GL-Cache

1.Introduction

一些LRU变体：[41,43,69,76,85]

结合频率和最近性：[4, 15, 26, 28, 56, 92];

频率和对象大小：[17,20]

learned caches：

- object-level learning,
- learning-from-distribution
- learning-from-simple-experts

对象级学习——LRB：利用对象特征预测下次访问的时间，并以此为依据逐出。

从数据分布学习——LHD：用寿命和大小算命中密度，淘汰密度最低的。

“learningfrom-simple-experts”——LeCaR，Cacheus

组级别学习面临的问题：

1. 如何对对象分组，有效淘汰
2. 如何衡量对象组的有用性（用来作为淘汰依据）
3. 如何在线学习并预测对象组的有用性

GL-Cache：

- 使用"write time"将对象聚类成组，基于merge的淘汰来剔除最没用的组
- 引入组实用性函数来给组排序。（能达到和对象级学习近似的淘汰效率）
- 两级淘汰：先在重量级的组级别学习识别要淘汰的组；再用轻量级对象指标从要淘汰的组中保留有用的对象。

2.背景和动机

Cache: 命中率——淘汰算法；吞吐量——资源利用

传统方法:LRU LFU的变体，基于少量特征作淘汰决定。不同的工作负载下，各个特征的重要性可能不同，甚至，同工作不同大小的Cache下，特征重要性也不同。

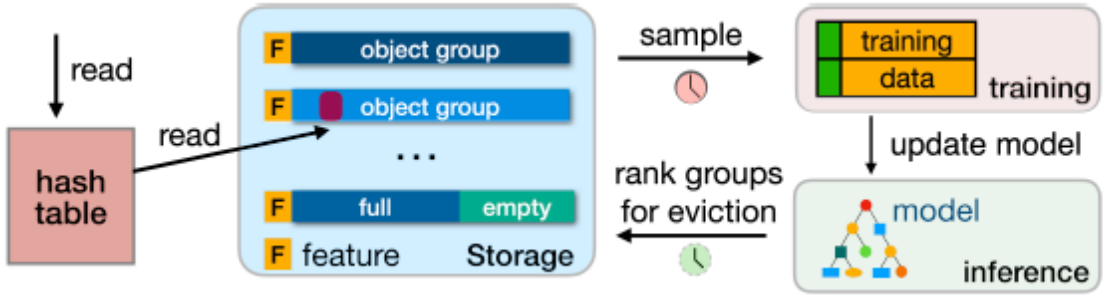
三种学习型Cache的优缺点

...

3.GL-Cache

3.1

总览：



3.2

相比于其他方法：

- 分组均摊了开销。
- 分组会积累更多信号？

  Cache遵循Zipf分布（类似28定律），多数对象很少被访问。

  以组为单位，请求更多，信息更多，易于学习和预测。

3.3 对象组

对象不能随意换组。

在进入Cache时，就应该决定好分组。根据简单的静态特征：时间，id，类型，大小等。本文主要关注写时间：

按照写时间分组后，与随机分组作对比，每组内对象的重用时间间隔更相似。以及一些其他特征也有相似性。

按写时间分组后，一些组的平均重用时间明显比其他高。（10倍以上）这些组就是很好的淘汰候选项。

根据以上两条观察，按写时间分组时可行的。

（按写时间分组也更适配日志结构来实现）

3.4 Utility of object groups

当对象的大小不均等时，找到最佳替换对象是NP-Hard，所以找到最佳组也是NP-Hard，以下是经验性的性质：

- 由较大的对象组成的组应该有较低的实用性
- 距离下次访问的时间间隔长的有较低的实用性
- 如果每组只有一个对象，应该退化为Belady. //?
- 在有限时间内的计算结果，应尽可能接近利用所有未来信息的计算结果（理想结果）。换句话说：遥远的将来才被请求，甚至不会被请求的对象的对象对实用性的贡献较少

实用性的定义：

T表示到下次访问的时间间隔，s表示大小。

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

3.5

选定了7个特征

GBM

训练

使得对象组的U值L2loss最小。

对Cache中的对象组采样，复制其特征到内存区域。当其中一个对象被访问时，用时间间隔（从采样到访问）计算U值加到该组的U值中，标记该对象（保证它只被计算一次）

//用这样的时间间隔计算的U比真实值大

//要是期间又被访问了，时间间隔不就变了吗??

一个样本组可能在被训练前就剔除了，GL—Cache保留“ghost entries”来弥补Utility计算中未考虑到的因素。对“ghost entry”的访问将更新U值。

每天从0开始，重新训练

推理

需要淘汰时，对所有的组进行预测，排序。一次排序结果用作多次淘汰，减少推理频率。

3.6 对象组的逐出

选出最没用的组后，再把它与$N_{merge}-1$个与它写入时间最相近的组合并，然后再从中保留出一部分可能仍有用的对象（基于age and size），形成新的组（这也是唯一可能的换组情况），其他的对象剔除。

注：merge中的其他N-1个组都是基于写时间相近选择的，而不是U值rank，因为必须要保证新生成的组内，对象的写入时间相近。

3.7 参数

组的大小S_{group}

合并个数N_{merge}

一次推理对应的逐出组数 $F_{eviction}$

4.Evaluation

Prototype system

基于Segcache

XGBoost库，参数默认。

GL-Cache：组大小1 MB，每次驱逐时合并五个组，驱逐
每组推理后的5%后重新推理。

Micro-implementation

基于一个Cache模拟库，做“storage-oblivious”的实现，只操作元数据。

GL-Cache-S: Sgroup = 60 objects, Nmerge = 2 groups, Feviction = 0.02.

GL-Cache-T: Sgroup = 200 objects, Nmerge = 5 groups, Feviction = 0.1

Workloads

Table 3: Three sets of 128 traces were used in the evaluation.

Dataset	# traces	# requests (millions)	Source
CloudPhysics [94]	103	2115	VM disk I/O
MSR [73]	14	410	Disk I/O
Wikimedia [87]	1	2804	CDN requests

指标:

命中率增量: $(HR_{alg} - HR_{FIFO})/HR_{FIFO}$

相对吞吐量:R_{alg}/R_{FIFO}

首先，对比了Oracle的对象淘汰实现和组淘汰实现（与GL-Cache类似）证明组淘汰不会成为效率瓶颈
//???

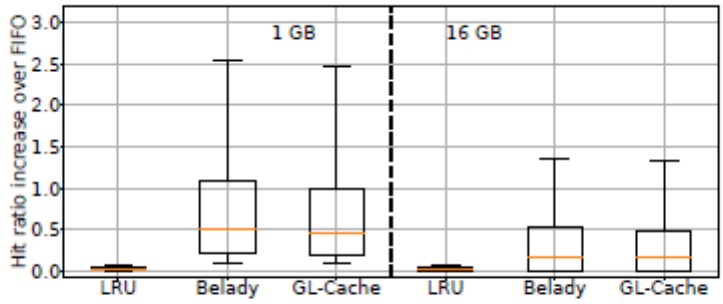
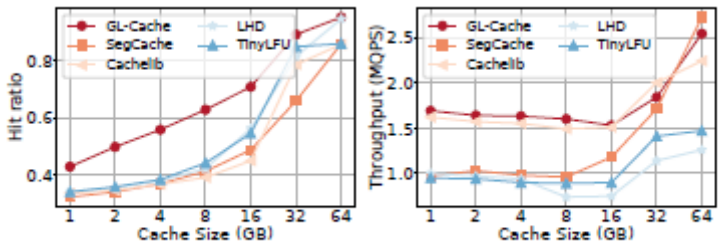


Fig. 5: With oracle assistance, group eviction can achieve a similar hit ratio improvement as object eviction.

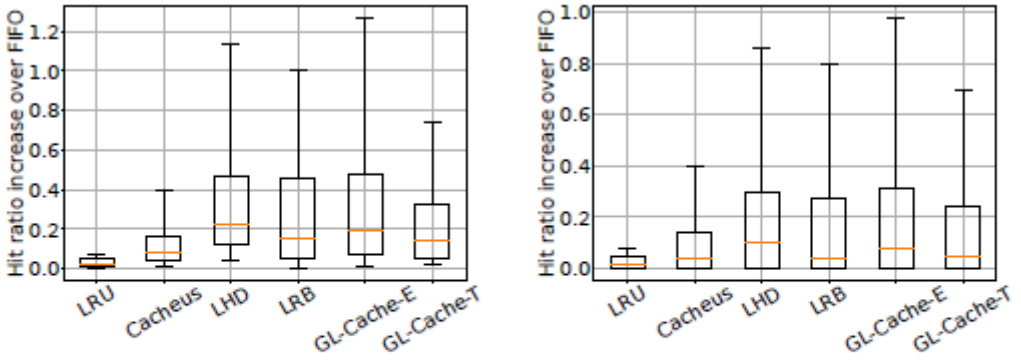
Cache效率

原型，CloudPhysics，命中率和吞吐量的对比



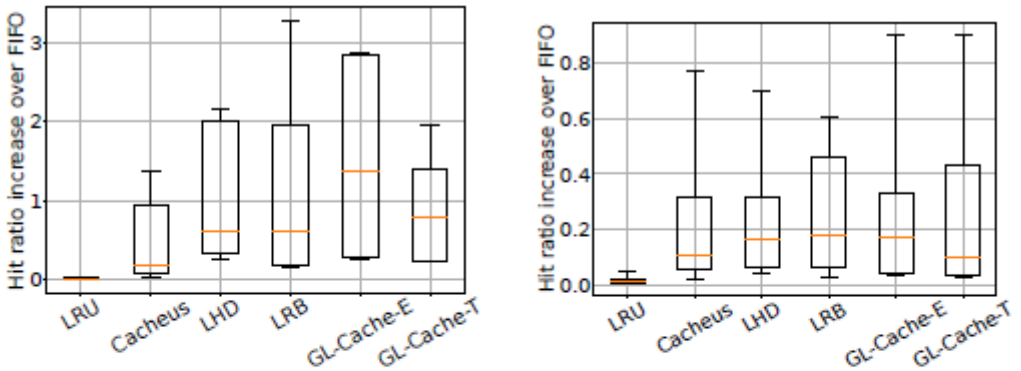
(a) Hit ratio (b) Throughput

微实现，CloudPhysics and MSR，观察相对于FIFO的命中率变化



(a) CloudPhysics, small cache size

(b) CloudPhysics, large cache size

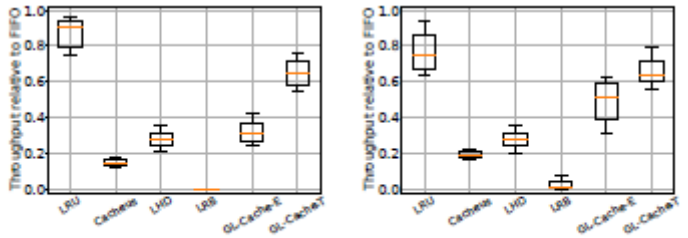


(c) MSR, small cache size

(d) MSR, large cache size

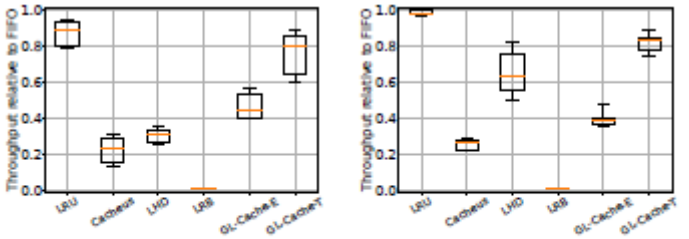
Fig. 7: Hit ratio increase over FIFO. GL-Cache runs under two modes, GL-Cache-E is the efficient mode, GL-Cache-T is the throughput mode.

微实现，相对吞吐量的对比



(a) CloudPhysics, small cache size

(b) CloudPhysics, large cache size



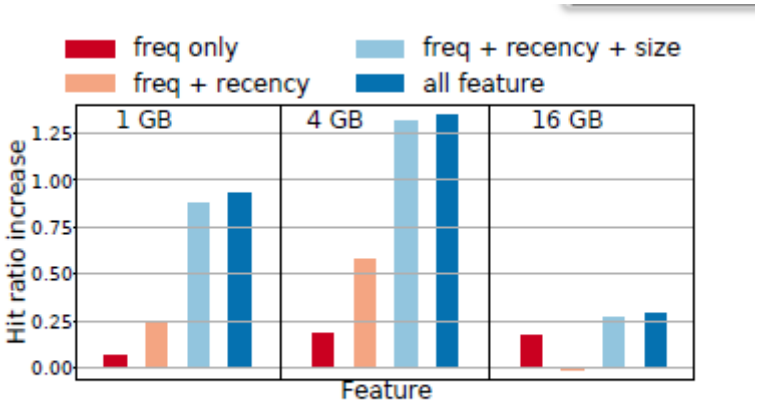
(c) MSR, small cache size

(d) MSR, large cache size

Fig. 8: Throughput relative to FIFO.

（在缓存上的机器学习也会引入一些存储开销）

分析XGBootst中特征的重要性，总体看，频率和寿命权重较高



(c) Feature utility case study.

在GL-Cache中，特性的选择和使用不仅适应工作负载
也适应不同的配置，如缓存大小。

4.6 Sensitivity analysis

分析三个参数对GL-Cache的影响情况。也讨论了训练频率，样本数

对比E,T两组参数的效果，说明GL-Cache鲁棒性较强。用户也可以自己微调做Trade-off。

Conclusion

组级学习很好地适应了工作负载和Cache大小，均摊了开销。

在小开销下做出了更好的逐出决策。

与其他学习型Cache相比，在保持高命中率的同时，显著提高了吞吐量。