GL-Cache: Group-level learning for efficient and high-performance caching

Juncheng Yang*, Ziming Mao$, Yao Yue@, K. V. Rashmi*

*Carnegie Mellon University, $Yale University, @Pelikan Foundation
What location are they going?

Grouping and the context make prediction easier!

Images generated by DALL·E
Introduction

Ubiquitous caching

• Different types of caches
  • Block/page cache
  • Key-value cache
  • Object cache (CDN cache)

• Different deployments
  • Data center
  • PC/mobile phone
Introduction

Metrics of a cache system

• Efficiency
  • Measured by hit/miss ratio

• Performance
  • Measured by requests/sec

learned cache
Learning from simple experts (e.g., LeCaR\textsuperscript{[1]})

**Introduction**

Learned caches

Which one to evict?

- Dark green
- Dark red

Maintain two sets of metadata is expensive and complex delayed reward

Introduction

Learned caches

which one to evict?

can only use limited number of features ➞ low efficiency upper bound
require sampling many objects to compare at each eviction ➞ low throughput

Learning from distribution (e.g., LHD[2])

**Introduction**

Learned caches

which one to evict?

- features
- features
- features
- features

Object-level learning (e.g., LRB\[^3\])

leverage more features than other learned caches

sampling and inference at each eviction => very very very slow

---

GL-Cache: a group-level learned cache
New idea

utilizes multiple features, while amortizes overheads

object groups

- features
- features
- features

score

- 100
- 8
- 20

groups accumulate more information and are easier to learn
GL-Cache architecture

- **object group**
- **full**
- **empty**
- **feature cache**

---

**sample**

- **training data**
- **training**

**update model**

**rank groups for eviction**

**model**

**inference**
Design decision

• How does GL-Cache group objects
• What does GL-Cache learn
• How does GL-Cache learn
• How does GL-Cache evict
How does GL-Cache group objects

Insertion-time-based grouping

• Why?
  • objects inserted at similar time are similar
  • simple and generally applicable
  • can be implemented on segment/log-structured storage

• But other grouping can also be supported
What does GL-Cache learn

A new utility function

- Quantify the usefulness of object groups
- Properties desired
  - smaller object -> larger utility
  - sooner-to-be-accessed -> larger utility
  - group size one -> Belady's MIN (weighted by size)
  - easy and accurate to track online

Which group is a better eviction candidate?

\[
U_o(t) = \frac{1}{T_o(t) \times s_o}
\]

\[
U_{\text{group}}(t) = \sum_{o \in \text{group}} \frac{1}{T_o(t) \times s_o}
\]

- \(T_o(t)\) time till next request since \(t\)
- \(s_o\) object size
* requires future information
How does GL-Cache learn

Features and model

- **Dynamic**
  - #requests
  - #active objects

- **Static**
  - write rate at insertion time
  - miss ratio at insertion time
  - request rate at insertion time
  - mean object size
  - age

- Model: gradient boosting tree with regression as the objective
How does GL-Cache use the model

Inference

each ranking result is used to evict a fraction of groups
pick the group with the lowest utility and the groups inserted after it

merge

evict

F_b evict evict
F_c evict
F_d evict

select based on \( \frac{1}{\text{size} \cdot \text{age}} \)

F_x
GL-Cache evaluation
Evaluation setup

• Traces
  • 103 Cloudphysics traces
  • 14 MSR traces
  • 1 Wikipedia trace

• Micro-implementation based on libCacheSim
  • LRU, CACHEUS, LHD, LRB

• Prototype implemented from Segcache
  • Cachelib (LRU), LHD, TinyLFU

• Two modes of GL-Cache:
  • GL-Cache-E, GL-Cache-T

• Metrics
  • hit ratio increase over FIFO
  • throughput relative to FIFO
Evaluation setup

• Traces
  • 103 Cloudphysics traces
  • 14 MSR traces
  • 1 Wikipedia trace

• Micro-implementation based on libCacheSim
  • LRU, CACHEUS, LHD, LRB

• Prototype implemented from Segcache
  • Cachelib (LRU), LHD, TinyLFU

• Two modes of GL-Cache:
  • GL-Cache-E, GL-Cache-T

• Metrics
  • hit ratio increase over FIFO
  • throughput relative to FIFO
Efficiency

GL-Cache-E is slightly better than state-of-the-art algorithms
GL-Cache-T is close to LRB
GL-Cache-E is faster than all state-of-the-art learned caches. GL-Cache-T is significantly faster.
Summary

Object-level learning (e.g., LRB)
Learning from distribution (e.g., LHD)
Learning from simple experts (e.g., LeCaR)

Group-level Learning (this work)

open-sourced at https://github.com/thesys-lab/fast23-GLCache
juncheny@cs.cmu.edu
https://junchengyang.com