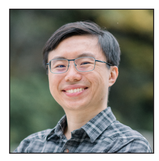
Baleen: ML Admission & Prefetching for Flash Caches

Daniel Lin-Kit Wong, Hao Wu†, Carson Molder§, Sathya Gunasekar†, Jimmy Lu†,
Snehal Khandkar†, Abhinav Sharma†, Daniel S. Berger‡, Nathan Beckmann, Gregory R. Ganger




Introduction

- Flash caches often used to reduce peak backend load
  - › Reducing backend #HDDs & servers needed
- Need to limit long-term flash write rate
  - › To avoid premature flash wearout

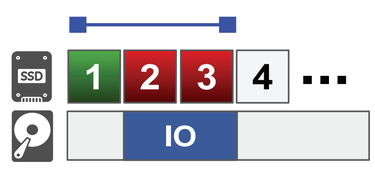
Goal: reduce peak load while avoiding excessive writes

- Use ML for cache policy decisions
- Key ideas
  - › Exploit a new cache residency model (episodes)
  - › Coordinate admission & prefetching
  - › Optimize for Disk-Head Time rather than miss rate

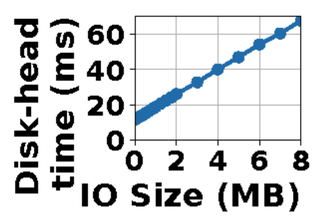
Ex: Flash Caching for Bulk Storage

- Caching minimizes backend load rather than latency

(1) Client requests a byte range
(2) Check Flash Cache for data
(3) A miss causes IO to HDD backend



- How to measure backend load?
  Small IOs: IOPS, *Large IOs:* MB/s
  Variable size IOs: Disk-head Time (DT)

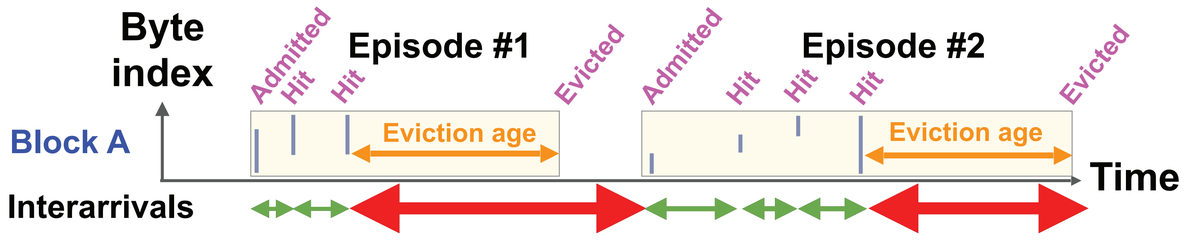


Lower peak load → Fewer HDDs → Lower TCO
Total Cost of Ownership (dominated by media costs)

Episodes: A Model for Flash Caching

- Observation: admission decisions made on misses

- Idea: group accesses temporally into episodes
  - › Episode goes from admission to eviction

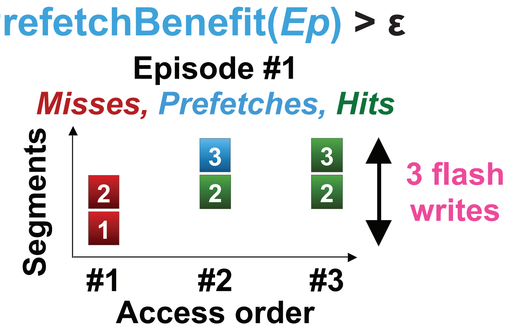


- How: model LRU cache state with assumed eviction age
  - › Split when interarrival time > eviction age

Baleen: ML for Admission & Prefetching

- OPT policies approximate optimal using episodes
  - › OPT admits episodes max. saved DT & min. flash writes
  - › OPT-Range prefetches smallest range covering episode
  - › OPT-When prefetches if PrefetchBenefit(*Ep*) > ε

$$\text{Score}(Ep) = \frac{\text{DTSaved}(Ep)}{\text{FlashWrites}(Ep)}$$



- Baleen uses OPT policies as labels to train ML policies
  - › GBM models for ML admission, ML-Range, ML-When

Features

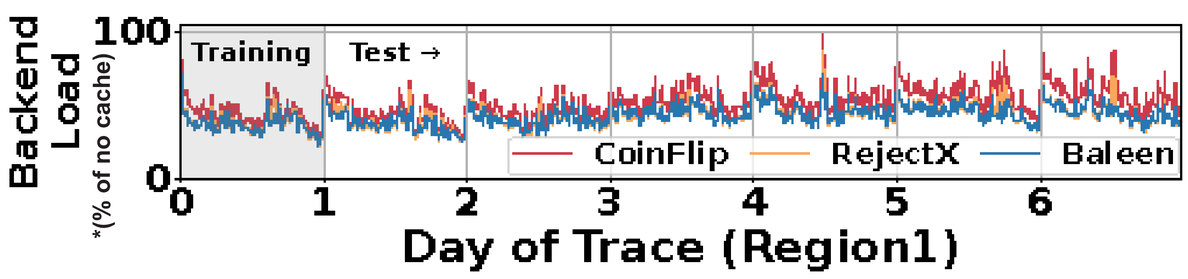
Metadata			*IO*		*Usage counters*					
ns	user	tmp	start	end	hr=1	hr=2	hr=3	hr=4	hr=5	hr=6

- Baleen-TCO optimizes flash write rate to minimize TCO

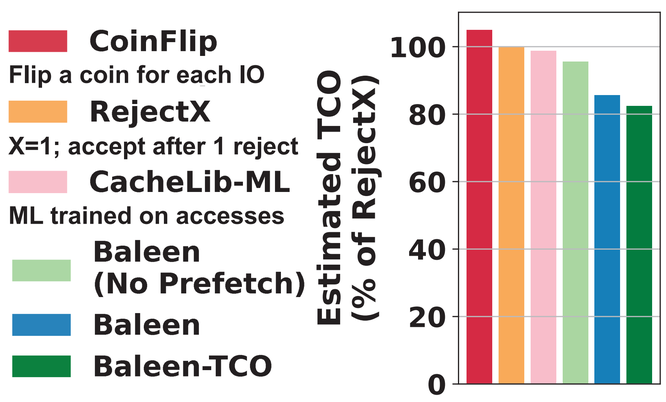
$$\mathbf{TCO_1} \propto \frac{\text{PeakDT}_\mathbf{1}}{\text{PeakDT}_0} \cdot \#\text{HDDs}_0 + \frac{\text{Cost}_{\text{SSD}}}{\text{Cost}_{\text{HDD}}} \cdot \frac{\text{FlashWR}_\mathbf{1}}{\text{FlashWR}_0} \cdot \#\text{SSDs}_0$$

Evaluation

- 7 traces from Meta's Tectonic in 2019, 2021, 2023
  - › Workloads: Data warehouse, blob storage, ... [Pan21]
- **Peak load:** P100 DT used (measured in 10 min intervals)



- Baleen-TCO reduces estimated TCO by 17%
  - › over production baselines CacheLib-ML and RejectX
- Baleen (fixed flash write rate) reduces peak load by 12%



Baleen-TCO saves	Over CacheLib-ML	Over RejectX
TCO	17%	18%
Peak load	16%	17%
Median load	2%	6%
IO miss rate	14%	14%
Byte miss rate	2%	2%

EXTRA TAKEAWAYS

1. Optimize for end-to-end metric (DT saved)
   - › Easy misstep: optimizing IO miss rate ≠ DT saved
   - › ML-Range on Every Miss good for IOs, bad for DT
   - › ML-Range on ML-When best for DT
2. Prefetching bad with bad admit decisions
   - › No reduction in peak DT with baselines
3. Still has room for improvement (OPT is 16% better)
4. GBM more efficient, DT saved on par with Transformer
5. Unsuccessful attempts: early eviction, segment-awareness, prefetch on PUT

LESSONS FROM ML IN PRODUCTION

- ML model accuracy ≠ system performance
- Encapsulate ML, cache & storage; avoid tight coupling

Carnegie Mellon University

Simulator & Traces
pdl.cmu.edu/CILES

Parallel Data Laboratory

FAST24