计算机体系结构论文复现

摘要

本文简述了论文《iCache: An Importance-Sampling-Informed Cache for Accelerating I/O-Bound DNN Model Training》解决深度神经网络训练时从存储系统获取训练数据造成的 I/O 时延过长导致 GPU 阻塞的问题，本文提出了一种重要性采样感知缓存系统，能够显著提高神经网络的训练效率。我对该论文的部分实验进行了复现，本文简述了实验复现结果和基于实验结果得出的结论。该论文来自 HPCA 2023。

论文索引信息：

```
@inproceedings{chen2023icache,
      title={icache: An importance-sampling-informed cache for accelerating i/o-bound dnn model training},
  author={Chen, Weijian and He, Shuibing and Xu, Yaowen and Zhang, Xuechen and Yang, Siling and Hu, Shuang and Sun, Xian-He and Chen, Gang},
  booktitle={2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)},
  pages={220--232},
  year={2023},
  organization={IEEE}
}
```

1. 问题描述

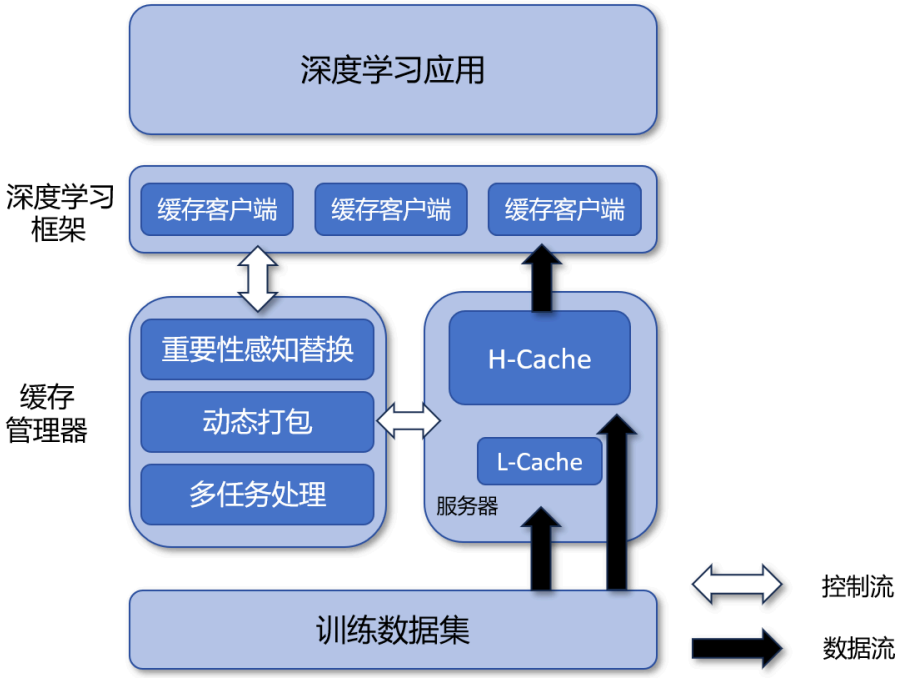
从存储系统中获取大量深度神经网络（DNN）训练数据会导致长时间的 I/O 延迟，并引发 GPU 的获取阻塞。在 DNN 训练中，重要性采样可以减少 GPU 上的数据计算量，同时保持类似的模型精度。

本文设计了一种新的重要性采样感知缓存，称为 iCache，用于加速I/O瓶颈的DNN训练任务。iCache 只获取部分样本，而不是整个数据集。缓存被分为两个区域：H-cache 和 L-cache，分别存储高重要性样本和低重要性样本。

- 根据样本的重要性管理 H-cache 中的数据项。当 L-cache 出现缓存未命中时，通过样本可替代性和动态打包技术提高缓存命中率，减少随机 I/O 次数。
- 当多个并发任务访问 H-cache 中的相同数据集时，设计了一个模型，为缓存的样本分配相对重要性值，以避免因并发训练任务缺乏协调而导致的缓存抖动。

2. 方法设计

本文设计了一种加速 DNN 模型训练的智能缓存系统，支持单节点多 GPU 训练和多节点分布式训练。单节点架构的细节如图所示，主要由客户端模块、缓存管理器和服务器组成。



iCache 客户端与服务器

客户端模块集成在深度学习框架（如 PyTorch）中，主要负责请求转发。当 DNN 应用程序的数据加载器开始随机选择样本进行读取时，客户端将通过调用 RPC 接口将请求转发至 iCache 服务器。每个客户端属于一个唯一的 DNN 训练任务，对用户是透明的。

客户端模块维护一个 H-list，用于记录训练任务的 H 样本信息，格式为 `<ID, IV>`，ID 表示样本的标识符，IV 表示样本的重要性值。IV 的计算基于重要性采样算法，本文选择了基于损失的简单高效算法进行当前设计。

现有的计算导向重要性采样（CIS）算法无法减少 I/O 时间，本文提出了面向 I/O 的重要性采样（I/O-Oriented Importance Sampling, IIS），以减少每个周期中被加载的样本数量。

- IIS 在每个周期开始前，基于样本的历史重要性值决定当前周期中要训练的样本，并仅加载和训练这些样本。

iCache 服务器的功能是提供用户级缓存，用于在内存中存储训练数据集，以加速 I/O 瓶颈的 DNN 训练。

- H-cache 存储记录在 H-list 中的 H 样本，其容量决定了能缓存的 H 样本数量。当 H-cache 容量不足以缓存所有 H 样本时，iCache 使用重要性感知缓存替换算法进行管理。
- L-cache 的设计目的是进一步减少因访问 L 样本导致的小型随机 I/O。它仅缓存 H-list 中未列出的 L 样本。另一个好处是保持较高的模型精度。

缓存管理器

用于根据样本的重要性值管理缓存数据，由于采样过程中参考的重要性值由基于历史值的重要性采样算法定期更新，因此缓存管理器会定期从客户端拉取 H-list，以在缓存性能和管理开销之间实现良好的平衡。

管理器需要动态打包样本，随后将其加载到 L-cache 中。I/O 的最小单元是一个样本包。

当多个任务访问相同数据集时，管理器需要根据来自不同任务的多个重要性值和任务的缓存效益来决定如何管理数据项。管理器采用多任务处理算法调整 H-list 中样本的重要性值，以减少所有任务的总训练时间。

3. 关键技术

重要性感知缓存算法

iCache 使用小顶堆（H-heap）管理 H-cache 中的样本，堆对象按重要性值排序，堆顶的样本（top-node）是重要性值最低的。H-heap 的空间开销较小，通常不超过 H-cache 容量的 0.5%。

iCache 的替换策略：当缓存未满时，直接将从存储系统加载的 H 样本插入 H-cache，并更新 H-heap；如果缓存已满，则比较新样本和 top-node 的重要性值。如果新样本更重要，替换 top-node；否则忽略新样本。对 L 样本，优先从 L-cache 中读取；若未命中，则从 L-cache 中随机选取其他样本替代，减少随机 I/O。

关键点：相比传统的 LRU 替换算法，该方法优先缓存高重要性样本，提高缓存命中率并减少 I/O 瓶颈对训练的影响。

算法伪码表示如下：

```TEXT
算法 1：重要性感知缓存算法
输入：
- bs：批量大小；
- batch_id：由 IIS 生成的一批样本 ID 的序列；
- H-list：服务器当前的 H 样本列表，从客户端拉取；

batch_data ← []
 for i ← 0 to bs-1 do
     id ← batch_id[i]
     if id ∈ H-list then // 样本是 H 样本
         if id ∈ H-cache then
             data ← 从 H-cache 读取(id)
         else
             data ← 从存储系统读取(id)
             if H-cache 未满 then
                插入样本到 H-cache(id, data)
            else
                iv_cur ← 获取样本 id 的重要性值
                iv_min ← 获取 H-heap 中的最小重要性值
                if iv_cur > iv_min then
```

```
                    从 H-cache 中删除并插入(id, data)
                end if
            end if
        end if
    else // 样本是 L 样本
        if id ∈ L-cache then
            data ← 从 L-cache 读取(id)
        else // 使用可替代性替换
            data ← 随机从 L-cache 获取数据
        end if
    end if
    batch_data.append(data)
end for
返回 batch_data
```

算法流程：

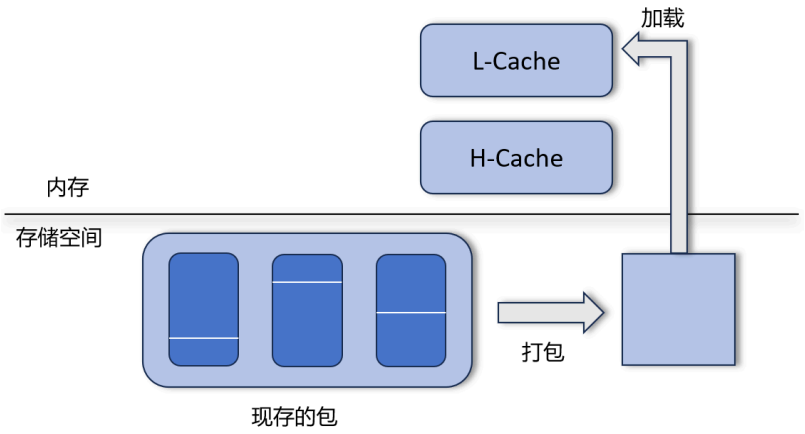
1. 遍历每个样本的 ID，检查它是否是 H 样本。
2. 如果是 H 样本，先尝试从 H-cache 中读取；若未命中，则从存储系统加载，并根据重要性值决定是否插入到 H-cache 中。如果 H-cache 已满，则替换重要性值最小的样本。
3. 如果是 L 样本，尝试从 L-cache 中读取；若未命中，利用可替代性，从 L-cache 中随机选择其他样本代替。

动态打包

iCache 使用两个并发线程（打包线程和加载线程）动态实现样本打包和加载：

- 打包线程：初始时，存储系统无预打包样本，打包线程随机选取 L 样本打包成大文件并存储。
- 加载线程：加载线程选取数据包加载到 L-cache。当请求样本不在 H-list 且未命中 L-cache 时，随机选择未访问的 L 样本替换。当 L-cache 数据全被访问时，加载新数据包。

重组机制：训练中更新 H-list 和 L 样本比例，定期对未命中样本重新打包以提升多样性，同时用随机 L 样本填充剩余空间。重组后数据包加载到内存并存入 L-cache。



4. 实验分析与评估

实验环境

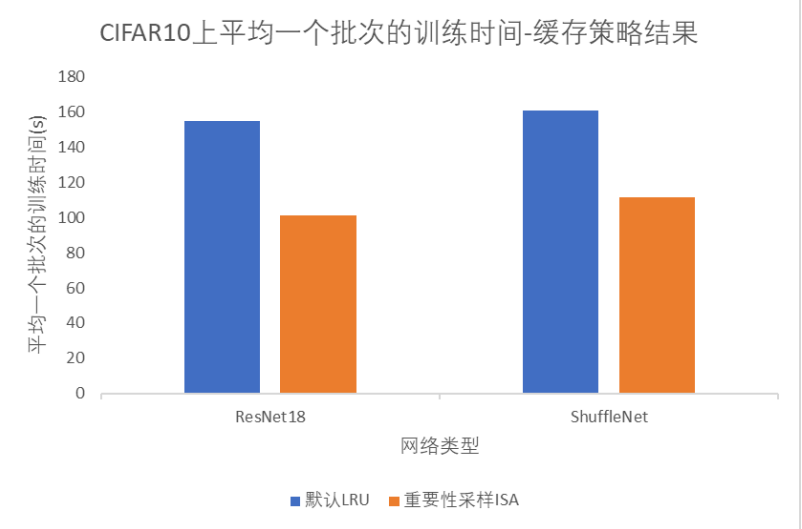
机器：装有 Ubuntu 22.04 系统的联想 y9000p 笔记本电脑，搭载 32 GB 内存和 RTX 4070 for LapTop 8 GB 显卡。

软件与工具：

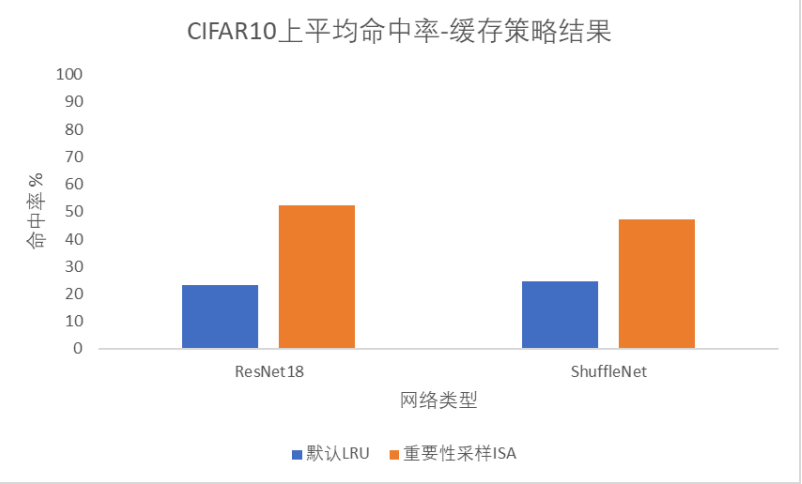
- Python 3.9
- PyTorch 1.8.0+cu111
- torchvision 0.9.0+cu111
- six
- grpcio 1.46.1
- grpcio-tools 1.46.1
- requests 2.27.1
- Golang 1.18.2
- etcd 3.4.4

实验结果

在 CIFAR 10 上测试 ResNet 18 和 ShuffleNet 的训练，缓存策略分别使用了默认的 LRU 和 iCache 重要性采样，设置总的训练轮数为 10，每一轮的批次大小为 32，高速缓存比率设为 0.2。得到结果：在默认 LRU 缓存策略下，ResNet 和 ShuffleNet 单个批次训练时间分别是 154.77 s 和 160.65 s；在重要性采样 ISA 缓存策略下，ResNet 和 ShuffleNet 单个批次训练时间分别是 101.23 s 和 111.64 s。在重要性采样 ISA 策略下，网络的训练时间有较为显著的提升。



同样的配置下得到两个网络在两个缓存策略下的缓存命中率，结果如下：在默认 LRU 缓存策略下，ResNet 和 ShuffleNet 的平均命中率分别为 23.25% 和 24.57%；在重要性采样 ISA 缓存策略下，ResNet 和 ShuffleNet 的平均命中率分别为52.32% 和 47.28%。



5. 结论

通过以上实验结果，可以得出以下结论：

1. 在神经网络的训练过程中，iCache 的重要性采样策略比传统的 LRU 缓存策略性能更优，能够消除一部分由于频繁 I/O 导致的性能瓶颈，提高 GPU 利用率，显著加速神经网络的训练过程；
2. iCache 的重要性采样策略有效的一个重要因素是在重要性采样的技术下，系统在高速缓存中的命中率得到了显著的提高。因此在存在大量数据样本需要读取的情况下，更高的高速缓存命中率能够减少大量的 I/O 次数，避免系统在慢速的存储系统中消耗过多的时间，从而导致 GPU 阻塞。

6. 相关工作

在深度学习训练的缓存系统设计上，AV Kumar 等人[2]的工作提出了一种智能存储缓存系统 Quiver，它通过深度结合深度学习框架，显著提升了深度学习工作负载的吞吐量。Quiver 的设计基于深度学习工作负载的几个关键特性，包括数据的可替代性、高预测性和跨作业的共享性，从而优化缓存效率。其核心创新包括基于内容的缓存索引以支持跨用户数据共享、可替代缓存命中的策略以避免缓存抖动，以及通过动态测量作业对缓存的敏感性来实现利益感知的缓存分配，在本文的实验中也引用了 Quiver 的设计，测试了以其作为缓存策略时的训练时长、命中率等指标。

M Dantas 等人[3]提出了 Monarch，一种面向深度学习框架的通用中间件，旨在通过分层存储管理优化高性能计算（HPC）环境中的深度学习训练性能。深度学习工作负载通常依赖共享的并行文件系统（PFS）存储和访问训练数据，但在多任务并行运行时，PFS容易因大量I/O请求而成为性能瓶颈。MONARCH通过利用HPC计算节点上的本地存储资源（如SSD）来缓解PFS的I/O压力，从而显著降低数据加载时间，提高训练效率。

参考文献

[1] Chen W, He S, Xu Y, et al. icache: An importance-sampling-informed cache for accelerating i/o-bound dnn model training[C]//2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2023: 220-232.

[2] Kumar A V, Sivathanu M. Quiver: An informed storage cache for deep learning[C]//18th USENIX Conference on File and Storage Technologies (FAST 20). 2020: 283-296.

[3] Dantas M, Leitao D, Correia C, et al. Monarch: Hierarchical storage management for deep learning frameworks[C]//2021 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2021: 657-663.

附录

复现的论文：Chen W, He S, Xu Y, et al. icache: An importance-sampling-informed cache for accelerating i/o-bound dnn model training[C]//2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2023: 220-232.

原始开源代码的链接：https://github.com/ISCS-ZJU/iCache

复现 Github 仓库：https://github.com/Northern114/iCache_Reproduction