计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202330402
x(x): xx−xx, xxxx

基于 GPU 直访存储架构的推荐模型预估系统

谢旻晖　陆游游　冯杨洋　舒继武
（清华大学计算机科学与技术系　北京　100084）
（xmh19@mails.tsinghua.edu.cn）

A Recommendation Model Inference System with GPU Direct Storage Access

Xie Minhui, Lu Youyou, Feng Yangyang, and Shu Jiwu
（*Department of Computer Science and Technology*, *Tsinghua University*, *Beijing* 100084）

Abstract　Emerging deep learning recommendation models (DLRM) have been widely used in modern recommendation systems. The unique embedding layer of DLRM, commonly with tens of trillions of parameters, induces massive irregular access to storage resources, which becomes the performance bottleneck of model inference. Existing inference systems rely on CPU access to embedding parameters on DRAM and SSD. However, we find that this architecture suffers from excessive CPU-GPU communication overhead and redundant memory copies, resulting in increased latency of embedding layers and limited inference performance. In this paper, we propose GDRec, a recommendation model inference system based on the architecture of GPU direct storage access. The core idea of GDRec is to eliminate the CPU from the access path of embedding parameters and let the GPU directly access storage resources with the paradigm of zero copy. For direct access to DRAM, GDRec retrofits the unified virtual addressing feature of CUDA, to allow GPU kernels to issue fine-grained access to host DRAM. GDRec further introduces two optimizations, access coalescing and access aligning, to fully unleash the performance of DRAM access. For direct access to SSD, GDRec implements a lightweight NVMe driver on GPU, allowing GPU to submit I/O commands to read data from SSD to GPU memory directly, without extra copies on DRAM. GDRec also leverages the massive parallelism of GPU to shorten the submission time of I/O commands. Experiments on three public datasets show that GDRec can improve inference throughput by 1.9x, compared to a highly-optimized recommendation model inference system, NVIDIA HugeCTR.

Key words　GPU direct storage access; parameter store; recommendation system; inference system; storage system

摘　要　新型深度学习推荐模型已广泛应用至现代推荐系统，其独有的特征——包含万亿嵌入参数的嵌入层，带来的大量不规则稀疏访问已成为模型预估的性能瓶颈. 然而，现有的推荐模型预估系统依赖 CPU 对内存、外存等存储资源上的嵌入参数进行访问，存在着 CPU-GPU 通信开销大和额外的内存拷贝 2 个问题，这增加了嵌入层的访存延迟，进而损害模型预估的性能. 提出了一种基于 GPU 直访存储架构的推荐模型预估系统 GDRec.GDRec 的核心思想是在嵌入参数的访问路径上移除 CPU 参与，由 GPU 通过零拷贝的方式高效直访内外存资源. 对于内存直访，GDRec 利用统一计算设备架构（compute unified device architecture, CUDA）提供的统一虚拟地址特性，实现 GPU 核心函数（kernel）对主机内存的细粒度访问，并引入访问合并与访问对齐 2 个机制充分优化访存性能；对于外存直访，GDRec 实现了一个轻量的固态硬盘（solid state disk, SSD）驱动程序，允许 GPU 从 SSD 中直接读取数据至显存，避免内存上的额外拷贝，GDRec 还

收稿日期：；**修回日期**：
基金项目：国家自然科学基金优秀青年科学基金项目(62022051).
This work was supported by the National Natural Science Foundation of China for Excellent Young Scientists (62022051).
通信作者：陆游游 (luyouyou@tsinghua.edu.cn).

利用 GPU 的并行性缩短提交 I/O 请求的时间. 在 3 个点击率预估数据集上的实验表明, GDRec 在性能上优于高度优化后的基于 CPU 访存架构的系统 (NVIDIA HugeCTR), 可以提升多达 1.9 倍的吞吐量.

关键词 GPU 直访存储; 参数存储; 推荐系统; 预估系统; 存储系统

中图法分类号 TP302.1

我们正身处一个信息过载的时代, 全球内容生产指数型增长: 新浪微博每天产生 1.17 亿条微博[1], 快手每天有千万条新发布的短视频[2]. 在如此庞大的规模下, 用户很难从中找到自己可能感兴趣的内容. 推荐系统负责从中筛选出可能感兴趣的内容个性化地推荐给每个用户, 被认为是解决信息过载的有效手段. 近年来, 随着深度学习的快速发展, 新型深度学习模型[3-6]已广泛应用至现代推荐系统之中, 显著提升了个性化推荐的准确率. 根据 Meta 公司的统计[3], 推荐模型预估(inference)耗时在其数据中心的人工智能服务总耗时占比高达 80%. 因此, 推荐模型预估系统设计已成为数据中心人工智能系统设计中的一个关键问题.

与计算机视觉、自然语言处理中的计算密集型深度学习模型不同, 推荐模型中包含 1 个巨大且访存密集的嵌入层(embedding layer). 该嵌入层用于映射对应输入的特征 ID 至低维嵌入参数(embedding vector), 由于推荐模型接受的输入包含海量高维稀疏的特征 ID(例如用户 ID、近期观看过的视频 ID), 而深度神经网络无法直接从高维稀疏特征中学习. 所有低维嵌入参数拼接后, 一同输入深度神经网络拟合样本标签. 由于特征交叉技术[5]的广泛使用与每个特征本身的海量 ID 空间, 嵌入层中包含的参数可多达千亿级, 占模型总体参数的 99.99% 以上[7]. 受限于 GPU 的内存大小, 嵌入参数通常存储在内存(dynamic RAM, DRAM)与固态硬盘(solid state disk, SSD)等较为廉价的硬件资源上[8-10].

现有推荐模型预估系统[11]依赖于 CPU 对存储资源上的嵌入参数进行访问(后文称为基于 CPU 访问存储架构). 具体地, 由 CPU 接收经 GPU 去重后的特征 ID, 访问相应 DRAM 或 SSD 获得嵌入参数后, 再将所需嵌入参数额外聚集(gather)到 1 块连续的内存空间, 通过直接存储器访问(direct memory access, DMA)引擎传输至 GPU.GPU 收到嵌入参数后, 完成后续的神经网络计算任务.

然而, 我们发现该架构存在如下 2 个问题: 一方面, GPU 所需的任何数据均需交由 CPU 访问再行传回, CPU-GPU 频繁交互带来的通信开销增加了参数访问的时延; 另一方面, 聚集操作引入的额外内存拷贝消耗了 CPU 资源与内存带宽, 为访存密集的嵌入层带来了严重的性能损失.

针对上述问题, 本文提出一种基于 GPU 直访存储架构的推荐模型预估系统 GDRec.GDRec 的核心思想是在嵌入参数的访问路径上移除 CPU 参与, 由 GPU 通过零拷贝的方式直接访问 DRAM, SSD 等存储资源, 以减少主机端 CPU 与 DRAM 的开销. 具体地, GDRec 包含 2 个直访存储机制: 内存直访机制(用于访问位于 DRAM 上的参数)与外存直访机制(用于访问位于 SSD 上的参数).

内存直访机制利用设备商提供的统一虚拟地址(unified virtual address, UVA)特性[12], 将嵌入参数所在的 DRAM 内存区域映射至 GPU 内存虚拟地址空间. 由此, GPU 核心函数(kernel)可以绕过主机端 CPU, 直接通过 load/store 指令以字节粒度直接访问 DRAM. 由于跨 PCIe 总线访问主机 DRAM 内存有一定开销, GDRec 进一步地提出访问合并与访问对齐 2 个优化, 通过调度 GPU 线程读取嵌入参数的方式, 尽可能减少 GPU 发出的 PCIe 事务数量.

外存直访机制在 GPU 上实现了一个类似于存储性能开发套件(storage performance development kit, SPDK)[13]的用户态 NVMe(non-volatile memory express)驱动程序, 允许 GPU 线程直接向 SSD 提交读写请求, 同时也允许 SSD 通过 DMA 引擎将读取到的硬盘数据直接写入 GPU 内存. 整个过程无需主机端的 CPU 与内存参与.

在每次推荐模型的预估过程中, GDRec 首先利用 GPU 高效地将特征 ID 去重, 然后根据对应的嵌入参数所在的位置将请求分流至内存直访机制与外存直访机制. 待所有嵌入参数的读取请求完成后, 类似于传统的预估系统, GDRec 进行后续的神经网络计算.

我们在 3 个公开的点击率预估数据集上进行了测试, 实验表明: 与 NVIDIA 公司深度优化的推荐模型预估系统 HugeCTR[11](HugeCTR-Inference)相比, GDRec 可以提升超过 1 个数量级的吞吐量; 相比于使用 SPDK 高度优化后的 HugeCTR, GDRec 仍有着

多达 1.9 倍的吞吐量提升.

综上所述，本文的主要贡献有 3 个方面：

1）分析了现有推荐模型预估系统（基于 CPU 访问存储架构）的性能问题；

2）提出了一种基于 GPU 直访存储架构的推荐模型预估系统 GDRec，其包含内存直访与外存直访 2 个机制，分别使能 GPU 直访 DRAM 与 SSD 上的参数；

3）使用实验说明 GDRec 设计的有效性. 特别地，在 3 个公开的点击率预估数据集上，GDRec 在性能上优于高度优化后的基于 CPU 访问存储的同类系统，吞吐提升多达 1.9 倍.

1 背景介绍与研究动机

在本节中，主要介绍深度学习推荐模型的基本结构与现有预估系统的主要处理流程，同时分析现有系统使用 CPU 访问存储架构导致的性能问题.

1.1 深度学习推荐模型结构

自微软公司的 Deep Cross[6] 与谷歌公司的 Wide&Deep 模型[5] 发布以来，推荐模型已全面步入深度学习时代. 与传统机器学习模型相比，深度学习推荐模型的表达能力更强，可以挖掘出用户（user）与物品（item）之间更深层次的联系，从而更好地推测用户的喜好.

深度学习推荐模型的输入特征主要包含 2 种类型：稠密特征和稀疏特征. 稠密特征亦称数值型特征、连续特征，指具有数值意义的特征，如用户的年龄、商品被访问的次数等；稀疏特征亦称类别型特征、ID 型特征，指如用户 ID、视频 ID 列表等以独热（one-hot）或多热（multi-hot）形式存在的高维特征. 稀疏特征一般被编码为 ID 的列表，如某用户近期观看过的视频为 [视频 ID_1, 视频 ID_4, 视频 ID_5].

图 1 展示了典型的深度学习推荐模型结构，它由 2 个部分组成：嵌入层（embedding）与多层感知机（multi-layer perceptrons, MLP）. 如前文所述，由于深度神经网络无法直接从高维稀疏特征中学习[14]，嵌入层用于将稀疏特征映射至稠密的嵌入参数. 具体地，嵌入层由多个嵌入表（embedding table）组成，如用户、视频嵌入表. 每个表用于映射对应的输入特征 ID 至低维嵌入参数，嵌入参数的维度称为嵌入维度（embedding dimension），映射的过程类似于哈希表查询. 在嵌入层映射完稀疏特征后，所有嵌入参数会与样本中的稠密特征拼接，得到包含样本所有特征的特征向量. 模型将特征向量输入多层感知机，多层感知机对特征向量各个维度进行复杂的交叉组合，最后拟合样本的标签（例如某用户是否可能观看某视频）. 典型的工业级推荐模型神经网络包括：Wide&Deep[5], DeepFM[15], DCN[16], DIN[17], DIEN[18]. 各模型在具体细节上有所不同，但模型主体结构都符合 embedding-MLP 2 个部分的结构.

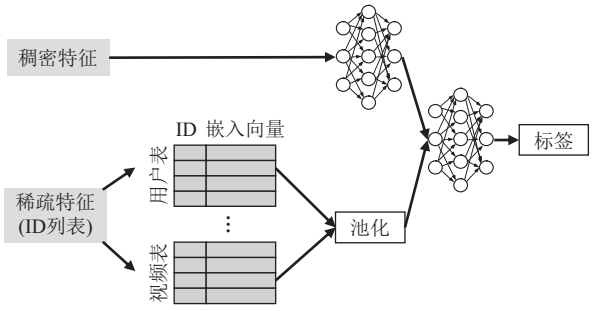

Fig. 1 Structure of deep learning recommendation model

图 1 深度学习推荐模型结构

从图 1 中我们还可以看到，深度学习推荐模型的参数可分为 2 类：稀疏参数（由嵌入层组成，用于处理稀疏特征）与稠密参数（由 MLP 层组成，用于处理稠密特征与稀疏特征经嵌入表映射后的中间结果）.2 类参数在计算模式与存储特征上有着本质的区别. 对于计算模式，只有样本包含的特征 ID 涉及到的一部分稀疏参数会参与计算，而全部的稠密参数都会参与计算；对于存储特征，稠密参数通常只有百兆级，而稀疏参数可多达千亿级，这一方面是由于每个稀疏特征 ID 空间巨大，另外一方面也由于广泛使用的特征交叉技术[5] 可以将多个特征做笛卡儿积（Cartesian product），从而形成 ID 空间更大的新特征.

许多公司报告（如阿里巴巴[19]、Meta[3]），在他们生产环境的模型中，嵌入层通常占据模型端到端预估的时延 60% 以上. 其原因在于嵌入层引入了大量对存储资源的随机访问. 因此，针对嵌入层的系统设计与优化是提升推荐模型预估系统性能的关键所在.

1.2 传统架构以及其问题分析

诸多文献[2,7,20] 表明，更大的参数量往往能带来更好的推荐质量，在过去几年中，推荐模型参数量级已从百亿级迅速膨胀至千亿级，海量参数使得纯内存存储方案成本过高. 另一方面，随着 Optane 等高速存储介质的发展，现代高性能 SSD 的平均延迟正不断降低. 例如，Intel 公司在 2018 年推出的 905P SSD 的 4 KB 随机读延迟仅为 10 μs.为了提高存储性价比，学术界与工业界的许多工作[7-8,10,21-23] 都已迁移至内存外存混合存储方案. 针对 1.1 节中所述推荐模型结构与稠密、稀疏参数在计算、存储上的特点，现有推荐

系统比较通用的做法是将稠密参数缓存在 GPU 的内存中, 而将稀疏参数存储在 DRAM, SSD 等相对廉价存储介质中.

如图 2, 现有系统采用基于 CPU 访问存储架构. 一次典型的预估处理流程如下:

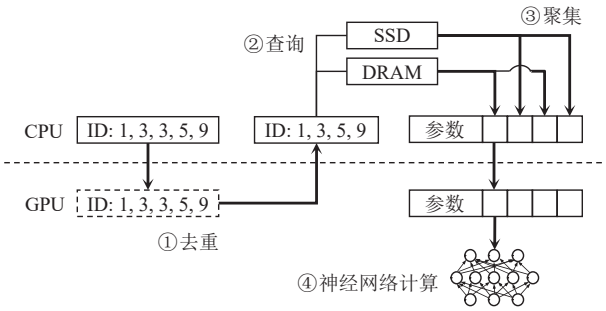

Fig. 2 Typical inference process

图 2 典型预估流程

①去重. 一次预估输入往往包含一批样本, 其中包含大量重复的特征 ID[11].CPU 会首先将所有特征 ID 传输给 GPU, 由 GPU 将 ID 去重. 得益于 GPU 的高并行性, GPU 去重可以获得比 CPU 去重更好的性能.

②查询.GPU 将去重后的特征 ID 传输回 CPU, CPU 根据去重后的特征 ID, 查取 DRAM 以及 SSD 上的嵌入表, 得到 ID 对应的嵌入参数.

③聚集.CPU 将上一步查询得到的嵌入参数聚集(gather)至 DRAM 内存上一块连续的区域, 再传输给 GPU.

④神经网络计算.GPU 将嵌入参数与样本中的稠密特征拼接, 输入 MLP 计算, 得到最后的预估结果.

然而, 我们发现该架构存在着如下 2 个问题:

1) CPU 与 GPU 之间的通信开销大. 如图 2 所示, GPU 需要将访问相关的元数据(即去重后的 ID)传输至 CPU, CPU 访问存储资源得到数据后, 再传输回 GPU.2 次通信交互均涉及到 GPU 上 DMA 引擎的启动、同步, 包含数次用户态-内核态切换, 这些开销增加了嵌入层参数访问的时延.

2) 聚集操作引入的额外拷贝, 消耗了 CPU 资源与内存带宽, 同时增加了嵌入层的时延. 为了便于 GPU 上的 DMA 引擎拷贝, CPU 需将访问得到的嵌入参数额外聚集(gather)至 1 块连续的内存空间, 再由 DMA 引擎传输给 GPU.这一步额外的拷贝带来了 3 方面弊端. 首先, 浪费宝贵的 CPU 资源, 除处理嵌入层外, CPU 还需服务于网络栈、模型预处理等过程.CPU 访问存储架构浪费了一部分 CPU 资源用于访问存储、嵌入参数聚集. 在面对包含复杂预处理的模型或单机多 GPU 卡等场景下, CPU 可能成为瓶颈而影响全局性能[24]. 其次, 这消耗了内存带宽, 给访存密集的嵌入层带来了严重的性能损失. 最后, CPU 单线程逐参数拷贝性能低下, 这增加了端到端预估时延.

2 GDRec 架构

本文提出一种基于 GPU 直访存储架构的推荐模型预估系统 GDRec. GDRec 同时使用 GPU 显存、DRAM 内存与 SSD 外存 3 个层级的存储资源, 参数存储方式与现有系统类似: GPU 显存中存储稠密参数, DRAM 与 SSD 分层存储稀疏参数. 具体地, 在加载模型时, 类似于现有工作[25], GDRec 根据参数访问热度对稀疏参数进行降序排序, 并对参数 ID 重编号.GDRec 将最热的一部分参数缓存于 DRAM, 剩余部分存储于 SSD.

图 3 描述了 GDRec 的总体结构与一次预估过程.CPU 上的特征 ID 会首先传输至 GPU 进行去重; GPU 根据去重后的特征 ID, 分别通过内存直访机制(3.1 节)与外存直访机制(3.2 节), 以零拷贝的方式直接访问主机端 DRAM 和 SSD 上对应的嵌入参数; 参数聚集操作直接在 GPU 上完成; 最后, 模型正常进行神经网络计算.

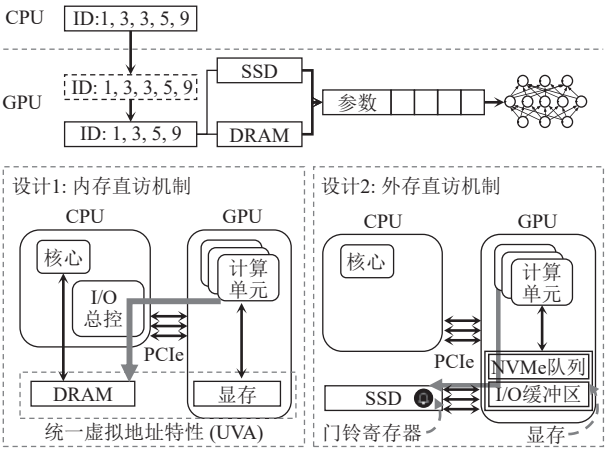

Fig. 3 Architecture of GDRec

图 3 GDRec 的总体架构

相比于传统 CPU 访存的架构, GDRec 这种基于 GPU 直访存储的架构带来了如下 3 点优势: 1)GPU 访存可以享受 GPU 高并行性带来的高性能, GPU 在并行访问 DRAM、并行提交 SSD 请求等方面可以获得比 CPU 更好的性能; 2)在访存时仅涉及 1 次 CPU 与 GPU 之间的通信, 即去重前的 ID 从 CPU 传入 GPU,

之后所有过程均由 GPU 单独完成；3）GPU 直访机制节省了聚集过程中的额外拷贝，参数可直接被读取到位于 GPU 内存的目的地址，无需经由 DRAM 缓存，这节省了 CPU 资源、DRAM 内存带宽，同时降低了端到端延迟.

需要说明的是，GDRec 的实现基于 NVIDIA 公司推出的 GPU，同时本文主要关注于 CUDA 编程模型；对于其他厂商的 GPU 以及编程模型（如 AMD 与 ROCm）的支持将是我们的未来工作.

3 关键技术设计与实现

在本节中，我们将逐一介绍 GDRec 的 2 个关键技术，包括内存直访机制与外存直访机制.

3.1 内存直访机制

根据 1.2 节的问题分析，GDRec 的内存直访机制核心思想是让 GPU 以零拷贝的方式从 DRAM 中读取数据，避免已有系统中 CPU 额外的聚集操作与 DMA 引擎启动的开销. 为了达到这个目标，GDRec 的内存直访机制利用了 CUDA 提供的统一虚拟地址（UVA）特性[12]. 该特性可以将 1 块 DRAM 内存地址空间映射至 GPU 内存虚拟地址空间，GPU kernel 以读写指令（load/store）直接对 DRAM 进行访问. 整个过程无需主机端 CPU 的参与.

GDRec 精心挑选了内存直访机制的实现方式.CUDA 提供了 2 种方式使 1 段内存可供 GPU 直访.1）如果内存尚未分配，可以通过 cudaMallocHost 函数同时实现内存的分配与地址空间的映射，该函数返回的指针可同时被 CPU 与 GPU 2 种处理器读写；2）如果内存已经分配，使用 cudaHostRegister 首先将该段内存区域锁页，再使用 cudaGetDevicePointer 获得该区域在 GPU 虚拟地址空间的指针.GDRec 使用后者，原因是我们发现其可以使用大页（huge page）分配内存，而前者不可以. 大页的使用可以减少转换检测缓冲区（translation lookaside buffer, TLB）缺失，从而提升嵌入层访存时的性能. 在系统初始化分配 DRAM 上的嵌入表后，GDRec 将对应的 UVA 虚拟地址指针传入 GPU.

内存直访机制使得 GPU 细粒度地访问 DRAM 上的稀疏参数成为可能，但细粒度的访问同时也可能带来 PCIe 开销大的问题. 一般来说，对于传输同样大小的数据，越多的 PCIe 事务意味着传输所需的元数据也越多，相应地访存的有效吞吐量也会降低. 例如，对于 PCIe 4.0 标准，每个事务层数据包（transaction layer packet, TLP）包含 12 B 或 16 B 的包头，如果数据包仅携带 32 B 有效数据，直访内存机制将遭受 27%~36% 的 PCIe 开销. 由于 PCIe 带宽是有限的，这导致 GPU 直访内存的有效带宽受限. 为了解决上述问题，GDRec 通过调整 GPU 线程读取嵌入参数的方式，尽可能地减少 GPU 发出的 PCIe 数据包数量，具体包括 2 个部分：访问合并与访问对齐. 对于访问合并，现代 GPU 支持以最大 128 B 的内存事务访问主机端的内存.GDRec 令 GPU 线程束（warp，32 个线程为 1 束）读取连续的内存区域，每个线程读取 4 B.这样，当该内存区域首地址是 128 B 对齐时，这个线程束的内存访问将被硬件合并为 1 个 PCIe 事务. 对于访问对齐，其处理的情况是当访问的参数首地址非 128 B 对齐时，例如参数尺寸非 128 B 的整数倍. 一种简单的方法是位移参数，将每个嵌入参数的首地址对齐至 128 B 的倍数，但这会引入额外的内存空间开销. 相对地，GDRec 没有改变参数在内存中的布局，而是调整了 GPU 线程读取嵌入参数的方式. 具体而言，如图 4 所示，当访问的内存区域非 128 B 对齐时，GDRec 所有线程束向高地址方向读取对齐的 1 个 128 B，并额外分配 1 个 warp 读取头部剩余的字节. 这样，假设要访问的内存区域大小为 X（单位为 B），我们只需发出 $\lceil X/(128\text{B})\rceil$ 个 PCIe 事务，这已达到最优情况.

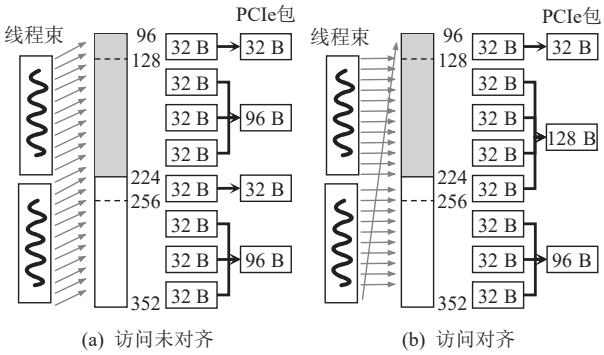

Fig. 4 Aligned access

图 4 访问对齐

3.2 外存直访机制

现有推荐模型预估系统在访问外存时并不高效，他们将嵌入参数视为键值，并依赖于已有的 SSD 键值存储系统. 例如 NVIDIA 公司的 HugeCTR 使用 RocksDB 系统[26] 读写 SSD.这种做法不仅面临着大量内核存储 I/O 栈的开销，其以 CPU 为中心的访存架构同样存在着如 1.2 节中所述的 CPU-GPU 通信开销大、额外拷贝引起的内存带宽消耗与 CPU 资源浪费

2 个问题.

对此, GDRec 提供了 GPU 上的 NVMe 驱动以实现 GPU 外存直访机制, 进而优化推荐模型预估系统的外存访问路径.GDRec 的外存直访机制具有如下 3 个特点: 1)纯用户态, 类似于 SPDK 绕过 Linux 内核 I/O 栈, 允许在用户态直接访问 SSD; 2)GPU 直访, GDRec 使得 GPU kernel 可以直接向 SSD 发起读取请求, 读到的数据直接写入 GPU 内存, 控制路径与数据路径皆无需途径 CPU, 这减少了 CPU-GPU 通信开销、主机端 CPU 和 DRAM 的使用; 3)GPU 并行快速提交 I/O 请求, GDRec 充分利用 GPU 并行性, 允许大量线程并行提交一次预估过程中所需参数对应的 I/O 请求至 NVMe 提交队列(submission queue, SQ), 以进一步缩短控制路径上的时延.

我们在 GPU 内存上分配 NVMe SSD 的提交队列、完成队列等空间, 并将这些队列注册给 SSD, 同时分配可供 SSD DMA 传输参数的数据缓存区, 基于 GPUDirect RDMA 机制, 这些空间可以被暴露给 SSD 读写访问; 然后将 SSD 的门铃寄存器(doorbell register), 包括提交门铃寄存器与完成门铃寄存器)映射至 GPU 内存虚拟地址空间, 此操作需要 2 步: 首先使用 mmap 将 SSD 的门铃寄存器所在地址映射至用户进程地址空间, 再使用 cudaHostRegister 将其进一步注册至 GPU 内存虚拟地址空间.

GDRec 的每次预估过程涉及一批位于 SSD 上的嵌入参数的查询(假设有 N 个), 直访外存的具体流程如下. 首先, GPU 接受到一批已经过去重后的特征 ID, 计算出对应的嵌入参数所在的逻辑块号; 其次, GDRec 为每个计算单元(SM)分配了单独的 SSD 请求提交队列与完成队列, 并充分利用 GPU 并行性将 I/O 命令批量写入提交队列; 同一批次的读请求中如果有多个参数在同一个逻辑块中, GDRec 会将对应的 I/O 命令合并以减少 SSD 读取的数据量; 然后, GDRec 单独分配 1 个线程用于更新提交门铃寄存器以通知 SSD, 利用门铃批处理机制(doorbell batching), 提交读请求时 GPU 产生的跨 PCIe 总线写次数可以从 N 次降低至 1 次; SSD 在收到提交门铃寄存器的更新后, 拉取位于 GPU 内存上的提交队列, 处理其中的 NVMe 命令; SSD 在完成 I/O 命令后, 会向位于 GPU 内存上的完成队列写入完成项, 并更新完成门铃寄存器.GPU 通过轮询完成队列得知 SSD 读取的完成情况.

GDRec 还采用如下 2 个优化进一步提升外存直访性能:

一方面, 由于嵌入参数的尺寸通常在 128~512 B, GDRec 通过 NVMe format 命令调整 SSD 的逻辑块大小, 其优势在于对于大小恰为逻辑块的嵌入参数, GDRec 可以将目的地址直接填入 NVMe 读命令, 并加入提交队列, 以减少 1 次从 I/O 缓存区到目的地址的拷贝, 并实现零拷贝.

另一方面, GDRec 尽可能地将所有嵌入参数均匀地分布在 SSD 整个空间上(如图 5(b)所示). 相比于简单的连续存放(如图 5(a)所示), 这种方式可以更充分地综合利用 SSD 内部不同存储芯片资源. 当多个模型需要共享 1 块 SSD 的存储空间时, GDRec 将各个模型的参数交错存储(如图 5(c)所示).

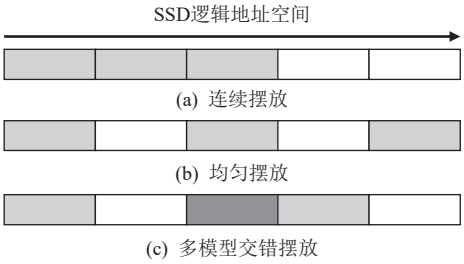

Fig. 5 Embedding placement strategy in SSD

图 5 SSD 嵌入参数放置策略

3.3 局限性

GDRec 目前的实现有 2 方面的局限性:

一方面, GDRec 仅支持静态模型的预估, 无法支持增量更新[2]的动态模型. 其原因是 GDRec 的直访机制要求在 GPU 内完成每个嵌入参数地址的定位, 动态模型需要额外维护从特征 ID 到嵌入参数地址的索引, 而这通常会超过 GPU 显存容量. 也因为同样的原因, GDRec 目前不支持冷热数据在 SSD 和 DRAM 间的迁移.

另一方面, GDRec 仅支持部分型号的 GPU.由于 GDRec 外存直访机制需要 GPU 支持 GPUDirect RDMA 机制. 因此, 对于 NVIDIA 公司推出的 GPU, GDRec 的外存直访机制可以在 NVIDIA Tesla 与 Quadro 系列的 GPU 上运行, 但无法在 NVIDIA GeForce 系列 GPU 上运行.

4 实 验

本节将通过实验对比和分析 GDRec 与现有系统的性能差异. 首先, 使用微观基准测试说明 GDRec 的 GPU 直访架构相比于传统 CPU 访存架构在性能上的优越性; 其次, 使用真实世界的推荐模型与数据

集对比测试所有系统的端到端吞吐量、延迟-吞吐曲线；最后，对比测试纯内存场景下，小型推荐模型预估时所有系统的性能.

4.1 实验平台与测试数据集

实验平台. 本实验使用的实验平台配置信息如表 1 所示. 本实验使用的 GPU 为 NVIDIA Tesla A30, 其包含 24 GB 显存，使用的 SSD 为 Intel Optane P5800X, 其 4 KB 随机读延迟为 5 μs.实验机器拥有 2 个非统一内存访问架构(non uniform memory access, NUMA)节点，为避免跨 NUMA 访问带来的性能下降，本实验只使用同一 NUMA 下的 CPU, GPU, DRAM, SSD.

Table 1 Platform Configuration

表 1 实验平台配置信息

项目	配置
CPU	Intel® Xeon® Silver 4 314 ×2
Memory	Samsung 32 GB ×6
SSD	Intel Optane P5800X
OS	Ubuntu22.04，Linux5.15.0
GPU	NVIDIA Tesla A30

数据集. 本实验共选用 3 个数据集，具体信息如表 2 所示. 数据集来自 Avazu 与 Criteo 2 家公司对真实世界点击率(click-through rate, CTR)负载的采样. 在所有数据集上，我们训练一个深度交叉网络(deep cross network, DCN) 作为推荐模型负载，其包含 6 个交叉层与形状为(1 024, 1 024)的 MLP 层. 对于 Avazu 与 Criteo-Kaggle 数据集，嵌入参数的维度被设为 32, 对于 Criteo-TB 数据集，维度被设为 128.

Table 2 Datasets for Evaluation

表 2 测试用数据集

数据集	嵌入表数	样本数/亿	ID 数/亿	模型大小/ GB
Avazu	22	0.4	0.5	5.8
Criteo-Kaggle	26	0.5	0.3	4.1
Criteo-TB	26	44	9	461

对比系统. 本实验将 GDRec 与 HugeCTR 进行性能对比.HugeCTR 是 NVIDIA 公司推出的针对推荐模型负载特别优化的预估系统.HugeCTR 使用 RocksDB 系统存储 SSD 上的参数，其会导致严重的内核 I/O 栈开销，同时其采用的 LSM-Tree(Log-structured merge-tree)数据结构对推荐模型点查询的负载并不友好. 为了公平对比与充分展现基于 GPU 直访架构的优势，我们基于 SPDK 重新实现了 SSD 参数存储模块，并替换 RocksDB，后文简称该系统为 HugeCTR-OPT. HugeCTR-OPT 可被视为极致优化的基于 CPU 访存架构的系统，其与 GDRec 仅在 I/O 提交方式上有区别.

在系统配置方面，如不作特殊说明，我们默认设置 DRAM 大小为 5% 的模型大小，模型剩余部分存储于 SSD 中. 对于 HugeCTR，我们使用 NVIDIA 提供的原生配置(RocksDB 开启 10 bit 布隆过滤器，8 MB 块缓存大小).HugeCTR 与 GDRec 均使用单个 CPU 线程处理预估请求.

所有代码使用 GCC 9.3 与 NVCC 11.3 编译，优化选项为-O2.对于 GPU，我们使用的 CUDA 版本为 11.3, cuDNN 版本为 8.2.

4.2 微观基准测试

本节测试对比基于 CPU 访存与 GPU 直访的 2 种架构在访问 DRAM 与 SSD 时的基准性能. 测试过程如下：在 GPU 中以均匀分布随机生成一定数量的特征 ID，由 2 种架构分别访问 DRAM/SSD 得到对应参数，并拷贝至 GPU 中. 该测试过程对应于推荐模型访问嵌入参数时的过程，我们分别选择 HugeCTR-OPT 与 GDRec 系统的对应功能实现用作本节的测试.

图 6 展示了当参数完全位于 DRAM 上时 2 种架构的性能. 对于同样数量的特征 ID，GDRec 的完成时延要比 HugeCTR-OPT 快 1.5~3.6 倍. 这一方面由于 GDRec 无需 CPU-GPU 之间多次通信，另一方面得益于 GDRec 避免了聚集时 CPU 额外拷贝造成的时延.

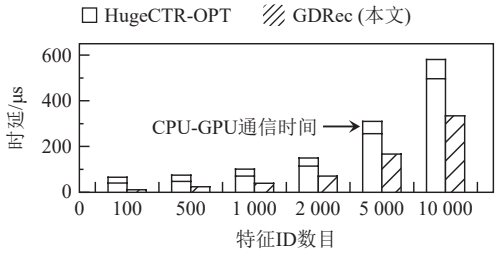

Fig. 6 Performance of accessing in-memory embeddings

图 6 内存参数访问性能对比

此外，当查询的特征数目较少时(例如 100)，HugeCTR-OPT 单就 CPU-GPU 通信时延(包括将 ID 从 GPU 传至 CPU、将参数从 CPU 传至 GPU，图 6 中使用箭头标出)，已与 GDRec 完成的总体时延相近，这充分说明了 GDRec 内存直访机制设计的优越性.

图 7 则展示了当参数完全位于 SSD 上时 2 种架构的性能.GDRec 相比于 HugeCTR-OPT 性能有 1.2~2.0 倍的提升. 其原因一方面类似于内存直访机制，外存直访机制同样节省了 CPU-GPU 通信开销与聚

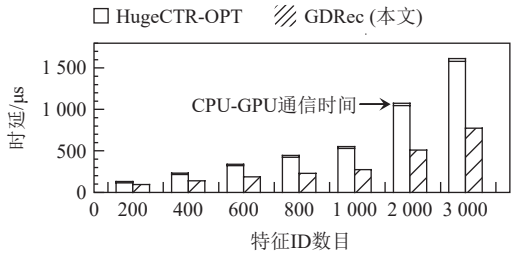

Fig. 7 Performance of accessing in-SSD embeddings

图 7 外存参数访问性能对比

集步骤的额外拷贝；另一方面，外存直访机制在提交SSD的I/O请求时可以充分利用GPU的高并行性并行提交，减少了参数读取时延.

对比图 6 和图 7，我们发现相比于在DRAM上，GDRec在SSD上获得的性能提升更少. 其原因是SSD有着比DRAM更高的硬件读取延迟，这导致GDRec可优化的软件部分时延占端到端的比例减少，进而在SSD上性能提升更少.

4.3 端到端吞吐量对比测试

图 8(a)展示了在不同批处理大小下，GDRec与对比系统的预估样本吞吐量情况. 从图 8(a)中可知：

1) 当批处理大小变大时，所有系统的吞吐量都随之上升，这是由于大的批处理大小可以更充分地利用GPU的并行性；

2) 原生HugeCTR系统吞吐最低，且与另外2个系统有着数量级上的差距，其原因是RocksDB适合于范围查询、写密集场景，并未针对推荐模型的点查询、读密集场景做定制优化；

3) HugeCTR-OPT使用SPDK定制了点查询友好的SSD参数存储模块，相比于HugeCTR获得了性能提升，但其CPU访存的架构仍然遭受1.2节所述的2个问题；

4) 相比于HugeCTR-OPT，GDRec在Avazu, Criteo-Kaggle, Criteo-TB 3个数据集上分别获得1.1~1.4倍、1.1~1.9倍与1.3~1.9倍的性能提升，性能提升来自于GDRec的GPU直访存储设计，内存直访机制与外存直访机制将CPU与GPU的通信次数降低为1次，同时零拷贝的设计消除了额外的拷贝开销，而GPU并行提交I/O命令缩短了读取SSD时控制路径的时间；

5) 随着批处理大小变大，GDRec相对于HugeCTR-OPT提升的性能更多，这是由于大批次样本情况下查询SSD的时间占比更多，此时GDRec的GPU外存直访机制的优势更得以充分体现.

进一步地，由于GDRec的所有设计仅针对于嵌入层，我们在图 8(b)单独展示不同系统在模型嵌入层的吞吐量，如图所示. 可以看到，在Avazu, Criteo-Kaggle, Criteo-TB 3个数据集上，相比于HugeCTR-OPT, GDRec分别可以获得1.1~2.1倍、1.4~2.4倍与1.4~2.7倍的性能提升. 此测试排除了GDRec未优化的模型MLP层，因此，相比于端到端预估，GDRec可以获得更多的性能提升.

4.4 延迟-吞吐量曲线测试

图 9 展示了HugeCTR-OPT于GDRec在3个数据集上的端到端预估中位数延迟-样本吞吐曲线. 由于原生HugeCTR的性能与他们有数量级的差异，我们省去其测试结果.

从图 9 中我们发现，相比于HugeCTR-OPT, GDRec可以达到更高的样本吞吐量与更低的预估时延. 例如，在Criteo-TB数据集上，给定中位数延迟为2 ms

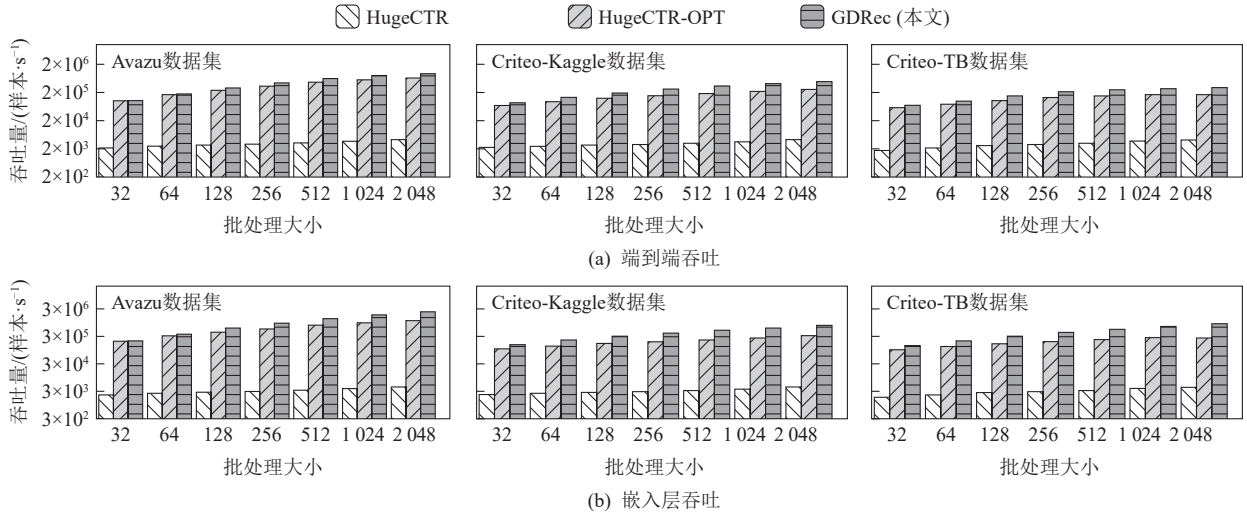

Fig. 8 System throughput comparison

图 8 系统吞吐量对比

的情况下，GDRec 提升样本吞吐量 1.9 倍，这主要是由于 GDRec 充分利用 GPU 并行性对内存与外存进行直访，可以达到更高的嵌入层吞吐，同时避免 CPU 访存架构在嵌入层的瓶颈现象；给定样本吞吐量为 15 万样本/s，GDRec 可以降低 74% 的中位数延迟，这一方面由于在数据路径上 GPU 直访省去了 CPU 聚集时在内存上的额外拷贝与大部分 CPU-GPU 通信开销，另一方面，GPU 可以并行提交 DRAM 的访存指令和 SSD 的 I/O 命令，在控制路径上具有更低的时延.

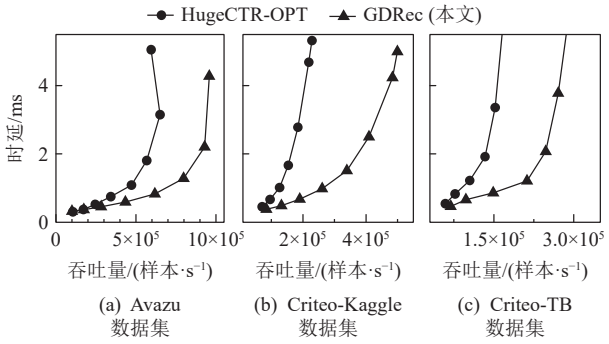

Fig. 9 Throughput vs. median latency

图 9 端到端延迟-吞吐量曲线

4.5 小模型性能对比测试

本节实验考虑小模型预估场景，即当模型可以完全存储在 DRAM 中时的情况.

图 10 展示了 GDRec 和 HugeCTR-OPT 在不同批处理大小下的样本吞吐量. 因为 Criteo-TB 数据集对应的模型容量已远超过 CPU 的 DRAM 内存大小. 这里测试用数据集选择了 Avazu 与 Criteo-Kaggle.从图 10 中我们可知：

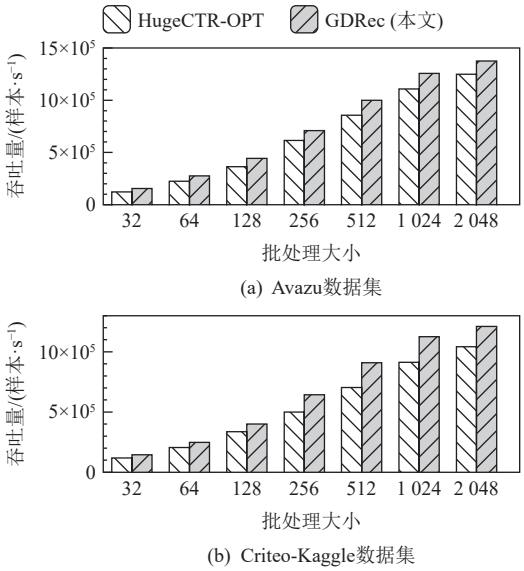

Fig. 10 End-to-end throughput when models fit in DRAM

图 10 小模型端到端吞吐量

1）GDRec 可以在纯内存场景下在 Avazu 与 Criteo-Kaggle 上分别获得 1.1~1.3 倍、1.2~1.3 倍的吞吐性能提升，这完全得益于 GDRec 的内存直访机制；

2）GDRec 的相对提升性能在较大的批处理大小情况下更少，其原因是此时 GPU 去重、CPU-GPU 通信等操作的时延增加，导致访问内存时延占比变少，因此内存直访机制可以提升的部分有限.

5 相关工作

本节从推荐模型预估系统与基于 GPU 直访存储架构的系统 2 方面介绍相关工作.

1）推荐模型预估系统. 由于推荐模型独特的访存特性，大量工作对此设计定制化的预估系统. 我们根据使用的硬件将他们分为两类.

一类工作仅使用 CPU，GPU 等通用硬件. HugeCTR-Inference[11] 是 NVIDIA 公司推出的高度优化的预估系统，其构建 GPU 显存、DRAM、SSD 多层级参数服务器存储模型参数.Fleche[27] 在 HugeCTR-Inference 的基础上优化了嵌入参数显存缓存的组织形式与查询方式，以减少缓存缺失率并提升预估性能.MERCI[25] 基于物化视图的思想缓存同时出现的嵌入参数经池化后的结果，以减少内存访问量提升性能.Bandana[10] 系统利用 NVMe SSD 存储模型以降低 DRAM 成本，其使用超图划分算法将可能同时出现的参数放于同一个数据块中，以减少块设备的读写放大现象.EVSTORE[8] 为 SSD 参数存储设计了混合型 DRAM 缓存，混合使用了组缓存、混合精度缓存与近似缓存，可以在几乎不影响模型准确率的情况下充分降低 DRAM 的使用量. 上述工作虽然在不同场景下缓解了嵌入层访存瓶颈问题，但他们在本质上仍属于传统 CPU 访存架构，并未解决 CPU-GPU 通信开销高、内存上的额外拷贝等问题.

另一类工作使用 FPGA、近内存计算（near memory processing, NMP）等定制硬件加速嵌入层的处理.MicroRec[28] 与 FleetRec[19] 利用带有高带宽内存的 FPGA 加速嵌入层的访存操作.RecNMP[29], TensorDIMM[30], FAFNIR[31] 使用近内存计算的方式，将池化操作卸载至内存；类似地，FlashEmbedding[22] 与 RecSSD[23] 将嵌入层的访存与计算操作卸载至 SSD 控制器. 在数据中心中部署这些工作需要付出额外的硬件开销，而 GDRec 则可以完全使用现有服务器

硬件部署.

2) 基于 GPU 直访存储架构的系统. 图计算、图神经网络、推荐系统等新兴应用催生了 GPU 对存储资源进行大量随机、细粒度的访问需求. 传统架构依赖 CPU 对存储资源进行访问, 但造成 CPU-GPU 通信开销大、CPU 处理时延高等问题, 严重地影响 GPU 计算效率. 在此背景下, 一些工作绕开 CPU, 使 GPU 直访存储并加速各类应用. EMOGI[32] 针对 GPU 上的图遍历负载, 利用细粒度主机内存直访加速图数据结构的访问. 伊利诺伊大学厄巴纳-香槟分校、IBM、NVIDIA 等单位合作于 2021 年提出的图卷积神经网络训练系统[33], 使用少量 GPU 线程以零拷贝的方式从主机内存预取图节点特征至 GPU 内存, 重叠计算与访存, 获得了多达 92% 的训练吞吐提升.BaM[34] 将 SSD 抽象成可供 GPU 直访的大数组, 并在其之上运行图分析与数据分析应用. 上述工作仅面向单一的内存或外存 GPU 直访, 且尚无现有工作针对推荐模型预估的独特访存流程与混合存储场景做定制设计.

6 结 论

推荐模型的参数访存已成为模型预估的性能瓶颈. 现有基于 CPU 访存架构的预估系统存在着 CPU-GPU 同步开销大和额外内存拷贝 2 个问题, 其性能已无法满足模型高速增长的访存需求. 本文提出一种基于 GPU 直访存储架构的推荐模型预估系统 GDRec, 通过在参数访存路径上移除 CPU 参与, 由 GPU 以零拷贝的方式高效直访内存外存资源. 实验显示, 相比于现有预估系统, GDRec 可以大幅提升模型预估的吞吐量, 同时降低预估延迟.

作者贡献声明: 谢旻晖进行了该论文设计、代码实现测试、论文撰写等工作; 陆游游进行了研究思路讨论与论文修改工作; 冯杨洋进行了前期实验方案的讨论设计与论文修改等工作; 舒继武进行了研究思路讨论工作.

参 考 文 献

[1] Wikipedia. Sina Weibo [EB/OL]. [2023-03-11].https://zh.wikipedia.org/wiki/%E6%96%B0%E6%B5%AA%E5%BE%AE%E5%8D%9A (in Chinese) (维基百科. 新浪微博 [EB/OL]. [2023-03-11].https://zh.wikipedia.org/wiki/%E6%96%B0%E6%B5%AA%E5%BE%AE%E5%8D%9A)

[2] Xie Minhui, Ren Kai, Lu Youyou, et al. Kraken: Memory-efficient continual learning for large-scale real-time recommendations [C/OL] // Proc of Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2020[2023-03-11].https://ieeexplore.ieee.org/document/9355295

[3] Gupta U, Wu C J, Wang Xiaodong, et al. The architectural implications of facebook's dnn-based personalized recommendation [C] // Proc of the 26th Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2020: 488−501

[4] Naumov M, Mudigere D, Shi H J M, et al. Deep learning recommendation model for personalization and recommendation systems[J]. arXiv preprint, arXiv:, 1906, 00091: 2019

[5] Cheng H T, Koc L, Harmsen J, et al. Wide & deep learning for recommender systems[C] // Proc of the 1st Workshop on Deep Learning for Recommender Systems. New York: ACM, 2016: 7−10

[6] Shan Ying, Hoens T R, Jiao Jian, et al. Deep crossing: Web-scale modeling without manually crafted combinatorial features[C] // Proc of the 22nd ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2016: 255−262

[7] Zhao Weijie, Xie Deping, Jia Ronglai, et al. Distributed hierarchical GPU parameter server for massive scale deep learning ads systems [C/OL] // Proc of the 3rd Conf on Machine Learning and Systems. 2020: 412-428[2023-03-11].https://proceedings.mlsys.org/paper_files/paper/2020/hash/6e426f4c16c6677a605375ae2e4877d5-Abstract.html

[8] Kurniawan D H, Wang Ruipu, Zulkifli K S, et al. EVSTORE: Storage and caching capabilities for scaling embedding tables in deep recommendation systems [C] // Proc of the 28th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2023: 281–294

[9] Xie Minhui, Lu Youyou, Wang Qing, et al. PetPS: Supporting huge embedding models with persistent memory[C] // Proc of the 49th Int Conf on Very Large Data Bases. New York: ACM, 2023: 1013−1022

[10] Eisenman A, Naumov M, Gardner D, et al. Bandana: Using non-volatile memory for storing deep learning models [C/OL] // Proc of the 2nd Conf on Machine Learning and Systems. 2019: 40-52[2023-03-11].https://proceedings.mlsys.org/paper_files/paper/2019/hash/d59a1dc497cf2773637256f50f492723-Abstract.html

[11] Wei Yingcan, Langer M, Yu Fan, et al. A GPU-specialized inference parameter server for large-scale deep recommendation models[C] // Proc of the 16th ACM Conf on Recommender Systems. New York: ACM, 2022: 408−419

[12] NVIDIA. Virtual memory management [EB/OL]. [2023-03-11]. https://docs.nvidia.com/cuda/cuda-driver-api/group__CUDA__VA.html

[13] Intel. Storage performance development kit [EB/OL]. [2023-03-11].https://spdk.io

[14] Liu Zhiyuan, Sun Maosong, Lin Yankai, et al. Knowledge Representation Learning: A Review[J]. Journal of Computer Research and Development, 2016, 53(2): 247−261 (in Chinese) (刘知远, 孙茂松, 林衍凯, 等. 知识表示学习研究进展 [J]. 计算机研究与发展, 2016, 53(2): 247−261)

[15] Guo Huifeng, Tang Ruiming, Ye Yunming, et al. DeepFM: A factorization-machine based neural network for CTR prediction[J].

arXiv preprint, arXiv:, 1703, 04247: 2017

[16] Wang Ruoxi, Fu Bin, Fu Gang, et al. Deep & cross network for ad click predictions [C/OL] // Proc of the 8th Int Workshop on Data Mining for Online Advertising. New York: ACM, 2017[2023-03-11].https://dl.acm.org/doi/10.1145/3124749.3124754

[17] Zhou Guorui, Zhu Xiaoqiang, Song Chenru, et al. Deep interest network for click-through rate prediction[C] // Proc of the 24th ACM SIGKDD Int Conf on Knowledge Discovery & Data Mining. New York: ACM, 2018: 1059−1068

[18] Zhou Guorui, Mou Na, Fan Ying, et al. Deep interest evolution network for click-through rate prediction[C] // Proc of the 32nd AAAI Conf on Artificial Intelligence. Piscataway, NJ: IEEE, 2019: 5941−5948

[19] Jiang Wenqi, He Zhenhao, Zhang Shuai, et al. Fleetrec: Large-scale recommendation inference on hybrid GPU-FPGA clusters[C] // Proc of the 27th ACM SIGKDD Conf on Knowledge Discovery & Data Mining. New York: ACM, 2021: 3097−3105

[20] Lian Xiangru, Yuan Binhang, Zhu Xuefeng, et al. Persia: A hybrid system scaling deep learning based recommenders up to 100 trillion parameters[J]. arXiv preprint, arXiv:, 2111, 05897: 2021

[21] Ardestani E K, Kim C, Lee S J, et al. Supporting massive DLRM inference through software defined memory[C] // Proc of the 42nd IEEE Int Conf on Distributed Computing Systems (ICDCS). Piscataway, NJ: IEEE, 2022: 302−312

[22] Wan Hu, Sun Xuan, Cui Yufei, et al. FlashEmbedding: Storing embedding tables in SSD for large-scale recommender systems[C] // Proc of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems. New York: ACM, 2021: 9−16

[23] Wilkening M, Gupta U, Hsia S, et al. RecSSD: Near data processing for solid state drive based recommendation inference[C] // Proc of the 26th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2021: 717−729

[24] Zhao M, Agarwal N, Basant A, et al. Understanding data storage and ingestion for large-scale deep recommendation model training: Industrial product[C] // Proc of the 49th Annual Int Symp on Computer Architecture. New York: ACM, 2022: 1042−1057

[25] Lee Y, Seo S H, Choi H, et al. MERCI: Efficient embedding reduction on commodity hardware via sub-query memoization[C] // Proc of the 26th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2021: 302−313

[26] Facebook. Facebook/RocksDB: A library that provides an embeddable, persistent key-value store for fast storage [EB/OL]. [2023-03-11].https://github.com/facebook/rocksdb

[27] Xie Minhui, Lu Youyou, Lin Jiazhen, et al. Fleche: An efficient GPU embedding cache for personalized recommendations[C] // Proc of the 17th European Conf on Computer Systems. New York: ACM, 2022: 402−416

[28] Jiang Wenqi, He Zhenhao, Zhang Shuai, et al. MicroRec: Efficient recommendation inference by hardware and data structure solutions [C/OL]. Proc of the 4th Conf on Machine Learning and Systems. 2021: 845-859[2023-03-11].https://proceedings.mlsys.org/paper_files/paper/2021/hash/9e9a5486cb2f8e44d5b5fedd2a9e5fcd-Abstract.html

[29] Ke Liu, Gupta U, Cho B Y, et al. Recnmp: Accelerating personalized recommendation with near-memory processing[C] // Proc of the 47th ACM/IEEE Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE 2020: 790−803

[30] Kwon Y, Lee Y, Rhu M. Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning[C] // Proc of the 52nd Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2019: 740−753

[31] Asgari B, Hadidi R, Cao J, et al. Fafnir: Accelerating sparse gathering by using efficient near-memory intelligent reduction[C] // Proc of the 27th IEEE Int Symp on High-Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE 2021: 908−920

[32] Min S W, Mailthody V S, Qureshi Z, et al. Emogi: Efficient memory-access for out-of-memory graph-traversal in GPUs[J]. arXiv preprint, arXiv:, 2006, 06890: 2020

[33] Min S W, Wu Kun, Huang Sitao, et al. Large graph convolutional network training with GPU-oriented data communication architecture[J]. arXiv preprint, arXiv:, 2103, 03330: 2021

[34] Qureshi Z, Mailthody V S, Gelado I, et al. BaM: A case for enabling fine-grain high throughput GPU-orchestrated access to storage[J]. arXiv preprint, arXiv:, 2203, 04910: 2022

Xie Minhui, born in 1997. PhD candidate. Student member of CCF. His main research interests lie in storage systems and machine learning systems.

谢旻晖, 1997 年生. 博士研究生. CCF 学生会员. 主要研究方向为存储系统和机器学习系统.

Lu Youyou, born in 1987. PhD, associate professor, PhD supervisor. Senior member of CCF. His main research interests lie in storage systems.

陆游游, 1987 年生. 博士, 副教授, 博士生导师. CCF 高级会员. 主要研究方向为存储系统.

Feng Yangyang, born in 1998. PhD candidate. Student member of CCF. His main research interests lie in storage systems and machine learning systems.

冯杨洋, 1998 年生. 博士研究生. CCF 学生会员. 主要研究方向为存储系统和机器学习系统.

Shu Jiwu, born in 1968. PhD, professor, PhD supervisor and fellow of CCF. His main research interests include non-volatile memory systems and technologies, storage security and reliability, and parallel and distributed computing.

舒继武, 1968 年生. 博士, 教授, 博士生导师, CCF 会士. 主要研究方向为非易失内存存储系统与技术、存储安全与可靠性、并行与分布式计算.