arXiv:2305.01515v1 [cs.IR] 19 Apr 2023

MTrainS：利用异构内存提升 DLRM 训练效率

Hiwot Tadese Kassa[1] Paul Johnson[1] Jason Akers[1] Mrinmoy Ghosh[1] Andrew Tulloch[1] Dheevatsa Mudigere[1] Jongsoo Park[1] Xing Liu[1] Ronald Dreslinski[2] Ehsan K. Ardestani[1]1Meta Platforms,Inc. [2]密歇根大学

摘要

推荐模型规模庞大，在训练期间需要太字节（TB）级内存。为了提升模型质量，模型的规模和复杂度会随时间不断增长，这就需要更多训练数据来避免过拟合。模型增长也使数据中心需要大量资源。因此，提高训练效率对于控制数据中心的功耗需求变得愈发重要。在深度学习推荐模型（DLRM）中，通过嵌入表处理类别型输入的稀疏特征是模型规模的主要来源，并且需要较高的内存带宽。本文研究了实际部署模型中嵌入表的带宽需求和局部性。我们发现，不同表的带宽需求并不均衡，而且嵌入表具有很强的时间局部性。随后，我们设计了 MTrainS，通过层级结构利用异构内存，包括支持字节和块寻址的存储级内存，以用于 DLRM。对于内存容量受限的用例，MTrainS 提高了每个节点的内存容量，并通过降低横向扩展至多个主机的需求来提升训练效率。通过优化平台内存层次结构，我们将训练所需的节点数量减少了 4-8×，在满足目标训练性能的同时，节省了训练功耗和成本。

1 引言

推荐模型广泛应用于大型科技公司，以个性化其受众的体验。例如，谷歌使用此类模型投放个性化广告[10], 亚马逊和阿里巴巴使用此类模型推荐其目录中的商品[36, 43], 微软使用此类模型向用户推荐新闻[13], Meta使用此类模型进行排序和点击率预测 [30]。

推荐模型规模非常庞大，在训练期间需要太字节（TB）级内存 [28],，而在推理期间则需要数百吉字节（GB） [7]。加速器平台，例如配备数十至数百个加速器的 GPU 系统 [31],，通常用于训练此类模型 [28, 42]。这些模型会占用数据中心的大量资源。例如，推荐模型

在 Meta [29]，训练消耗了超过 50% 的人工智能训练资源。为了提高推荐质量，模型容量和复杂度都在以每年超过 1.5× 的速度增长 [23],，这需要更多训练数据来避免过拟合。以同样的速度训练参数量多出 n 倍的模型，需要的数据量也多出 m 倍，从而使 O 的资源和功耗需求增加 $n \times m$。因此，在数据中心中，提高推荐模型训练效率以管理资源和功耗需求变得愈发重要。

深度学习推荐模型（DLRM）是基于神经网络的个性化与推荐模型 [30]。

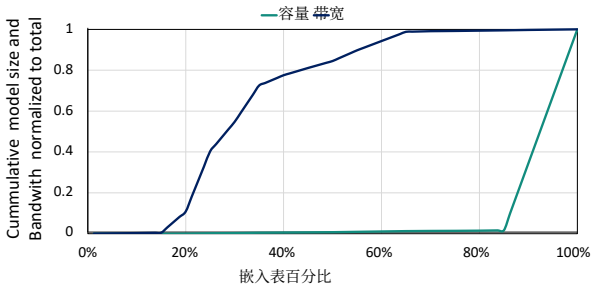


图 1：我们评估的一个真实世界模型的累计带宽和内存容量。

在 DLRM 中，通过嵌入表捕获类别型输入的稀疏特征是太字节级模型规模的主要来源，而由多层感知机（MLP）构成的稠密特征则影响模型计算复杂度。除了对内存容量有较高要求外，稀疏特征还可能需要较高的内存带宽。由于 DLRM 的资源需求和增长速度都很可观，在数据中心中容纳此类模型的典型解决方案是将模型横向扩展到多台主机。我们可以将 Meta 将 DLRM 部署横向扩展到单台主机之外的原因归纳如下：

- **内存容量受限**：训练模型规模（模型参数、优化器状态和激活值）决定了满足内存容量需求所需分配的最少主机数量（高带宽内存和动态随机存取存储器）。
- **受计算能力限制/带宽受限**：根据模型的计算强度（例如，以拍 FLOPS/秒·天为单位）和/或内存带宽需求，使用一定数量的 GPU（主机）来提升训练速度（例如，每秒查询数），以达到预期目标。

生产环境中已开发并部署了属于上述两类的各种 DLRM 工作负载配置。每类模型对底层硬件的利用方式各不相同，因此在提升性能和效率方面面临着独特的挑战。其中一类配置受内存容量限制。在这种情况下，横向扩展的程度主要取决于每台主机最多可容纳的模型参数数量。图 1 展示了我们数据中心中一种最重要的容量受限模型用例，其累计内存容量和带宽需求均按总容量归一化。图 1 中，每个嵌入表的容量根据行数和嵌入维度计算得出。此外，将表的池化因子（每个样本平均读取的行数）乘以嵌入维度大小和目标每秒查询数，

即可计算出每个表的带宽需求。有趣的是，图中大多数容量较大的嵌入表带宽需求相对较低，而带宽贡献最大的表容量较小。利用高带宽内存 (HBM)、动态随机存取存储器和存储级内存 (SCM) 等异构内存构成的层级结构，可以用更少的主机同时满足容量和带宽需求。这样可以提升训练效率：只有在计算或带宽需求确有必要时，才需要横向扩展到多台主机。

存储级内存（SCM）是一类兼具内存和存储特性的技术。SCM 通过提供在容量、带宽、功耗和成本方面各具特点的内存，弥补了高带宽内存和动态随机存取存储器的不足。SCM 可作为字节寻址内存使用（采用双列直插式内存模块，或未来采用 Compute Express Link（CXL）[1] ），从而在延迟和带宽接近动态随机存取存储器的情况下，将容量提升 ~5×；也可作为块寻址存储使用（采用非易失性内存快速通道 NVMe），容量更大，但延迟更高、带宽更低 ~20× 。这种灵活性使不同内存技术能够通过平衡每台主机的带宽和容量，覆盖广泛的训练系统解决方案。然而，这种设计也面临挑战。在开发将模型参数分布到不同内存类型中的方案时，必须考虑密度更高的内存类型所带来的更高延迟和更低带宽。

先前研究已证明，SCM 可作为动态随机存取存储器的补充，带来诸多益处 [22, 24, 37, 39, 40]。但对于在商业数据中心中将这些技术用于推荐系统所面临的挑战及其优势，人们知之甚少。近期研究 [12, 38] 探讨了如何将块寻址存储用于嵌入表，但重点关注的是推理；推理对容量和内存带宽的需求与训练不同，因此给硬件带来了不同的挑战。先前研究也主要关注将块寻址存储与中央处理器搭配使用。基于 GPU 的加速器正逐渐成为训练推荐系统的常见选择。因此，研究如何将带宽较低的 SCM 与内存带宽需求较高的 GPU 配合使用至关重要。

本文分析了 Meta 上多种 DLRM 部署的特征，发现其模型主要受容量限制和带宽限制。随后，我们研究了这些模型的带宽、容量以及局部性并确定模型内嵌入表之间不均匀的容量和带宽需求。我们的研究还表明，嵌入表的空间局部性较低，但具有较高的时间局部性，其分布遵循幂律 [11] 。这些特征使我们的深度学习推荐模型（DLRM）适合采用包含高带宽内存、动态随机存取存储器以及可按字节和块寻址的存储级内存的分层内存。随后，我们设计了 MTrainS，这是一种端到端训练器，可与图形处理器协同，高效利用异构内存层次结构。MTrainS 在固态硬盘中部署了一个键值 [15] 存储系统，用于管理大型（TB 级）嵌入表。随后，为了掩盖固态硬盘带宽较低、延迟较高的问题，MTrainS 实现了一种由图形处理器管理、基于软件且可配置的分层缓存，分别使用动态随机存取存储器和存储级内存存放热数据和冷数据的嵌入行。我们的实验采用了两个资源消耗极大的模型，分别代表容量受限和带宽受限的工作负载。结果表明，对于容量受限模型，使用 MTrainS 后，我们可以将当前模型的训练所需主机数量减少 4× ，并将未来扩展模型的训练所需主机数量减少 8× ，同时达到我们的服务级别协议（SLA）每秒查询数目标。据我们所知，这是首项研究在商业数据中心的 GPU 系统中使用存储级内存进行推荐系统训练的工作。总而言之，我们作出以下贡献：

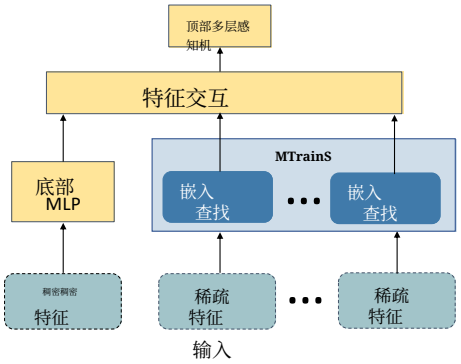


图 2：DLRM架构以及 MTrainS 在 DLRM 中的作用。

- 将 DLRM 训练的内存层次结构扩展到高带宽内存和动态随机存取存储器之外，延伸至可按字节和块寻址的存储级内存。
- 针对大规模生产环境中的真实工作负载进行表征和实验，并探讨此类分层内存在哪些真实场景下能够提升效率。
- 讨论在 DLRM 训练用例中，字节和块可寻址存储级内存的系统级性能权衡。
- 扩展开源 DLRM，支持不同的内存配置，以便针对更大规模的人工智能模型开展硬件与软件研究。

2 背景

本节将介绍深度学习推荐模型（DLRM）的架构、FBGEMM_GPU（一种我们在设计中使用的优化型开源图形处理器内核库），以及我们系统中的各种内存和存储组件。

2.1 深度学习推荐模型

推荐系统广泛用于根据用户偏好和交互行为对新闻动态、视频和产品等内容进行排序。为了准确地对用户偏好进行排序，DLRM等推荐系统 [30] 使用深度神经网络。然而，与深度神经网络不同，DLRM具有稠密和稀疏特征，用于捕获用户和物品属性，因此就底层系统而言呈现出不同的特征。

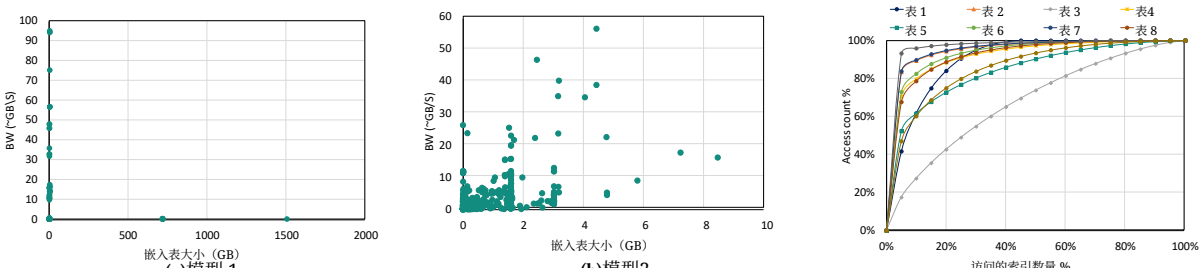
*2.1.1 DLRM*架构和组件。图2展示了DLRM的内部组件，具体细节如下：

输入特征：用户数据和产品由稠密和稀疏输入特征表示。**稀疏**特征表示类别型输入，例如用户在页面列表中喜欢的某个页面。顾名思义，这类数据具有稀疏性，即用户很可能只与数十亿个可用页面中的一个很小的子集进行过交互。**稠密**特征包括连续输入，例如用户年龄。

嵌入表：类别型输入的一种朴素表示方式是使用二进制向量。例如，如果我们有 4 个 ID 从 0 到 3 的页面，且用户喜欢 ID 为 0 和 2 的页面，那么该用户的嵌入向量

特性	NAND闪存固态硬盘	傲腾固态硬盘	DRAM	傲腾内存	HBM
功耗（mW / GB）（高带宽内存为 mW / GB/ s）	5.7	35	375	98	5000
相对于 NAND闪存固态硬盘的每GB成本	1	10.4	68.8	26.5	-
访问粒度	块	块	byte	byte	byte
每主机总容量 (GB) /每主机总带宽 (GB/s)	8192/6	2048/6	384/200	2048/84	320/12800
耐久性（每日整盘写入次数）	0.8	100	-	-	-

表 1：根据产品规格整理的每模块 NAND 固态硬盘、傲腾固态硬盘、动态随机存取存储器、傲腾内存和高带宽内存的特性。



(a)模型 1 (b)模型2 (c)嵌入表的时间局部性分析.

图3：模型 1 和模型2的带宽与容量分布，以及各种嵌入表的局部性分析。

将为 (1,0,1,0)。但是，我们拥有数十亿个页面，因此这种表示方式会非常庞大且稀疏。此外，二进制向量无法表示相似页面之间的关系。为避免这些问题，DLRM 使用嵌入表将类别特征映射为稠密表示。在这种情况下，页面等类别型输入将由一个较短的向量表示，并且相似页面在欧几里得空间中的位置会更接近。在嵌入表中，列表示嵌入向量，行表示某个类别中的项目。在一个模型中，有多个类别特征，并且不同表中的行数各不相同。嵌入表算子会在嵌入表中查找部分行，并使用 求和、平均值 或最大值 [33]对结果进行池化。**底部多层感知机层：** 连续输入由底部多层感知机（MLP）转换并投影到稠密空间中，该多层感知机由一系列全连接（FC）层和激活函数组成。

特征交互和顶部多层感知机：类别特征和连续特征的稠密投影会被聚合（例如通过拼接），随后由一组多层感知机层捕获不同特征之间的交互。

2.1.2 嵌入表和运算符。 嵌入表给推荐系统的系统设计带来了独特挑战。实际应用中的嵌入表需要大量内存容量，最高可达数十 TB。公式1给出了包含 T 个嵌入表的模型的内存容量需求，其中每个表平均有 H行（也称为嵌入表哈希大小），每行包含 D 个元素（嵌入向量维度），元素精度为 p 字节。

$$Memory(SparseParameters) = T \times H \times D \times p \quad (1)$$

鉴于嵌入表的容量巨大，每行状态数量较少的典型优化器（例如 Adagrad [27], ）通常用于稀疏特征。我们可以重写公式1，使其同时包含模型参数和优化器状态（o）。

$$MemoryCapacity = T \times H \times (D + o) \times p \quad (2)$$

嵌入表同样对内存带宽要求很高，因为每个训练样本在每次嵌入查找中都会访问多行。假设每个训练样本对每个表访问 L 行，公式3将嵌入表训练的带宽需求表述为达到给定的每秒查询数。由于前向和反向传播都会消耗所有已访问的行，因此该公式要乘以2。

$$MemoryBW = QPS \times T \times D \times p \times L \times 2 \quad (3)$$

2.2 FBGEMM_GPU 内核库

本文所有软件开发工作均基于 FBGEMM_GPU（FBGEMM GPU 内核库） [2, 25], 展开。该库是用于深度神经网络训练和推理的高性能 GPU CUDA 算子库。FBGEMM_GPU 提供高效的嵌入表算子、数据布局变换和其他优化。它利用基于高带宽内存的缓存和统一虚拟内存实现动态随机存取存储器利用，从而支持高效访问嵌入表。我们扩展了 FBGEMM_GPU，添加了用于基于 SSD/SCM 的训练的算子，以及各种缓存机制，以隐藏 SSD 较高的访问延迟。

2.3 存储和内存类型

我们研究了不同的存储和内存技术，以便每个节点容纳更大的模型。每种技术都有其优缺点。表 1 展示了各项技术的特性。具体说明如下：

NAND闪存固态硬盘：如表 1 所示，这是我们采用的密度最高、成本最低的内存技术。然而，与我们评估的其他内存类型相比，这些固态硬盘的每秒输入输出操作数（IOPS）较低，且延迟较高。此外，它们的写入耐久性有限，以每日整盘写入次数（DWPD）定义。基于 NAND 闪存的解决方案受限于较低的 IOPS，因此最适用于内存容量大、带宽要求低且范围有限的稀疏特征。

傲腾固态硬盘： 它基于英特尔的 3D XPoint 技术。与 NAND 闪存固态硬盘相比，它具有更高的每秒输入输出操作数，尤其适用于访问粒度较小的请求，并且延迟更低。这些固态硬盘的读写延迟较为均衡，并且具有 100× 更好的写入耐久性（每日整盘写入次数）（见表 1）。然而，与 NAND 固态硬盘相比，这些固态硬盘的每 GB 价格更高。我们将这种内存称为 BLA-SCM，即可按块寻址的存储级内存。

Optane 内存（PMEM）： Intel Optane 内存位于内存层次结构中的动态随机存取存储器和固态硬盘之间。与动态随机存取存储器相比，它是一种成本更低的替代内存，密度高出 4-8× 倍，但

带宽较低，延迟较高。我们将这种内存称为 BYA-SCM，即字节可寻址存储级内存。BYA-SCM 可工作在内存模式和应用直连模式下。在**内存模式**下，动态随机存取存储器充当直接映射缓存，而 BYA-SCM 则作为单一易失性内存区域呈现。动态随机存取存储器缓存和 BYA-SCM 的访问完全由 CPU 内存控制器处理，应用程序无法控制其内存分配的位置（动态随机存取存储器缓存或 BYA-SCM）。在**应用直连模式**下，动态随机存取存储器和 BYA-SCM 配置为两种独立的内存。此时，应用程序可以完全控制对每种内存的读写访问。请注意，对于我们使用 DLRM 进行训练的场景，这些内存的持久性特征并不重要。

动态随机存取存储器： 动态随机存取存储器具有高带宽和低读写延迟。然而，如表 1 所示，与 BYA-SCM 和固态硬盘相比，其每 GB 成本更高、密度更低，而且每 GB 功耗更高。此外，动态随机存取存储器的最大容量受主机上可用双列直插式内存模块数量的限制。

高带宽内存： 许多面向高性能计算/人工智能训练的现代 GPU 使用一种速度甚至快于动态随机存取存储器的内存技术。高带宽内存 (HBM) 的内存带宽高于传统动态随机存取存储器。这些内存模块焊接在 GPU 上，因此容量是固定的。因此，每个 GPU 可用于嵌入表存储的容量有限。

	模型 1	模型 1+	模型2
特征	~ 10*s*	~ 10*s*	~ 100*s*
总带宽（GB/s）	1300	2600	7136
嵌入维度	128	256	128
数据类型	4 字节	4 字节	4 字节
平均池化因子	33	33	18
多层感知机层数	7	7	20

表 2：工作负载规格。

3 工作负载特征分析

DLRM 工作负载庞大的容量和带宽需求给系统设计带来了挑战。我们研究了生产环境中各种已部署 DLRM 模型的嵌入表容量、带宽和局部性，以了解如何利用异构内存提高其性能和功耗效率。我们选取了数据中心中两个最具代表性的模型，它们具有不同的特性，具体如下。模型 1 用于各种服务中的内容排序，模型 2 用于预测用户内容的点击率（CTR），并进行物品推荐。模型 1+ 是模型 1 的未来扩展，其带宽和局部性特性与模型 1 相似。这些模型的容量达到 TB 级。其特性如表 2 所示。

3.1 带宽和容量分布

图3a和3b展示了模型 1 和模型2 中嵌入表带宽与大小分布的一个样本。我们使用公式1和公式3来研究容量和带宽。我们根据生产环境配置中的表行数、维度和精度，计算嵌入表的容量。然后，我们使用数据中心中训练每个模型时可接受的每秒查询数（SLA）和池化因子来计算带宽。池化因子是根据大量历史数据确定的，这些数据记录了每个嵌入表在每次查找中被访问的次数。如图所示，这两个模型的带宽与容量分布各不相同。模型 1 中，较小容量的嵌入表带宽需求较高，而较大容量的嵌入表带宽需求较低。这类表天然适合采用混合内存系统，该系统同时包含大容量／低带宽和小容量／高带宽组件。模型 1 在目标每秒查询数下的累计带宽，可由单个高带宽内存+动态随机存取存储器系统满足。在这里，我们将模型横向扩展到多台主机，以便用更大的内存容量容纳模型参数。相反，模型2 中的嵌入表数量要多得多，且容量和带宽差异显著。在这种情况下，我们将模型横向扩展到多台主机，以同时增加内存容量和带宽。由于容量／带宽需求差异很大，模型的带宽和容量分布是 DLRM 内存层次结构设计中的关键因素。

3.2 嵌入表查找中的局部性

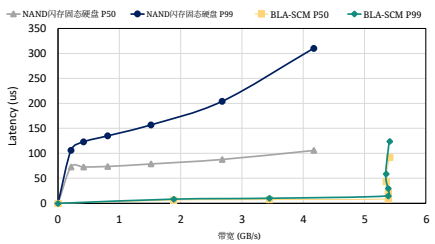
如 [7], 中所讨论的，由于嵌入表表示稀疏类别特征，嵌入表的空间局部性很低，且嵌入行访问非常不规则。不过，我们的训练器设计中存在显著的时间局部性，因此缓存非常有效。为了研究时间局部性，我们考察了模型 1 和模型2 中若干嵌入表的索引在生产环境运行 24 小时期间的访问频率。图3c 展示了具有代表性的嵌入表的结果。图中显示，大多数表的访问都遵循幂律分布。如 [7], 所示，对于大多数表，我们观察到，80 % 的索引访问来自总索引的 10%–40%。因此，我们可以利用异构内存和存储：将较冷的嵌入表和嵌入行放置在容量大但速度较慢的内存中，例如 BYA-SCM、BLA-SCM 或 NAND闪存固态硬盘；而较热的嵌入表和嵌入行则可通过缓存，继续使用速度更快但容量较小的内存，例如高带宽内存和动态随机存取存储器，这得益于 DLRM 工作负载较高的时间局部性。在第 5 节介绍的设计中，我们重点说明如何通过分层缓存最大限度地利用缓存来隐藏延迟并提供更高的带宽。

4 系统设计挑战与考量

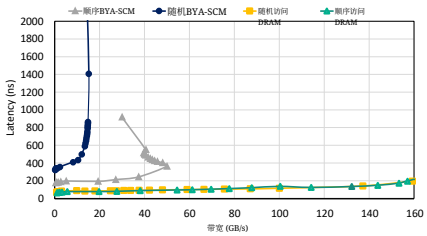
本节讨论在 DLRM 训练中采用异构内存和存储所面临的挑战，以及我们在设计中对工作负载和硬件特性的考量。

4.1 内存性能评估

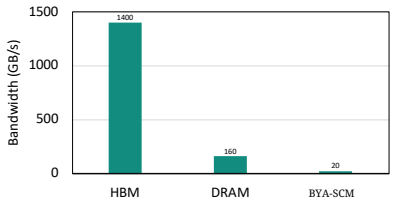
我们比较了内存和存储的延迟与带宽，以了解它们如何适应我们的工作负载特性。我们使用 FIO [8] 测量 BLA-SCM 和 NAND闪存固态硬盘在不同队列深度下处理随机读工作负载时的延迟和带宽，以提高带宽利用率。在图 4a 中，BLA-SCM 在带宽相近时，P50 和 P99 延迟均处于~10µs 量级；而 NAND闪存固态硬盘的延迟为 100µs，且 P99 延迟显著高于 P50。还要注意，提高 NAND闪存固态硬盘的带宽利用率会增加访问延迟。这表明，对于 NAND闪存固态硬盘，我们必须谨慎控制带宽利用率，以避免延迟显著增加。鉴于我们的模型具有较高的时间局部性，



(a)BLA-SCM 和 NAND 固态硬盘的 P50、P99 延迟及带宽。



(b)动态随机存取存储器和 BYA-SCM 的延迟与带宽。



(c) 高带宽内存、动态随机存取存储器和 BYA-SCM 的带宽对比。

图 4：系统内存和存储特性分析。

采用缓存来减少固态硬盘流量是有利的，尤其是考虑到 NAND闪存固态硬盘的性能限制。

我们使用英特尔的内存延迟检测器（MLC） [17] 测量动态随机存取存储器和 BYA-SCM 的延迟与带宽。我们采用不同内存流量速率下的顺序读取和随机读取工作负载。在图 4b 中，BYA-SCM 的带宽达到 ~15 GB/s，动态随机存取存储器的带宽为 170 GB/s。此外，随着内存流量增加，BYA-SCM 的延迟也随之上升（顺序读取时为 ~200 ns - 800ns，随机读取时为 ~350ns - 1500ns），并且在流量较高时带宽会达到饱和。之后，延迟继续增加，但带宽不再变化。然而，动态随机存取存储器的延迟远低于 BYA-SCM，带宽则远高于 BYA-SCM，并且其顺序访问和随机访问的延迟相同。虽然在我们的工作负载中，空间局部性相对于固态硬盘的 4KB 块访问粒度较低，但 BYA-SCM 的顺序访问粒度为 256 字节。我们工作负载中的嵌入查找访问粒度为 512-1024B。因此，我们仍可实现 BYA-SCM 的顺序访问性能。不过，由于动态随机存取存储器和 BYA-SCM 在带宽与延迟方面存在差异，最实际的设计是采用分层配置：使用动态随机存取存储器存放热点嵌入行，使用 BYA-SCM 存放较冷的嵌入行。我们还应考虑流向 BYA-SCM 的流量，以避免延迟过高和带宽饱和。

我们使用 gpumembench[26] 测量 HBM 的带宽，并使用 MLC 测量 DRAM 和 BYA-SCM 的带宽，以比较它们之间的带宽差异。如图 4c 所示，HBM 的带宽显著高于 DRAM 和 BYA-SCM。这些带宽和容量差异（如表 1 所示）表明，我们需要优化嵌入表及其行在这些存储器中的放置方式，以最大限度地提高带宽利用率。

4.2 每秒输入输出操作数与带宽

尽管嵌入表具有时间局部性，但相邻行的访问并非按顺序进行，因此缺乏空间局部性。此外，嵌入维度通常为 64 到 256（即单精度下为 256–1024 字节）。对于 BLA-SCM 和 NAND 闪存固态硬盘等块寻址技术，每次访问嵌入行所需的数据可能少于块大小（例如 4KB），造成带宽浪费（称为读放大）。为考虑读放大的影响，对于块寻址单元，我们跟踪每秒输入输出操作数，而非带宽，并研究如何利用这些内存来满足每秒输入输出操作数需求。公式 4 给出了所需的每秒输入输出操作数，假设 T_B 表存放在固态硬盘上，平均池化因子为 L_B。α 用于考虑访问嵌入表时的局部性，从而减少对较低层级块寻址内存的访问。

$$IOPS = QPS \times T_B \times L_B \times \alpha \quad (4)$$

在不考虑局部性的情况下，为满足我们的目标每秒查询数，将模型 1 和模型2完整地容纳在一个节点中，同时将 T_B 表存放在固态硬盘上，所需的每秒输入输出操作数分别为 6.25M 和 75M。通常，固态硬盘的每秒输入输出操作数上限约为 500K–1M。例如，模型 1 的缓存命中率若为 70%，每秒输入输出操作数将为 1.875M。而对于局部性为 70% 的模型2，仍需 22.5M 每秒输入输出操作数。因此，局部性不仅对优化存储级内存的高延迟至关重要，也有助于我们在硬件的每秒输入输出操作数上限内运行。因此，我们在设计中着重于最大化模型的局部性。

4.3 耐久性

NAND闪存固态硬盘和 BLA-SCM [1] 在存储单元磨损失效之前，可执行的编程/擦除周期数有限。该指标衡量的是在整个使用寿命期间（通常为 3-5 年），每天可对整个驱动器写入的次数，称为每日整盘写入次数（DWPD）。公式 5 给出了在给定每秒查询数下训练期间每天写入的数据量，其中 T_B 表存储在固态硬盘上，L_B 平均池化因子、每行D 个元素、 α 局部性，以及 p 字节的元素精度。

$$write/day = 24 \times 3600 \times QPS \times T_B \times L_B \times D \times p \times \alpha \quad (5)$$

例如，对于模型 1，我们每天向固态硬盘写入 ~10TB；对于模型2，则写入 ~100TB，同时将 T_B 个表放置在固态硬盘中。DLRM 工作负载的写入密集度很高。使用固态硬盘存储嵌入表时，我们希望将每日写入量控制在标称的每日整盘写入次数限制之下，以避免硬盘过早故障。

4.4 工作负载扩展

包括 DLRM 在内的深度学习模型，其复杂度和容量都在迅速增长。模型中使用了许多稀疏特征，因此，嵌入表的数量以及每个表的嵌入维度，是推动 DLRM 工作负载内存容量增长的主要因素之一。任何系统解决方案都需要能够在系统的整个生命周期内考虑这种扩展。本文针对当前模型设计系统，并测试我们的设计能否适应未来的扩展。

4.5 软件设计选择

软件设计首先要考虑如何高效利用基于块的存储。块设备每次至少必须写入一个块，即使只有一个字节发生了变化。在

[1]本文所称的采用 DIMM 外形规格的 Optane（简称 BYA-SCM）据称不受耐久性限制

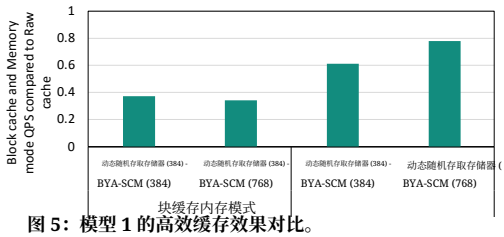


图 5：模型 1 的高效缓存效果对比。

为了高效利用硬盘，我们使用 RocksDB [15]。RocksDB 是一种键值存储引擎，可提供低延迟的数据库操作。RocksDB 的结构（例如高效的数据库分片）有助于快速访问存储。此外，RocksDB 使用内存数据结构来加快写操作。由于新的写入会先进入内存（动态随机存取存储器），RocksDB 可以将多次写入合并为一次较大的连续硬盘写入。这显著减少了固态硬盘写入，并延长了固态硬盘寿命。

其次，由于嵌入表具有很强的时间局部性，缓存能够有效隐藏固态硬盘的延迟。不过，要充分利用动态随机存取存储器和 BYA-SCM，我们必须考虑如何组织缓存。我们首先考察复用现有 RocksDB 块缓存的方式。该缓存利用内存缓存数据，以便快速读取，并将数据存储在动态随机存取存储器/持久内存中，如[24]所示。另一种方案是使用英特尔傲腾内存（内存模式）自带的硬件管理型动态随机存取存储器缓存。在这种模式下，硬件会透明地将动态随机存取存储器用作 BYA-SCM 的直接映射缓存。我们将这两种方法与原始缓存进行了比较，原始缓存的访问粒度等于嵌入表行维度。原始缓存以应用直通模式分层使用动态随机存取存储器和 BYA-SCM，其中动态随机存取存储器是一级缓存，BYA-SCM 是二级缓存。图 5 比较了模型 1 在两种配置下，原始缓存与块缓存和内存模式的每秒查询数。块缓存旨在为只读场景提供最佳性能，因此我们发现，在读写比例为 50/50 的 DLRM 模型训练中，它在两种硬件配置下的性能都不理想，仅为原始缓存性能的 0.35–0.38×。这是因为嵌入行一旦更新，其在磁盘上的位置就会因写入压缩而改变，因此无法再通过原来的块缓存行访问。此外，块缓存会导致数值被重复缓存，浪费容量。同样，从图 5 还可以看出，内存模式也无济于事，因为它会造成重复缓存。因此，设计一种能够充分利用动态随机存取存储器和 BYA-SCM 统一容量的缓存至关重要。基于这些原因，我们采用应用直通模式设计了独占式分层缓存，以全面控制访问粒度以及嵌入表和行的放置。

5 MTRAINS 设计

5.1 概述

在 DLRM 系统设计中添加异构内存，需要考虑内存技术的延迟、容量和带宽

上述内容讨论了差异，并进一步审视了我们的模型。我们设计了 MTrainS，这是一条端到端训练流水线，可在 DLRM 中利用高带宽内存、动态随机存取存储器、BYA-SCM、BLA-SCM 或 NAND闪存固态硬盘。MTrainS 将嵌入表存储扩展到 BLA-SCM 和/或

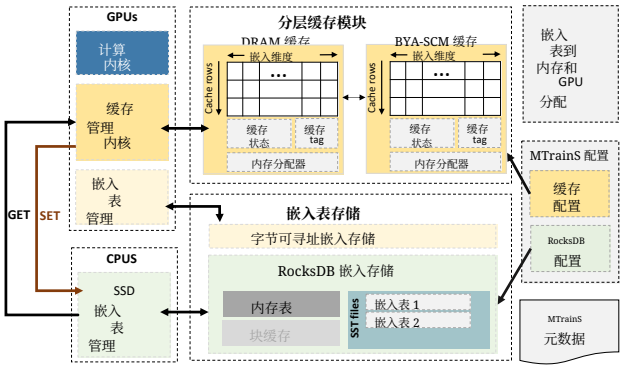


图 6：MTrainS 组件与架构。

NAND闪存固态硬盘，并采用 **基于 RocksDB 的嵌入表存储**，使我们能够灵活地支持容量各异、嵌入带宽需求不同的 TB级模型。基于模型的时间局部性（见第 3.2 节），MTrainS 为存储在固态硬盘中的嵌入表实现了**分层缓存模块**，并采用最近最少使用（LRU）策略，以隐藏较大的固态硬盘延迟。该模块使用动态随机存取存储器和 BYA-SCM 进行缓存，将热门嵌入行放在动态随机存取存储器中以实现快速访问，将较冷的数据放在 BYA-SCM 中，从而提供速度较慢但容量充足的缓存空间。鉴于嵌入表的容量和带宽需求范围很广（见第 3.1 节），为最大限度地利用内存和存储的容量与带宽，MTrainS 使用**嵌入表放置**方案，该方案基于混合整数线性规划求解器，以表容量和每次访问的数据量（池化因子）为输入，以内存容量和带宽为约束，旨在提高嵌入表的带宽并缩短表的访问时间。图 2 展示了 MTrainS 在 DLRM 中的位置概览，图 6 展示了 MTrainS 的架构。

5.2 嵌入表存储

MTrainS 使用基于 RocksDB 的存储，将嵌入表存放在块寻址存储（BLA-SCM 和 NAND闪存固态硬盘）中。将表存储在固态硬盘中，可为大容量嵌入表提供每个节点充足的存储空间。RocksDB[15] 是一种针对高速存储进行优化的键值存储。在嵌入表中，键是嵌入表行索引，存储的值是包含权重的嵌入行。RocksDB 以排序字符串表（SST）格式将键值数据库组织成块（在我们的实现中为 4KB）。我们对这些嵌入表的数据库（SST文件）进行分片，以实现负载均衡和快速键查找。我们的 RocksDB 实现使用位于动态随机存取存储器中的内存表来优化写入。不过，如上文所述，我们关闭了块缓存，因为它无法使我们的 DLRM 工作负载受益。为提高读取速度，我们使用 RocksDB 中针对批量查找进行优化的 MultiGet() [14] API。

MTrainS 还允许使用二维张量将嵌入表放置在按字节寻址的内存（高带宽内存、动态随机存取存储器、BYA-SCM）中。这些张量在所有内存类型中都具有相同的结构，但内存分配器参数不同。

5.3 分层缓存模块

我们利用动态随机存取存储器和 BYA-SCM 作为 RocksDB 嵌入存储之上的软件管理的缓存，以掩盖固态硬盘带宽低、延迟高的问题。分层缓存模块包含一个由缓存类派生的缓存列表，以及一个可配置的缓存层级结构，用于将缓存列表组织成多个层级。图 6 展示了缓存概览。

5.3.1 缓存类。 此类用于存储位于固态硬盘中的嵌入表热点行。由于嵌入表缺乏空间局部性，该类具有一个**缓存内存**，用于保存原始嵌入行（而不是由多行组成的块），如图 6 所示。为了分配缓存内存，该类设有一个**内存分配器** 参数，可将其设为不同的内存类型。此类还包含标签和状态。**缓存标签** 用于跟踪哪些嵌入行索引位于缓存中，以及哪些缓存条目已占用或空闲。**缓存状态** 根据时间戳跟踪每个缓存条目。动态随机存取存储器缓存和 BYA-SCM 缓存是使用不同缓存内存分配器的缓存类实例。

5.3.2 缓存层级结构。 我们使用动态随机存取存储器和 BYA-SCM 作为可配置的多级缓存。当系统中只有动态随机存取存储器时，缓存配置为单级缓存。当系统中同时有动态随机存取存储器和 BYA-SCM 时，我们将这两种内存组织为两级互斥缓存，其中动态随机存取存储器缓存为第一级，BYA-SCM 缓存为第二级。采用互斥缓存设置可以更高效地利用内存空间。请注意，我们也可以仅使用 BYA-SCM 作为单级缓存，但由于其延迟较高，我们发现这种配置性能不佳。我们根据缓存层级结构确定缓存位置相关操作，例如缓存之间的数据移动以及优先访问哪个缓存。请注意，此处只有动态随机存取存储器缓存和 BYA-SCM 缓存，但该缓存结构能够处理两个以上的缓存，例如在复杂层级结构中使用多个动态随机存取存储器缓存和存储级内存缓存。

5.4 嵌入表管理

此模块接受并响应存储在所有内存类型中的嵌入表请求。对于存放在字节可寻址内存中的表，嵌入表查找和更新由图形处理器管理；对于存放在固态硬盘中的表，则由中央处理器管理。虽然嵌入查找由图形处理器发起，但当索引未命中缓存模块时，查找请求会交由中央处理器处理，以从 RocksDB 嵌入存储中获取数据。请注意，我们可以使用 GPU直接存储（GDS） [32], 从图形处理器直接访问固态硬盘，但为了利用主机端内存（动态随机存取存储器和 BYA-SCM）作为固态硬盘缓存，我们的设计并未采用这种方式。使用 GDS 会将缓存限制在高带宽内存中。如图 6 所示，此模块通过固态硬盘嵌入存储向训练器的其他部分提供两个应用程序接口：GET 和 SET。GET 和 SET 应用程序接口必须同步中央处理器和图形处理器，以保持数据一致性。为了提高效率，我们在中央处理器中为 RocksDB 嵌入存储设计了一个多线程管理单元，可并行查找 RocksDB 分片。该模块还负责在训练开始前初始化嵌入表权重。我们提供了两种初始化选项：

5.4.1 预初始化。 在训练开始前，按照所需分布为存储在所有内存类型中的嵌入表的所有权重赋予随机值。

5.4.2 读取时延迟初始化。 存储在固态硬盘中的嵌入表具有非常大的容量。预先初始化所有这些权重需要很长时间。我们设计了一种延迟初始化技术，以避免在训练开始时执行这一耗时的初始化过程，同时保持固态硬盘的写入耐久性。在该技术中，如果在数据库索引中找不到某个键，我们便会在首次读取时按需初始化嵌入值。在延迟初始化过程中，为了降低初始化延迟，我们使用一个独立的后台线程，根据所需的分布生成一个由随机初始化值组成的队列。这种独立线程优化在单个请求尝试读取大量未初始化行时，尤其有助于降低延迟。当我们尝试读取一个从未被访问过的嵌入行时，便会从队列中取出数值，对所需的嵌入行进行随机初始化。对于模型 1，该技术可将写入量减少 ~15%。

5.5 缓存管理

在我们的设计中，缓存由图形处理器管理，这与 [41]中提出的缓存机制类似。我们在其工作的基础上进行扩展，并在 FBGEMM_GPU 中设计了 GPU 内核，使其支持以层级结构组织多个内存中的缓存，例如动态随机存取存储器和 BYA-SCM。由图形处理器管理缓存，让我们能够利用图形处理器更强的计算能力和带宽，加速缓存管理操作。下面将介绍缓存管理内核。

5.5.1 标签/状态查找。 此 GPU 内核会查找缓存标签和状态，以检查传入的嵌入行请求所涉及的嵌入行是否位于缓存或固态硬盘存储中。在两级缓存中，为了提高效率，该内核会并行查找两个缓存的标签和状态。确定所有缓存中的命中与未命中后，内核会根据内存目标（即动态随机存取存储器、BYA-SCM 或固态硬盘）对传入查找请求的索引进行分组。*5.5.2* 缓存算法。缓存算法内核使用最近最少使用策略。它会查找标签 / 状态查找内核按内存分组的索引。随后据此执行以下操作：1）更新各缓存状态（动态随机存取存储器和 BYA-SCM）中缓存命中索引的访问时间（LRU 状态）；2）为未命中的索引确定要插入的缓存槽位；3）如果没有可用的空闲缓存槽位，则确定要逐出的缓存槽位。我们研究了最近最少使用（LRU）和最不经常使用（LFU）两种缓存算法。我们的实验表明，LRU 的性能比 LFU 高出 ~8-10%。这是因为，尽管 LRU 和 LFU 都能捕捉嵌入表的时间局部性，但采用 LRU 时，在前向传播期间将嵌入行插入缓存后，这些行在反向传播期间仍会是最近使用过的行。因此，即使这些行并非访问频率最高的行，它们在反向传播期间仍留在缓存中的可能性也会增加；与 LFU 相比，这提高了反向传播期间的缓存命中率。

5.5.3 数据查找/更新。 它从查找内核获取分组缓存索引并返回相应的行。它还接受新数据或对缓存中索引的更新，并返回被驱逐的行。

5.5.4 动态随机存取存储器、*BYA-SCM* 与固态硬盘之间的数据移动。在单级缓存中，如果传入的嵌入查找请求命中，发起请求的图形处理器将获取所请求的嵌入行。若未命中，则控制权会交给中央处理器以访问固态硬盘。随后，中央处理器从固态硬盘存储中获取所请求的行。新获取的行将从

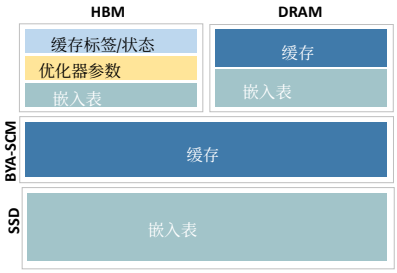


图 7：MTrainS 的内存分配。

固态硬盘插入一级缓存，以便快速访问。如果缓存已满，则会将 LRU行移回固态硬盘以腾出空间存放新行。

对于两级缓存，我们在查找期间并行访问动态随机存取存储器和 BYA-SCM 缓存。两个缓存中的命中数据都会返回到图形处理器。动态随机存取存储器缓存未命中但在 BYA-SCM 缓存中命中的数据，以及从固态硬盘读取的 BYA-SCM 缓存未命中数据，都会提升至动态随机存取存储器缓存，以便快速访问。当动态随机存取存储器容量已满时，会将动态随机存取存储器中 LRU行驱逐至 BYA-SCM 缓存。同样，当 BYA-SCM 缓存已满时，会将 LRU行驱逐回固态硬盘存储。

5.6 内存和图形处理器分配

5.6.1 嵌入表分配。 单个 DLRM 模型包含多个大小和带宽各不相同的嵌入表。我们的系统中有多种容量和带宽各异的内存类型。我们使用动态随机存取存储器和 BYA-SCM 缓存存储在固态硬盘中的低带宽嵌入表，同时可以使用高带宽内存和部分动态随机存取存储器来存放高带宽和中等带宽的嵌入表，并通过缓存满足存储在固态硬盘中的嵌入表的延迟和带宽需求。图 7 展示了 MTrainS 各组件的内存分配。高带宽内存用于存储高带宽嵌入表、优化器参数，以及缓存的标签和状态。动态随机存取存储器用于存储中等带宽的嵌入表，以缓存热点嵌入行；BYA-SCM 则用于存储固态硬盘中各表的较冷嵌入行。我们的实验表明，仅将 BYA-SCM 用于缓存，而不是进行显式分配，效果更好。嵌入表分配的搜索空间非常庞大，因为我们可以将任意表分配到任意类型的内存中。我们使用一个简单的混合整数线性求解器来优化分配。表分配是一个复杂的问题，通过采用更复杂的启发式方法，并考虑局部性等其他因素，我们有可能实现比当前方案更好的表分配。我们将这些复杂设计留待未来的工作中探索。

输入变量：分配的输入变量是模型中每个嵌入表的容量和带宽，以及内存类型的容量和带宽。

约束条件：表分配的约束条件如下：

(1) 每个表只能分配给一种内存类型，但每种内存类型可以容纳多个表。(2) 分配给每种内存类型的表的累计容量不能超过内存容量。

目标函数：根据公式 6 近似计算，最小化嵌入查找总时间。

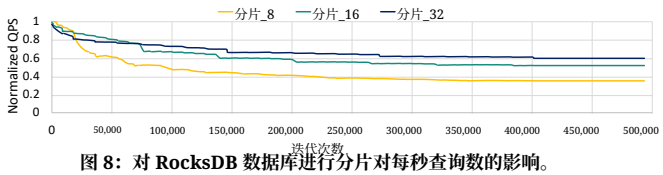


图 8：对 RocksDB 数据库进行分片对每秒查询数的影响。

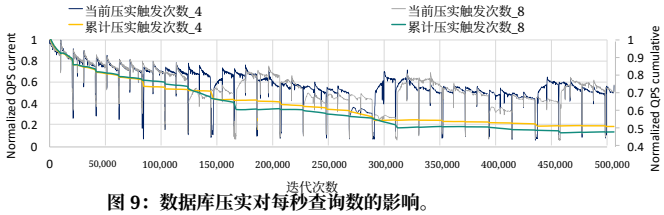


图 9：数据库压实对每秒查询数的影响。

$$lookup_time = Max(time(g)), g \epsilon GPUCount$$
$$time(g) = \sum_{M} \sum_{T_{gm}} (D \times L \times p)/BW_{g_m} \quad (6)$$

公式 6 中，M 表示内存类型（例如高带宽内存或动态随机存取存储器）。T_{g_m} 表示分配给某个图形处理器上特定内存类型的嵌入表，BW_{g_m} 表示分片 g 上内存类型 m 的带宽。例如，对于高带宽内存，BW_{g_m} 表示高带宽内存带宽。对于共享的动态随机存取存储器，它表示 $DRAM_BW/^{num_{gpus}}$。我们在第 7.6 节展示不同放置方式的影响。

5.6.2 GPU 分配。基于表分配，对于 N 个 GPU，分配到 GPU_i 的高带宽内存中的嵌入表将由 GPU_i 处理。此外，表放置算法将分配到动态随机存取存储器和固态硬盘的嵌入表分配给 N 个 GPU 管理，以最小化公式 6 中的查找时间。

5.7 流水线处理

为了隐藏缓存未命中时访问固态硬盘的部分延迟，我们可以提前数个批次对缓存的访问进行流水线处理。我们不再按顺序执行 1) 获取、2) 预处理、3) 在图形处理器上加载、4) 训练，而是将每个步骤拆分为独立阶段，并针对不同批次同时执行。在我们的案例中，我们增加了一个步骤：4a) 在训练前将稀疏索引预取到缓存中。只要我们能够维持这样一个不变量：预取到缓存中的嵌入行在该批次完成训练之前不会被驱逐，就可以让任意数量的批次进入流水线。在 4a) 预取和 4) 训练之间增加额外阶段，可以不断增强流水线隐藏延迟的能力，直到其超过 GET 调用的典型固态硬盘延迟。如果满足每秒查询数目标所需的带宽超过固态硬盘的能力，那么增加再多额外阶段也无济于事。

5.8 MTrainS 配置与元数据

5.8.1 MTrainS 元数据。元数据记录所有嵌入表的内存和图形处理器分配，并在每次嵌入表查找时用于将请求定向到相应的内存。

5.8.2 缓存配置。这包括缓存配置，可将动态随机存取存储器和 BYA-SCM 存储级内存分层呈现，或单独用作

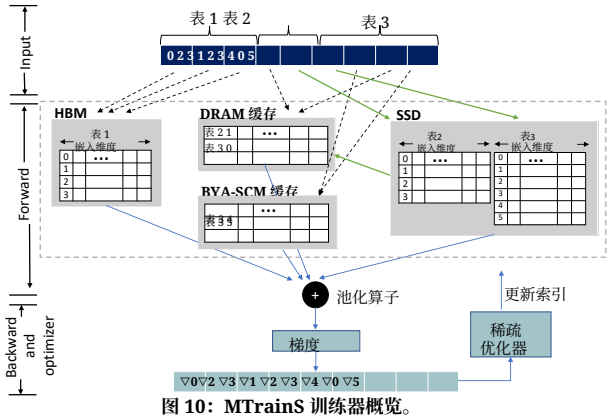


图 10：MTrainS 训练器概览。

第一层缓存，并设置缓存行数和列数，以适配目标 DLRM 模型的嵌入维度。

5.8.3 RocksDB 配置。RocksDB 嵌入存储提供了多种可调参数，用于优化性能。这些参数包括用于嵌入表查找的 CPU 线程数、数据库分片数、压实时间、内存表大小，以及是否启用块缓存。分片是最重要的参数之一，它能提高键查找效率并缩短压实时间。如图 8 所示，对数据库进行分片可使每秒查询数最多提高 40%。另一个参数是数据库压实，它对于 RocksDB 在训练期间将数据库容量维持在合理范围内是必不可少的。所有 RocksDB 分片和训练器同步进行数据库压实，会引发严重的惊群问题，导致内存和输入输出峰值大幅升高。我们观察到，在数据库压实期间，每秒查询数会显著下降（某些情况下超过 50%；见图 9）。调整压实参数（例如图 9 所示的压实触发时间）可使累计每秒查询数提升 5%–8%。在实验中，我们展示了为每个模型找到的最佳 RocksDB 配置所取得的结果。

5.9 端到端训练器

图 10 展示了端到端训练器。我们首先执行嵌入表分配，并根据最优放置将嵌入表分配到高带宽内存、动态随机存取存储器和固态硬盘中。请注意，除非模型发生重大变化，否则无需在每次训练时都执行放置。对于训练数据，稠密特征会沿批次维度分布到多个 GPU 上。模型的嵌入表按表进行分配，分布到各个 GPU；每个 GPU 都会负责查找分配给它的嵌入表。对于当前的工作负载，按表分区已能为各 GPU 提供足够的模型并行性。随后，在嵌入查找期间，MTrainS 会将传入的输入索引分配到各个 GPU 和内存中。例如，在图 10 中，表 1 放置在高带宽内存中，表 2 和表 3 放置在固态硬盘中。在前向传播期间，GPU 使用 MTrainS 根据索引发起嵌入表查找。例如，图中的 MTrainS 会从动态随机存取存储器中获取表 2 的索引 1 对应的数据。GPU 从内存中获取嵌入行后，会执行聚合操作和优化器操作，然后在反向传播期间更新相应内存中的权重。

6 系统设置与实现

6.1 软件设计与实现

我们使用 PyTorch 版本的 DLRM，并实现了一种新的 EmbeddingBag，用于嵌入表操作，可使用异构内存和存储，而不再使用 DLRM 中由 PyTorch 实现的默认 EmbeddingBag [6]。我们的新 EmbeddingBag 集成了我们开发的 FBGEMM_GPU 内核，用于管理缓存和内存。图 11 展示了软件概览。

在我们的实验中，我们使用 IPMCTL [18] 将英特尔 DCPMM 配置为应用直连模式。我们使用 Linux 内核 5.4，它支持以易失性方式使用 DCPMM。随后，我们使用 NDCTL 6.7 [4] 工具将存储级内存配置为 devdax 模式。该模式可直接访问 DCPMM，速度快于基于文件系统的访问。我们使用 DAXCTL [4] 将 DCPMM 配置为 system-ram 模式，使其能够作为独立的易失性内存 NUMA 节点使用。为了在 DLRM 中实现存储级内存，我们使用 Memkind 库 [3]。Memkind 会在应用空间内将堆划分为多种内存类型，例如动态随机存取存储器和存储级内存。我们使用 MEM KIND DAX KMEM 访问存储级内存。– – –

我们使用 CUDA11 实现 GPU 内核。如图所示，我们使用 cudaMallocManaged [5]，通过统一内存系统，让 GPU 透明地访问高带宽内存和动态随机存取存储器。对于 BYA-SCM，由于 GPU 无法通过统一内存直接访问这类内存，我们使用 cudaHostRegister [5]，将 Memkind 库预先分配的现有主机内存区域注册到 CUDA。我们实现了多个缓存管理内核和计算内核。借助 PyTorch 的 torch.cuda.Stream[34], ，我们在不同的流中启动可并行运行的内核，例如查找标签，或从动态随机存取存储器和 BYA-SCM 缓存中获取数据。随后，我们使用 torch.cuda.synchronize [35]，在计算内核启动前同步从缓存获取数据的内核。在处理一个批次的计算时，我们会并行更新标签/状态，并管理所有缓存和固态硬盘存储中的插入、更新和驱逐。随后，这些内核会与后续批次的数据查找操作同步，以便为下一批次获取更新后的缓存/数据。

6.2 工作负载描述与设置

在实验中，我们使用了两个源自真实应用场景、最具代表性的 DLRM 模型（模型 1 和模型 2 ）。这些模型展现了我们大多数模型所具备的不同特性（容量受限和内存受限）。我们还使用了一个容量为模型 1 两倍的模型（模型 1+），以代表模型在未来两年内的增长情况。表 2 展示了这些模型的特性。

6.3 评估硬件描述

在评估中，我们使用了一台基于英特尔 Barlow Pass 的系统，配备 2 个安装了英特尔 Ice Lake 处理器的 CPU 插槽和 8 个 Nvidia A100 GPU。该平台专为人工智能／机器学习、深度学习和高性能计算应用而设计，配备 8 个 Nvidia A100 GPU，并通过全互连的 NVLink / NVSwitch [21] fabric 互联。硬件规格如表 3 所示。

6.4 服务器设计

我们通过改变内存和存储的类型与容量，考察了多种服务器设计。表4总结了我们在实验中使用的配置。在所有系统配置中，我们

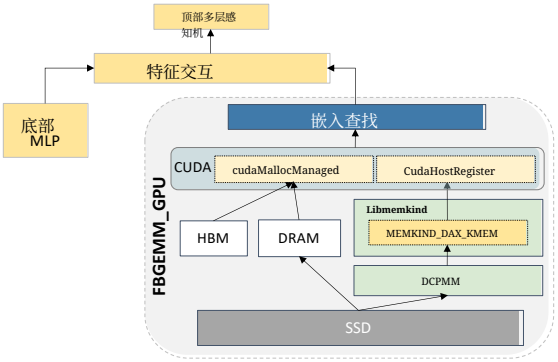


图 11：MTrainS 软件设计。

规格	系统配置
操作系统 /Linux 内核版本	CentOS-8 /5.4.135
CPU 型号	Intel(R) Xeon(R) Gold 6348 CPU @ 2.60GHz
插槽 /每插槽核心数 /总线程数	2 /28 /112
L1I/L1D 缓存/L2 缓存/L3 缓存	32 KB/48 KB/1.25 MB/42 MB
GPU 型号/ GPU 数量	A100-SXM4-40GB / 8
GPU 驱动程序/CUDA 版本	450.142.00/ CUDA 11
高带宽内存容量	HBM2e 320 GB（40 × 8）
动态随机存取存储器容量	DDR4 384 GB（32 GB × 12 个 DIMM插槽）
存储级内存	DDR-T 2 TB（128 GB × 16 个 DIMM插槽）
容量	

表 3：系统配置：硬件规格。

配置。	HBM	DRAM	BYA-SCM	BLA-SCM	Nand SSD
基线	320	384			
configNand	320	384			8192
configBLA	320	384		2048	
configBYA-1	320	384	384		8192
configBYA-2	320	384	768		8192
configSCM	320	384	384	2048	

表4：系统配置（所有容量均以 GB 为单位）。

将动态随机存取存储器的容量限制为384GB。我们使用两种容量的BYA-SCM：384GB和768GB。选择这两种容量，是为了确定系统所需的动态随机存取存储器与BYA-SCM之比。我们也测试了其他配置，例如增加BYA-SCM的容量。然而，对于现有工作负载，将BYA-SCM容量增加到768GB以上并无额外收益，因为此配置下我们受计算能力限制。对于容量为768GB的BLA-SCM配置，情况也类似。

我们将系统中一半的动态随机存取存储器（192 GB）用于缓存，其余部分用于存储容量较小的嵌入表，以及满足 DLRM 的其他系统需求。我们将所有 BYA-SCM 用于缓存，即在 384GB 配置中使用 360GB，在 768GB 配置中使用720GB。剩余的 BYA-SCM 用于存储 Optane 元数据。

7 评估

7.1 基线

我们将 MTrainS 与表4所示的基线系统配置进行比较。基线系统中的实现同样整合了 DLRM 和 FBGEMM GPU。基线系统使用高带宽内存和动态随机存取存储器来存储嵌入并进行缓存。该实现采用了与 CDLRM 相同的技术 [9]。CDLRM 使用中央处理器管理缓存；为了与使用图形处理器进行快速缓存管理的 MTrainS 公平比较，我们将 CDLRM 中的缓存管理转移到图形处理器上。我们将此版本称为 CDLRM+。此外，我们还

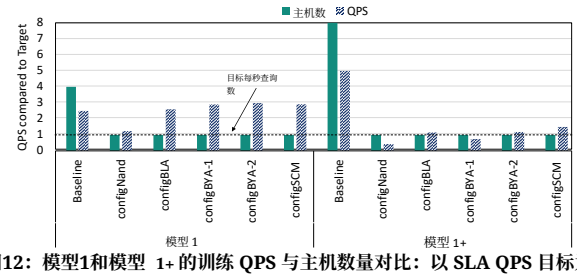


图12：模型1和模型 1+ 的训练 QPS 与主机数量对比：以 SLA QPS 目标为基线，比较 CDLRM+ 和 MTrainS

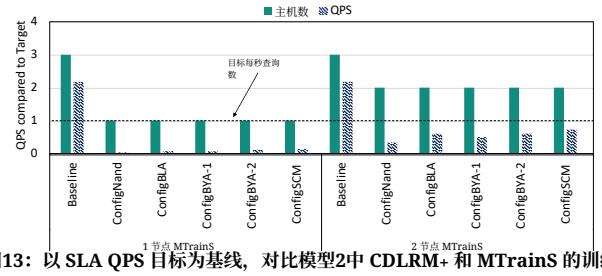


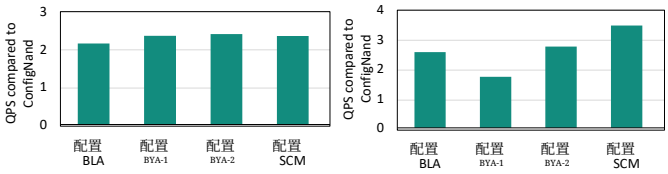
图13：以 SLA QPS 目标为基线，对比模型2中 CDLRM+ 和 MTrainS 的训练 QPS 与主机数量。

在 CDLRM+ 中加入了高效的嵌入表放置方案。我们的实验将训练运行 CDLRM+ 的模型所需的节点数量，与单个节点运行 MTrainS 时的性能进行比较。为了比较性能，我们采用目标每秒查询数（QPS）——即数据中心可接受的 QPS——来训练特定模型；目标值取决于模型的训练频率以及新训练数据的产生速率。实验中的 QPS 表示每秒可用于训练模型的输入数据量，其中包括前向和反向传播。它与训练时间成反比。

7.2 训练效率

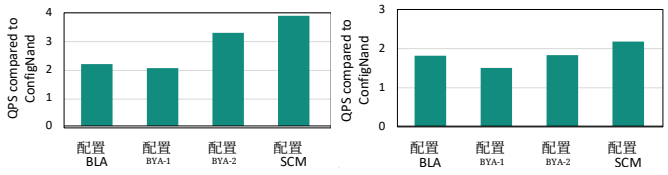
采用更高密度内存技术的一个主要理由是，在训练特定模型时减少节点数量，并提高功耗和成本效益。因此，我们首先根据目标 QPS，对比 MTrainS 和 CDLRM+ 在模型 1 和模型2上的整体部署效率。

图12展示了模型 1 和模型 1+ 的效率对比，这两个模型都受内存容量限制。请注意，模型 1+ 是模型 1 面向未来扩展后的版本，容量更大（2×）。使用基准配置 CDLRM+时，这些模型需要 4-8 台主机才能加载和训练。借助 MTrainS，一台主机即可提供足够的扩展内存容量来加载和训练模型。如图12所示，在这种配置下，我们也能达到目标每秒查询数（服务级别协议）。虽然不使用 MTrainS 的基线方案能够实现更高的每秒查询数，但它需要多用 4-8×台主机，因此这部分性能被闲置，无法为部署带来额外的效率，因为我们已经满足了训练 QPS 的要求。因此，MTrainS 在提供服务级别协议所要求的性能的同时，可以提高模型 1 和模型 1+ 的功耗和成本效率。我们的实验表明，扩展到超过模型 1+规模的模型需要进行多节点训练，每个节点都利用 MTrainS。尽管如此，与基线方案相比，MTrainS 仍能减少主机数量，因为它可以在每台主机上存储更大的模型。



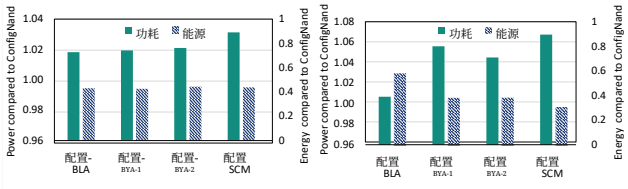
(a)模型 1　　(b)模型 1+

图14：以 NAND闪存固态硬盘为基线时，MTrainS 在模型 1 和模型 1+ 上不同配置的每秒查询数对比。



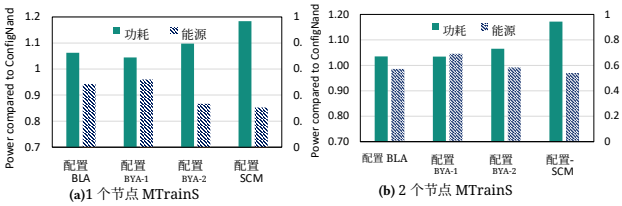
(a)1 节点 MTrainS　　(b)2 节点 MTrainS

图15：以 NAND闪存固态硬盘为基线时，MTrainS 在模型2上不同配置的每秒查询数对比。



(a)模型 1　　(b)模型 1+

图16：平均功耗和总能耗。



(a)1 个节点 MTrainS　　(b) 2 个节点 MTrainS

图17：平均功耗和总能耗。

图13展示了模型 2（带宽受限工作负载）使用 MTrainS 与 CDLRM+ 的对比。使用 CDLRM+的基准配置时，该模型需要 3 台主机才能加载和训练。MTrainS 虽然可以让模型在一台主机上加载和训练，但由于模型需要更高的内存带宽，无法满足每秒查询数要求。图中还显示，使用 2 个节点且每个节点都运行 MTrainS，与单节点相比，能显著提升模型 2 的性能。尽管如此，这 2 个节点仍无法达到每秒查询数目标，因为即使有 2 个节点，系统中存储级内存所提供的额外带宽也不足以满足模型 2 的高带宽需求。不过，这种能力可以提升研发效率，适用于没有高每秒查询数生产需求的场景。

接下来，我们评估各种 MTrainS 系统配置相较于 NAND闪存固态硬盘的性能，以了解 BYA-SCM 和 BLA-SCM 相较于 NAND闪存固态硬盘所带来的性能影响。在图14a 和图14b 中，我们比较了不同 MTrainS 配置下模型 1 和模型 1+ 所达到的每秒查询数。如图14a 所示，对于模型 1，使用 BLA-SCM（configBLA）替代 NAND闪存固态硬盘（configNand）可使每秒查询数提高 2×。将 BYA-SCM 与 NAND闪存固态硬盘结合使用（configBYA-1 和 configBYA-2）会增加缓存容量，从而降低发送到 NAND闪存固态硬盘的流量，因此可提供 2.4× 每秒查询数。我们的结果表明，对于同时使用 BYA-SCM 和 BLA-SCM 的 configSCM 配置，每秒查询数不会进一步提升，因为使用 BYA-SCM 增加每台主机的缓存容量后，对固态硬盘的访问已不再处于关键路径上。

对于图14b所示的模型 1+，配备384GB BYA-SCM（configBYA-1）的NAND闪存固态硬盘可将每秒查询数提高1.73×，而768GB BYA-SCM（configBYA-2）可将其提高2.76×，这是由于缓存命中率提升所致。当我们同时使用BYA-SCM和BLA-SCM（配置 SCM）时，可达到 3× 每秒查询数。模型 1+ 由于嵌入维度增加而具有更高的带宽需求。使用NAND闪存固态硬盘时，每次输入输出已经读取4KB数据块，大于嵌入维度。然而，在缓存容量固定的情况下，嵌入维度增加会导致缓存（动态随机存取存储器或BYA-SCM）中可维护的嵌入行数减少。这会提高未命中率，因此
这会增加对固态硬盘的输入输出。BLA-SCM 具有更高的每秒输入输出操作数，能够支持这种输入输出量的增加，因此每秒查询数更高。

在图15a中，对于使用单个节点的 MTrainS 模型2，由于 BLA-SCM 的带宽更高、延迟低于 configNand 中的 NAND 闪存固态硬盘，性能提升了 2.2×。加入 BYA-SCM 后，由于缓存命中率提高，性能又提升了 2× 和 3.2×。与模型1类似，BYA-SCM 与 BLA-SCM 的组合性能最佳。在图15b中，模型分布在2个节点上，每个节点上的模型容量是使用单个节点时的一半。使用2个节点时，性能提升趋势与使用单个节点时相同，但加速比有所下降，因为每个节点上的模型容量减少后，固态硬盘流量也随之减少，因此缓存的影响不如使用单个节点时显著。尽管与 configNand 相比，各种配置都提升了模型2的性能，但如图13所示，对于内存带宽受限工作负载，使用存储级内存无法达到所需的性能（每秒查询数目标）。

7.3 功耗与能源分析

在图16a和图16b中，我们展示了模型 1 和模型 1+ 的各种 MTrainS 配置的功耗和能耗，以研究在系统中引入 BYA-SCM 和 BLA-SCM 后功耗的增加以及总能耗的变化。添加 BYA-SCM 和 BLA-SCM 仅使平台的总体功耗增加 1-3.2%。这是因为这些单元各自的功耗较低，而主要的功耗来源是图形处理器、中央处理器和动态随机存取存储器。考虑到执行时间缩短以及总体能耗降低，这种每个节点增加的额外功耗是合理的（$Energy = Power \times Time$）。与 configNand 相比，我们观察到两个模型的能耗降低了 60% 至 70%。图17a和图17b展示了模型2中 MTrainS 的功耗和能耗。在这种情况下，添加存储级内存会使功耗增加 3-18%。模型2的功耗较高，是因为与模型1相比，其数据访问量更大，因此缓存和嵌入存储操作更多，从而增加了 BYA-SCM 和 BLA-SCM 的功耗。

图18展示了使用 MTrainS 训练模型 1 和模型1+ 时的功耗与能耗，并将其与仅使用高带宽内存和动态随机存取存储器的基准配置进行了比较。与基线训练相比，后者

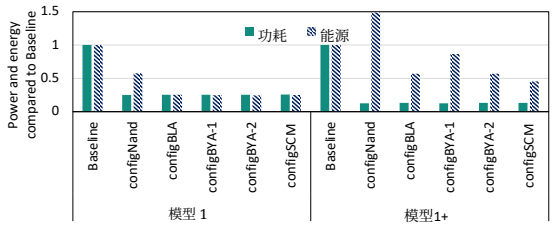


图18：MTrainS 与运行 CLDRM+ 的基线系统在模型 1 和模型 1+ 上的功耗与能源对比。

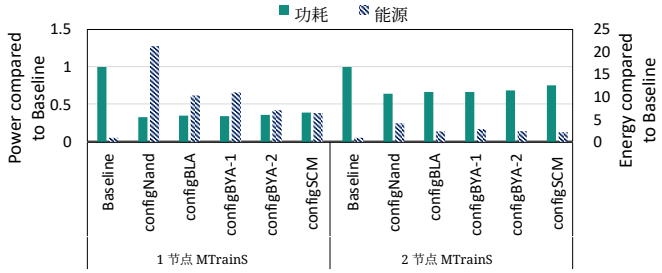


图19：MTrainS 与运行 CLDRM+ 的基线系统在模型2上的功耗与能源对比，以及 MTrainS 使用 1 个和 2 个节点时的对比。

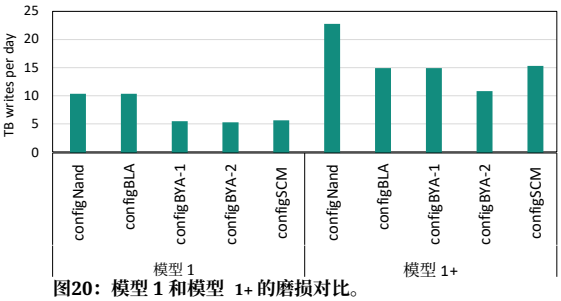
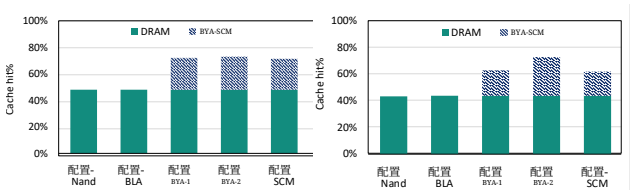


图20：模型 1 和模型 1+ 的磨损对比。

需要 4-8 个节点来容纳这些模型，而使用 MTrainS 可使功耗降低 ~ 1/4 - 1/8。这主要是因为使用 MTrainS 训练所需的节点数量减少了，因此能耗最多可降低 50%。不过，如图所示，对于模型 1+，NAND 闪存固态硬盘的能耗高于基线系统，这是因为每秒查询数较低（训练时间较长）。请注意，这种大幅降低是因为训练设有每秒查询数目标。如果比较基线系统和 MTrainS 实际达到的性能，那么运行 MTrainS 所需的节点数会更多，从而降低功耗和能耗的降幅。图19显示，在 1 节点和 2 节点的 MTrainS 配置下，模型 2 的平均功耗最多分别降低 60% 和 30%。然而，由于 MTrainS 训练模型2的每秒查询数远低于基准配置（训练时间更长），即使使用 2 个节点运行 MTrainS，其能耗在所有配置下仍高于基线，如图19所示。

7.4 存储耐久性与磨损

在本节及后续各节中，我们将详细分析从 MTrainS 中受益的模型的硬件特性，即



(a)模型 1　　(b)模型 1+

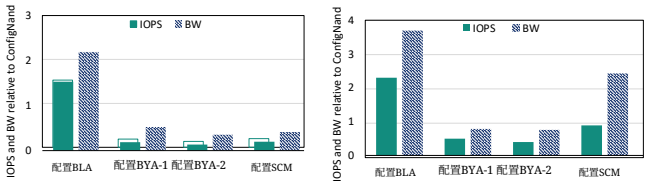
图21：不同 MTrainS 配置下模型 1 (a) 和模型 1+ (b) 的缓存命中率。

模型 1 和模型 1+。图20测量了模型 1 和模型 1+ 每日写入的 TB 数，以研究 MTrainS 的各种配置是否满足我们的耐久性要求。根据 BLA-SCM 和 NAND闪存固态硬盘的容量，为避免存储磨损，其耐久性要求分别为每日数据写入量 200TB 和 8TB（每日整盘写入次数）。如图所示，虽然 configNand 能让模型 1 达到每秒查询数目标，但其每日写入量超过了 NAND闪存固态硬盘的耐久性上限（8TB）。configBYA-1 和 configBYA-2 通过增大缓存容量来减少写入存储的数据量，因此满足耐久性要求。采用 BLA-SCM 的所有配置（configBLA 和 configSCM）都满足耐久性要求，因为其每日整盘写入次数更高（200TB）。对于模型 1+，所有采用 NAND闪存固态硬盘的配置都无法满足耐久性要求。因此，对于未来的模型，BLA-SCM 是防止存储磨损的最佳选择。

7.5 缓存命中与输入输出利用率

图21展示了模型 1 和模型 1+ 的缓存命中率。对于图21a中的模型 1，我们观察到，使用 NAND闪存固态硬盘和仅使用动态随机存取存储器作为缓存的 configBLA 时，缓存命中率为 50%；加入 BYA-SCM（configBYA-1）后，缓存命中率提高至 75%。768GB 的 BYA-SCM 配置（configBYA-2）并未进一步提高缓存命中率，因为 384GB 的 BYA-SCM 已足以捕获时间局部性。因此，如图14a所示，BYA-SCM 的两种配置实现了相近的每秒查询数。对于模型1+，如图21b所示，我们测得 configNand 的缓存命中率为 40%。加入 BYA-SCM 后，configBYA-1 和 configBYA-2 的缓存命中率分别提高至 60% 和 70%。在这种情况下，如图14b所示，更大容量的 BYA-SCM 带来了更高的每秒查询数。模型 1 和模型 1+ 表现不同，是因为后者具有更大的嵌入维度，对缓存造成了更大的压力。因此，容量更大的缓存能够捕获更多局部性，从而减少输入输出流量并提高每秒查询数。

如图22所示，使用 BLA-SCM 时，训练器可实现更高的每秒输入输出操作数和带宽。图22a显示，对于采用 BLA-SCM 1 的模型 1，测得的每秒输入输出操作数增加了 1.5×，有效带宽增加了 2.3×（$IOPS \times embedding_dim$）；如图14a所示，与 NAND闪存固态硬盘相比，每秒查询数也相应提高了超过 2×。加入 BYA-SCM 后，每秒输入输出操作数最多降低 80%，带宽最多降低 60%。此时，输入输出流量大幅减少，固态硬盘流量已不再处于性能关键路径上。图22b显示，模型1+ 也呈现类似趋势，主要区别在于，增加 BYA-SCM 后每秒查询数仍持续提升，从而提高每秒输入输出操作数和有效带宽。总之，表4中的 MTrainS 配置通过以下方式提高每秒查询数：1）使用 BYA-SCM 减少输入输出流量；和/或 2）使用 BLA-SCM 提供更多每秒输入输出操作数。



(a)模型 1 (b) 模型 1+

图22：模型 1 和模型 1+ 的每秒输入输出操作数及有效固态硬盘带宽。

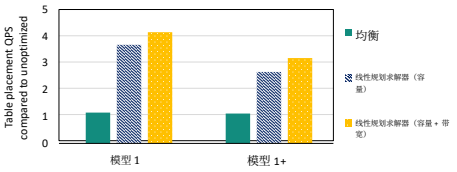


图 23：模型 1 和模型 1+ 的不同嵌入表放置方式带来的每秒查询数提升

7.6 嵌入表分配效率

本节比较模型 1 和模型 1+ 在资源最充足的配置 configBYA-2 中运行时，采用四种嵌入表放置方案所带来的收益。第一种方案是将所有嵌入表放在 RocksDB 存储中（NAND闪存固态硬盘和 BLA-SCM），并使用可用的动态随机存取存储器和 BYA-SCM 容量进行缓存。我们将其作为图 23 中的基线，并称其为未优化方案，因为表的放置仅针对最大化固态硬盘容量的利用率进行优化，而未考虑带宽；因此，各图形处理器之间的数据访问量并不均衡。将未优化方案与利用带宽平衡各图形处理器数据访问的方案进行比较后，我们发现每秒查询数提高了 15%。请注意，仅使用固态硬盘，并辅以动态随机存取存储器和 BYA-SCM 缓存，会导致高带宽内存利用率较低。接下来，我们将其与使用线性规划求解器进行的方案比较，该求解器仅以嵌入表的容量和内存类型为输入。采用这种放置方案，每秒查询数提高了 2.5-3.5× 。同时考虑容量和带宽的放置方案可进一步将每秒查询数提高到 3.2-4.2×。因此，在使用异构内存训练 DLRM 时，表的放置至关重要；同时考虑容量和带宽可获得最佳性能。

8 相关工作

推荐系统的内存容量扩展：推荐系统中嵌入操作对高内存容量和带宽的需求给内存系统带来了挑战。训练和推理用例各自面临独特的挑战。例如，在推理过程中，每个查询的延迟都很重要。Ardestani 等人 [7] 提出了一个端到端系统，以利用固态硬盘，同时将延迟控制在可接受范围内；Wilkening 等人 [38] 利用固态硬盘中的控制器，将部分计算卸载到更靠近数据的位置。Eisenman 等人 [12] 使用存储级内存来增加每台主机的内存容量。然而，这需要对嵌入表进行离线预处理，因此不适用于训练。

训练对延迟不太敏感，但需要更高的带宽和频繁的参数更新，因此会产生大量固态硬盘的读写流量。Zhao 等人 [42] 提出了一种利用高带宽内存、动态随机存取存储器和固态硬盘的训练系统。他们利用流水线处理来隐藏固态硬盘延迟，并使用缓存来隐藏较低的固态硬盘带宽。他们采用参数服务器方案来训练大型模型，而不是我们的分布式同步方案。此外，与之不同的是，为了维持高吞吐量，

我们利用图形处理器进行缓存管理。此外，我们展示了不同 SCM 技术的性能和功耗影响对训练的影响。Balasubramanian 等人 [9]实现了 CPU 管理的缓存，以利用高带宽内存和动态随机存取存储器扩展内存容量嵌入表，但仅扩展到动态随机存取存储器。Yang 等人 [41] 提出并实现了一种由 CPU 内存支持的图形处理器嵌入软件缓存方案。我们基于与 [41]类似的方案，启用由动态随机存取存储器和 BYA-SCM（由固态硬盘支持）组成的多级缓存。然而，尽管他们也在 DLRM 中使用缓存，但他们专注于使用不同的精度来减少嵌入存储容量。相比之下，我们将内存层次结构扩展到 BYA-SCM 和闪存，以扩展每台主机的内存容量，同时不影响训练精度。

面向人工智能工作负载的存储级内存使用：Hildebrand 等人 [16] 开发了一种基于整数线性规划的系统，可在动态随机存取存储器和存储级内存之间移动张量。他们表明，经过优化的数据放置比朴素方法能够实现更高的性能。同样，在我们的设计中，我们根据应用程序的特性，谨慎地将数据放置在不同的内存中。

9 结论

为了提升模型质量，推荐模型的复杂度、容量和训练数据不断增加，这给底层平台的计算、带宽、输入输出和内存容量带来了相当大的压力。本文针对内存容量压力，提出一种基于分层内存的训练器 MTrainS，以提高每台主机的内存容量。我们定量比较了字节和块可寻址存储级内存的性能及其系统级影响。根据大规模生产工作负载中嵌入表的时间局部性，我们采用多级缓存，并利用存储级内存，将加载和训练内存受限模型所需的节点数减少至原来的 1/4–1/8。由于图形处理器等其他组件的功耗占主导，在平台中增加存储级内存只会使平台成本和功耗略微上升。因此，由于训练模型所需的节点更少，我们观察到整体效率有所提升。对于带宽受限模型，这种方法存在局限。在这些情况下，在所评估的硬件配置下，多级缓存无法提供足够的带宽来满足模型的性能要求。

参考文献

[1] Compute Express LinkTM：突破性的 CPU 到设备互连技术。2020年。https://www.computeexpresslink.org/。[2] FBGEMM_GPU。2021年。https://github.com/pytorch/FBGEMM/tree/main/ fbgemm_gpu。[3] Memkind 库。2021年。https://github.com/memkind/memkind。[4] Ndctl 和 DAXCTL。2021年。https://github.com/pmem/ndctl。[5] CUDA C++ 编程指南。2022年。https://docs.nvidia.com/cuda/cuda-c- programming-guide/index.html。[6] EmbeddingBag。2022年。https://pytorch.org/docs/stable/generated/torch.nn.EmbeddingBag.html。[7] Ehsan K. Ardestani、Changkyu Kim、Seung Jae Lee、Luoshang Pan、Valmiki Rampersad、Jens Axboe、Banit Agrawal、Fuxun Yu、Ansha Yu、Trung Le、Hector Yuen、

Shishir Juluri、Akshat Nanda、Manoj Wodekar、Dheevatsa Mudigere、Krishnaku- mar Nair、Maxim Naumov、Chris Peterson、Mikhail Smelyanskiy 和 Vijay Rao。通过软件定义内存支持大规模 DLRM 推理。CoRR，abs/2110.11489，2021。https://arxiv.org/abs/2110.11489。[8] Jens Axboe。Fio 基准测试，2013。https://github.com/axboe/fio。[9] Keshav Balasubramanian、Abdulla Alshabanah、Joshua D Choe 和 Murali An- navaram。cdlrm：面向可扩展推荐模型训练的前瞻缓存。载于第十五届 ACM 推荐系统大会，第 263 272 页，2021 – 。[10] Heng-Tze Cheng、Levent Koc、Jeremiah Harmsen、Tal Shaked、Tushar Chandra、Hrishi Aradhye、Glen Anderson、Greg Corrado、Wei Chai、Mustafa Ispir、Rohan Anil、Zakaria Haque、Lichan Hong、Vihan Jain、Xiaobing Liu 和 Hemal Shah。用于推荐系统的宽深度学习。arXiv:1606.07792，2016。http: //arxiv.org/abs/1606.07792。[11] C. K. Chow。存储层次结构的优化。IBM 研究与开发期刊，18(3):194–203，1974。[12] Assaf Eisenman、Maxim Naumov、Darryl Gardner、Misha Smelyanskiy、Sergey Pupyrev、Kim Hazelwood、Asaf Cidon 和 Sachin Katti。Bandana：使用非易失性内存存储深度学习模型。机器学习与系统论文集，1:40–52，2019。[13] Ali Mamdouh Elkahky、Yang Song 和 Xiaodong He。推荐系统中的跨域用户建模的多视图深度学习方法。载于第 24 届万维网国际会议论文集，WWW '15。国际万维网会议指导委员会，2015。[14] Facebook。MultiGet 性能。2021。https://github.com/facebook/rocksdb/wiki/MultiGet-Performance。[15] Facebook。RocksDB。2021。https://rocksdb.org/。[16] Mark Hildebrand、Jawad Khan、Sanjeev Trika、Jason Lowe-Power 和 Venkatesh Akella。AutoTM：使用整数线性规划在异构内存系统中实现自动张量迁移。载于第二十五届编程语言与操作系统架构支持大会论文集，ASPLOS '20，第 875–890 页，美国纽约州纽约市，2020 年。计算机协会。[17] 英特尔。英特尔内存延迟检测器 v3.9a。2021。https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html。[18] 英特尔。IPMCTL。2021。https://github.com/intel/ipmctl。[19] 英特尔。Intel® OptaneTM 持久内存。2022。https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html。[20] 英特尔。面向高要求存储工作负载的 Intel® OptaneTM SSD 905P 系列。2022。https://www.intel.com/content/www/us/en/products/docs/memory-storage/solid-state-drives/enthusiast-ssds/optane-ssd-905p-brief.html。[21] Alexander Ishii 和 Denis Foley。交换结构与 FPGA 架构。载于 Hot Chips：高性能芯片研讨会（HC30，2018），2018。https://old.hotchips.org/hc30/2conf/ 2.01_NVIDIA_NVSwitch_HotChips2018_DGX2NVS_Final.pdf。[22] Joseph Izraelevitz、Jian Yang、Lu Zhang、Juno Kim、Xiao Liu、Amir Saman Memaripour、Yun Joon Soh、Zixuan Wang、Yi Xu、Subramanya R. Dulloor、Jishen Zhao 和 Steven Swanson。英特尔傲腾 DC 持久内存模块的基本性能测量。CoRR，abs/1903.05714，2019。http: //arxiv.org/abs/1903.05714。[23] Norman P Jouppi、Doe Hyun Yoon、Matthew Ashcraft、Mark Gottscho、Thomas B Jablin、George Kurian、James Laudon、Sheng Li、Peter Ma、Xiaoyu Ma 等。塑造谷歌 TPUv4i 工业产品的三代产品带来的十条经验。载于 2021 年 ACM/IEEE 第 48 届计算机体系结构年度国际研讨会（ISCA），第 1–14 页。IEEE，2021。[24] Hiwot Tadese Kassa、Jason Akers、Mrinmoy Ghosh、Zhichao Cao、Vaibhav Gogte 和 Ronald Dreslinski。使用存储级内存作为易失性内存扩展，提升基于闪存的键值存储性能。载于 2021 年 USENIX 年度技术会议（USENIX ATC 21），第 821–837 页。USENIX 协会，2021 年 7 月。[25] Daya Khudia、Jianyu Huang、Protonu Basu、Summer Deng、Haixin Liu、Jong- soo Park 和 Mikhail Smelyanskiy。FBGEMM：实现高性能低精度深度学习推理。arXiv 预印本 arXiv:2101.05615，2021。[26] Elias Konstantinidis 和 Yiannis Cotronis。GPU 快速片上内存的定量性能评估。载于 2016 年第 24 届 Euromicro 并行、分布式与基于网络处理国际会议（PDP），第 448–455 页，2016。[27] Agnes Lydia 和 Sagayaraj Francis。AdaGrad——一种用于随机梯度下降的优化器。Int. J. Inf. Comput. Sci，6(5)，2019。[28] Dheevatsa Mudigere、Yuchen Hao、Jianyu Huang、Zhihao Jia、Andrew Tulloch、Srinivas Sridharan、Xing Liu、Mustafa Ozdal、Jade Nie、Jongsoo Park、Liang Luo、Jie Amy Yang、Leon Gao、Dmytro Ivchenko、Aarti Basant、Yuxi Hu、Jiyan Yang、Ehsan K. Ardestani、Xiaodong Wang、Rakesh Komuravelli、Ching-Hsiang Chu、Serhat Yilmaz、Huayu Li、Jiyuan Qian、Zhuobo Feng、Yinbin Ma、Junjie Yang、Ellie Wen、Hong Li、Lin Yang、Chonglin Sun、Whitney Zhao、Dimitry Melts、Krishna Dhulipala、KR Kishore、Tyler Graf、Assaf Eisenman、Kiran Kumar Matam、Adi Gangidi、Guoqiang Jerry Chen、Manoj Krishnan、Avinash Nayak、Krishnaku- mar Nair、Bharath Muthiah、Mahmoud khorashadi、Pallab Bhattacharya、Petr Lapukhov、Maxim Naumov、Ajit Mathews、Lin Qiao、Mikhail Smelyanskiy、Bill Jia 和 Vijay Rao。面向深度学习推荐模型快速、可扩展训练的软硬件协同设计。2021 年。https://arxiv.org/abs/2104.05158.[29] Maxim Naumov、John Kim、Dheevatsa Mudigere、Srinivas Sridharan、Xiaodong Wang、Whitney Zhao、Serhat Yilmaz、Changkyu Kim、Hector Yuen、Mustafa Ozdal 等。Facebook 数据中心的深度学习训练：横向扩展系统的设计。arXiv 预印本 arXiv:2003.09518，2020 年。[30] Maxim Naumov、Dheevatsa Mudigere、Hao-Jun Michael Shi、Jianyu Huang、Narayanan Sundaraman、Jongsoo Park、Xiaodong Wang、Udit Gupta、Carole-Jean Wu、Alisson G. Azzolini、Dmytro Dzhulgakov、Andrey Mallevich、Ilia Cherni- avskii、Yinghai Lu、Raghuraman Krishnamoorthi、Ansha Yu、Volodymyr Kon- dratenko、Stephanie Pereira、Xianjie Chen、Wenlin Chen、Vijay Rao、Bill Jia、Liang Xiong 和 Misha Smelyanskiy。面向个性化和推荐系统的深度学习推荐模型。CoRR，abs/1906.00091，2019 年。[31] NVIDIA。NVIDIA DGX 系统：专为独特需求打造的系统。2021 年。https://www.nvidia.com/en-us/data-center/dgx-systems/. [32] NVIDIA。NVIDIA Magnum IO GPUDirect Storage 概览指南。2021 年。https: //docs.nvidia.com/gpudirect-storage/overview-guide/index.html. [33] Adam Paszke、Sam Gross、Francisco Massa、Adam Lerer、James Bradbury、Gre- gory Chanan、Trevor Killeen、Zeming Lin、Natalia Gimelshein、Luca Antiga、Alban Desmaison、Andreas Kopf、Edward Yang、Zachary DeVito、Martin Raison、Alykhan Tejani、Sasank Chilamkurthy、Benoit Steiner、Lu Fang、Junjie Bai 和 Soumith Chintala。PyTorch：一种命令式风格、高性能的深度学习库。载于 H. Wallach、H. Larochelle、A. Beygelzimer、F. d'Alché-Buc、E. Fox 和 R. Garnett 编，《神经信息处理系统进展第 32 卷》，第 8024–8035 页。Curran Associates, Inc.，2019 年。[34] PyTorch。torch.cuda.Stream。2022 年。https://pytorch.org/docs/stable/generated/torch.cuda.Stream.html.[35] PyTorch。torch.cuda.synchronize。2022 年。https://pytorch.org/docs/ stable/generated/torch.cuda.synchronize.html?highlight=torch%20cuda% 20synchronize#torch.cuda.synchronize. [36] Brent Smith 和 Greg Linden。Amazon.com 推荐系统的二十年发展。IEEE Internet Computing，21(3)：12–18，2017 年 5 月。[37] Z. Wang、X. Liu、J. Yang、T. Michailidis、S. Swanson 和 J. Zhao。非易失性内存系统的特征分析与建模。载于《2020 年第 53 届 IEEE/ACM 微体系结构国际研讨会》（MICRO），第 496–508 页，2020 年。[38] Mark Wilkening、Udit Gupta、Samuel Hsia、Caroline Trippel、Carole-Jean Wu、David Brooks 和 Gu-Yeon Wei。ReCSSD：面向基于固态硬盘的推荐推理的近数据处理。载于第 26 届 ACM 编程语言与操作系统架构支持大会论文集（ASPLOS 2021），第 717–729 页，美国纽约州纽约市，2021 年。计算机协会。[39] Kai Wu、Frank Ober、Shari Hamlin 和 Dong Li。使用高性能计算 I/O 工作负载对英特尔傲腾非易失性内存进行早期评估。CoRR，abs/1708.02199，2017 年。[40] Kan Wu、Andrea Arpaci-Dusseau 和 Remzi Arpaci-Dusseau。迈向英特尔傲腾固态硬盘的不成文契约。载于第 11 届 USENIX 存储与文件系统热点研讨会（HotStorage 19），华盛顿州伦顿，2019 年 7 月。USENIX 协会。[41] Jie Amy Yang、Jianyu Huang、Jongsoo Park、Ping Tak Peter Tang 和 Andrew Tulloch。使用缓存的混合精度嵌入。CoRR，abs/2010.11305，2020 年。[42] Weijie Zhao、Deping Xie、Ronglai Jia、Yulei Qian、Ruiquan Ding、Mingming Sun 和 Ping Li。用于大规模深度学习广告系统的分布式分层 GPU 参数服务器。CoRR，abs/2003.05622，2020 年。[43] Guorui Zhou、Na Mou、Ying Fan、Qi Pi、Weijie Bian、Chang Zhou、Xiaoqiang Zhu 和 Kun Gai。用于点击率预测的深度兴趣演化网络，2018 年。