Load-Aware Asynchronous Tiering in Distributed Object Storage: Replica-to-EC Migration Study and Implementation

Department of Computer Science and Information Engineering, NCKU
Authors: Shuo Huang **Advisor:** Prof. Hung-Chang Hsiao

01 Abstract

This project implements a distributed object storage prototype with hot replication and cold erasure-coded storage. It studies how background replication-to-erasure-coding migration affects foreground PUT/GET latency under different system pressures. Compared with age-based and throttled migration, the pressure-aware policy uses node metrics such as CPU usage, memory usage, I/O wait, queue depth, and node health to decide when migration should run. GKE experiments show that I/O pressure is the main bottleneck: pressure-aware tiering reduces migration progress but better protects foreground tail latency.

02 Motivation

Real Systems

Modern object storage systems already use automatic tiering, erasure coding, or cache-tier movement. Amazon S3 Intelligent-Tiering and GCS Autoclass automatically move objects across access or storage tiers, while Ceph cache tiering includes background flush / evict behavior. MinIO and Ozone also show that erasure coding is a practical storage layout, not only a theoretical optimization.

Timing Matters

These systems prove that tiering and erasure coding matter, but they also reveal a deeper systems problem: background data movement consumes disk I/O, CPU, network bandwidth, and metadata resources. If migration, flush, repair, or tier conversion runs during foreground PUT/GET traffic, storage efficiency may improve while tail latency becomes worse.

Our Focus

This project uses replication-to-erasure-coding migration as a controllable case study of background data movement. By building an observable prototype, we compare Age-based, Throttled, and Pressure-aware policies under the same workload. The goal is to evaluate whether load-aware admission control can decide when background tiering should run without causing excessive foreground interference.

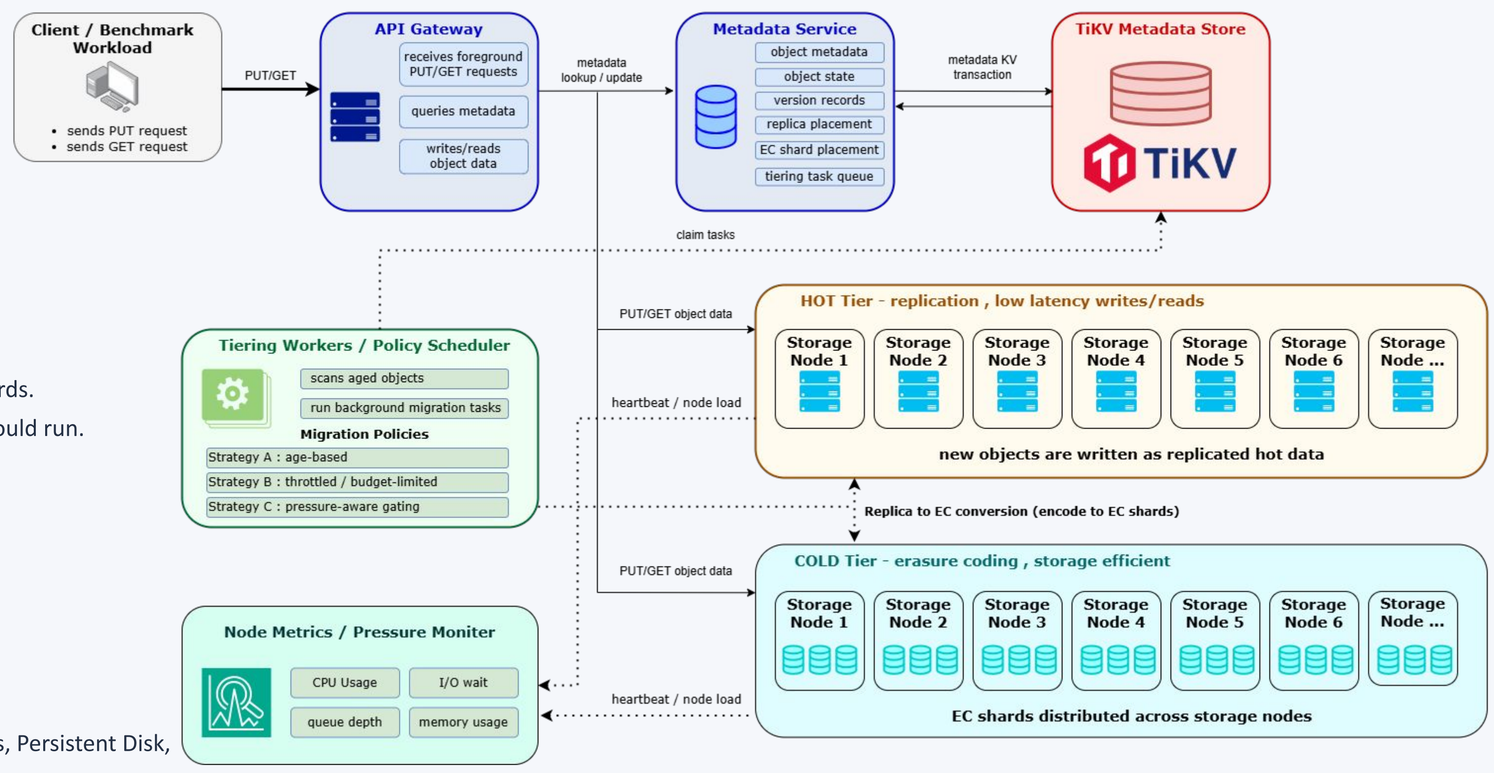
03 System Architecture

Request & Replication Flow

Request & Tiering Flow

1. Client sends foreground PUT/GET requests to the API Gateway.
2. New objects are written to the hot tier using replication.
3. Metadata Service records object state, version, and placement.
4. Tiering Worker scans aged objects and schedules migration tasks.
5. Background migration converts replicated objects into erasure-coded shards.
6. Pressure-aware policy uses node metrics to decide whether migration should run.

Language : Go
Metadata Store: TiKV / PD
Storage Model: Hot replication + cold erasure coding
Orchestration: Docker, Kubernetes, GKE
Benchmark: In-cluster PUT/GET workload
Pressure Injection: stress-ng
Metrics: CPU usage, memory usage, I/O wait, queue depth, node health
Evaluation Setup: 6 storage nodes + system node pool, n2-standard-2 nodes, Persistent Disk, Kubernetes.



04 Evaluation

The prototype was deployed on GKE. CPU and I/O pressure were injected with stress-ng jobs scheduled inside the Kubernetes cluster. Each run used 50 objects of 1 MiB, a 60-second mixed workload, concurrency 2, and a 70% GET / 30% PUT ratio. We compared Baseline, Age-based, Throttled, and Pressure-aware policies under no pressure, CPU pressure, and I/O pressure. Pressure was injected using stress-ng jobs. We measured foreground p99 latency, throughput, error rate, and completed migration tasks.

05 Results

In the GKE in-cluster evaluation, the baseline p99 latency stayed in the tens of milliseconds and was used as the reference for comparing tiering policies. Overall, the strongest interference appeared under I/O pressure, showing that replication-to-erasure-coding migration competes with foreground PUT/GET requests for storage I/O resources.

In the representative I/O pressure run, Age-based and Throttled completed 66 tasks, but showed higher foreground p99 latency. Pressure-aware completed only 16 tasks, while keeping GET / PUT p99 at 55.6 ms / 52.1 ms.

This shows that Pressure-aware tiering does not blindly maximize migration throughput. Instead, it sacrifices part of the migration progress under system pressure to protect foreground request latency.

Conclusion

This project does not aim to replace existing object storage systems. It uses replication-to-erasure-coding migration as a case study for a broader tiered-storage problem: when background data movement should run.

Results show that I/O pressure is the main source of interference. Pressure-aware tiering acts as admission control, sacrificing migration progress when the system is busy to better protect foreground tail latency.

06 Future Work

This project evaluates a prototype-scale system on GKE, so the results may still be affected by cloud resource variability and limited cluster size. The current policy also decides whether migration should run, but does not yet choose the best idle source/target nodes for each migration.

Future work includes larger-scale and longer-running experiments, stronger statistical validation, node-aware migration placement, and applying load-aware scheduling to other background storage tasks such as compaction, repair, and rebalancing.