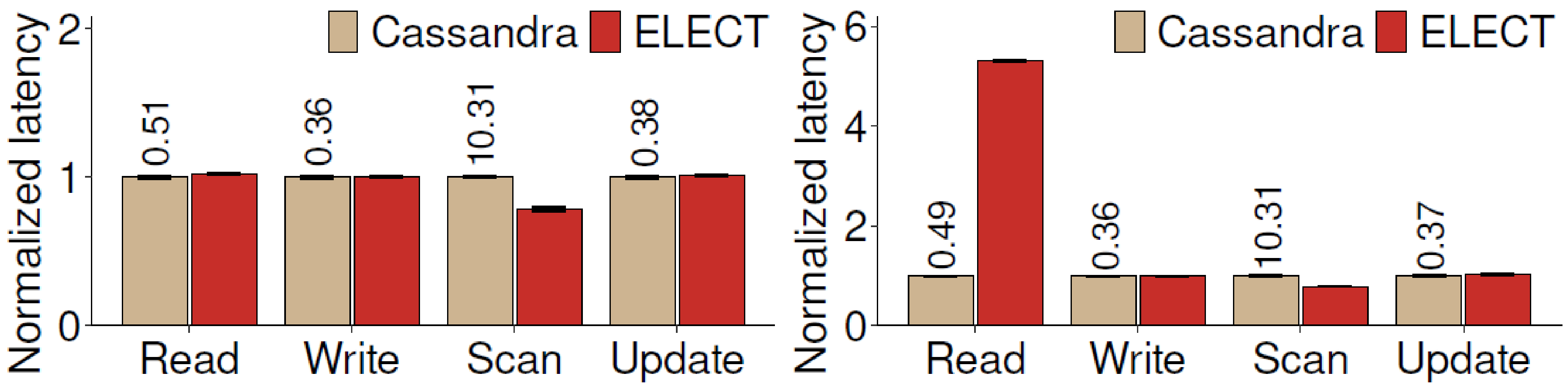
YCSB Core Workloads



(a) Storage size (b) Throughput

- ELECT achieves **56.1%** edge storage saving from Cassandra
  - 39.1% overall storage savings (in both edge and cloud)
- ELECT outperforms Cassandra in workload E (scan-intensive) by **2.84x** due to decoupled replication management
  - Similar throughput for other workloads (up to 3% differences)
  - Note that the improvement is less on Chameleon Cloud

Individual KV Operations



(a) Normal mode (b) Degraded mode

- ELECT maintains performance in normal mode, but has high latency in reads in degraded mode
  - Due to retrieval of parity SSTables from the cold tier
  - On Chameleon Cloud, the overhead is reduced from 5x to 1.2x

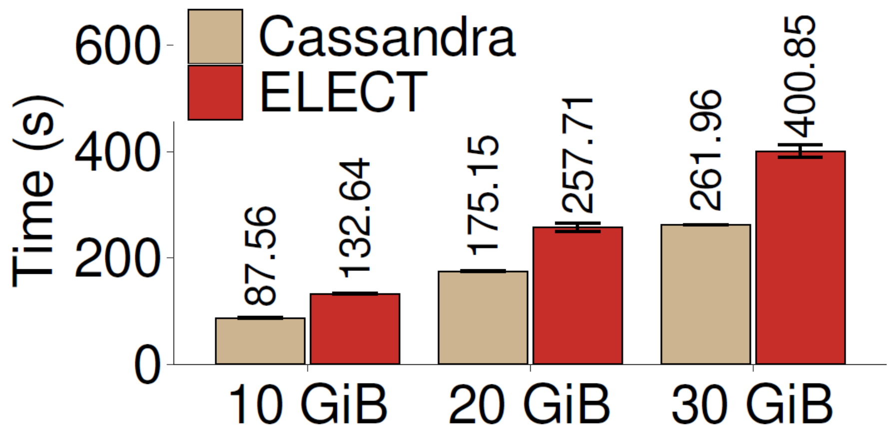
Performance Breakdown

- Redundancy transitioning and cold-data offloading have long processing time
  - Performed offline with limited overhead
  - ELECT maintains write performance as Cassandra

Steps	Cassandra	ELECT
Write		
WAL	21.32 ± 0.76 ms	21.84 ± 0.28 ms
MemTable	37.98 ± 1.73 ms	40.84 ± 0.13 ms
Flushing	16.95 ± 0.29 ms	17.70 ± 0.18 ms
Compaction	205.87± 2.21 ms	169.03 ± 3.23 ms
Transitioning	-	239.05 ± 2.69ms
Offloading	-	162.84 ± 12.05 ms
Read in normal mode		
Cache	17.05 ± 0.27 ms	18.35 ± 0.34 ms
MemTable	20.78 ± 0.95 ms	23.20 ± 0.61 ms
SSTables	182.69 ± 2.53 ms	177.55 ± 0.60 ms
Read in degraded mode		
Cache	17.41 ± 0.33 ms	18.75 ± 0.18 ms
MemTable	21.54 ± 0.66 ms	23.38 ± 0.46 ms
SSTables	184.39 ± 1.67 ms	184.14 ± 2.35 ms
Recovery	-	1957.64 ± 34.16 ms

Average latency of processing 1MiB of writes/reads and 95% confidence interval under Student's t-distribution

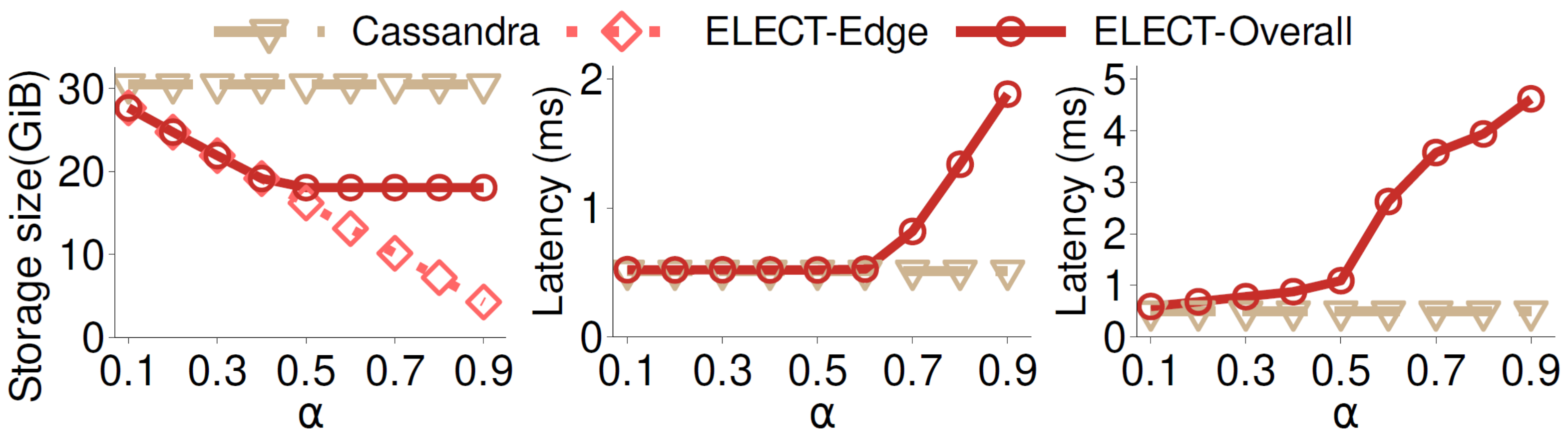
Full-node Recovery



Steps	Time
Copy	13.54 ± 0.22 s
Retrieve	373.98 ± 11.61 s
Decode	13.34 ± 0.48 s

- ELECT incurs **50%** higher recovery time than Cassandra
  - ELECT retrieves data and parity SSTables to decode lost SSTables in primary LSM-tree
  - Recovery performance is network-bounded

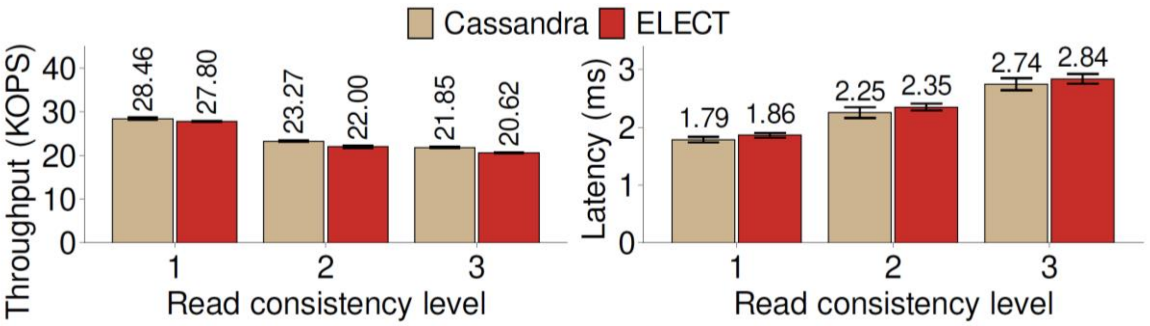
Impact of α



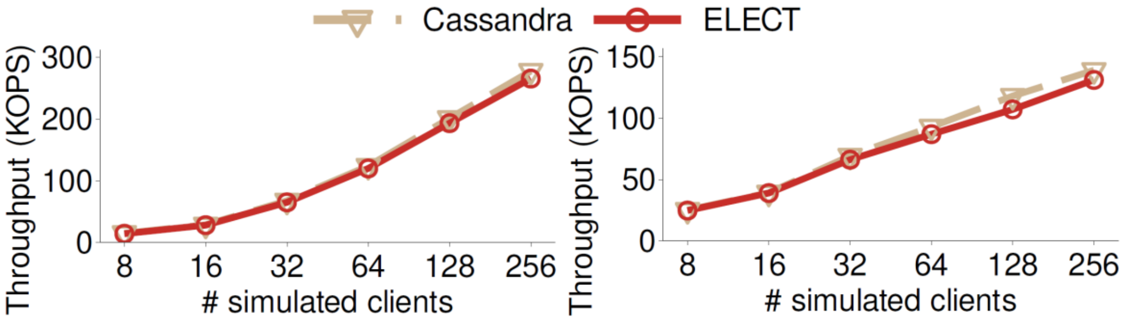
(a) Storage size (b) Reads in normal mode (c) Reads in degraded mode

- ELECT reduces edge storage overhead by **9.2 - 86%** over Cassandra when α increases from 0.1 to 0.9
  - ~4% difference from α due to metadata overhead
- ELECT maintains read latency in normal mode before offloading data SSTables (i.e., $\alpha \leq 0.6$)

Consistency and Scalability



(a) Read throughput (b) 99th-percentile latency



(a) Reads in normal mode (b) Writes in normal mode

- ELECT maintains consistent read performance as Cassandra
- ELECT has **4% (5.7%)** less normal read (write) throughput than Cassandra due to redundancy transitioning and offloading
- More results in our paper: resource utilization, impact of KV sizes, coding parameters

Conclusions

- **ELECT**: a distributed KV store that enables erasure coding tiering
  - LSM-tree-based redundancy transitioning (with decentralized parity node selection)
  - Hotness-aware redundancy transitioning and cold data offloading
  - Tunable configuration for balancing storage-performance trade-off
- Source code: **https://github.com/adslabcuhk/elect**

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED