基于CXL的分布式内存系统设计与前沿技术探索

秦杰 [1)]

[1)]（华中科技大学计算机科学与技术学院, 武汉 中国 430074）

摘 要 随着技术协议发展，CXL 缓存一致性协议进入到大家的视野。CXL 作为一种开放的工业标准级的互联协议，由于其高可扩展性、访问一致性、细粒度共享等良好特性，正逐步投入到分布式内存系统优化设计、分层内存设计等相关领域当中。

本文以分布式内存系统为切入点，引入新型高速互联优化技术 CXL，并对于 CXL 的提出背景、主要设计原则、版本和设备类型进行介绍。而后深入探索 CXL 技术在分布式内存系统设计优化当中的具体进展和相关工作，注重讨论云服务下 CXL 内存池化技术。最后，将 CXL 技术运用在当中计算机其他领域的前沿研究进行总结梳理，发现其在分层内存、内存扩展、异构存储系统搭建上也存在良好前景。

最后总结本文，提出对于 CXL 技术的展望，期待前沿研究充分利用 CXL 协议的良好特性扩展技术应用方向，为未来的计算和存储系统带来更多的创新和变革。

关键词 CXL；分布式内存；池化；内存扩展；异构存储

Abstract With the development of technical protocols, the CXL cache consistency protocol has entered everyone's field of vision. CXL, as an open industry-standard level interconnection protocol, is gradually being applied in related fields such as distributed memory system optimization design and hierarchical memory design due to its good characteristics such as high scalability, access consistency, and fine-grained sharing.

This article takes distributed memory systems as the starting point, and introduces the new high-speed interconnection optimization technology CXL,such as the background, main design principles, versions and device types of CXL. Then, we will delve into the specific progress and related work of CXL technology in the design and optimization of distributed memory systems, focusing on discussing CXL memory pooling technology under cloud services.Finally,the application of CXL technology in cutting-edge research in other fields of computer science was summarized and sorted out, and it was found that it also has good prospects in layered memory, memory expansion, and heterogeneous storage system construction.

Finally, this article makes an conclusion and makes up with the prospects for CXL technology like fully utilizing the good characteristics of CXL protocol and expanding the direction of technology application, bringing more innovation and transformation to future computing and storage systems.

Key words CXL;Distributed Memory;Pooling;Memory Expansion;Heterogeneous Storage

1 引言

本章主要阐述分布式内存及系统相关概念以及传统优化相关技术，如内存数据压缩、分布式共享内存等，同时介绍新型高速互联技术（RAMA、CXL），主要探讨 CXL 技术在分布式内存系统优

化当中所起的作用。本文后续将介绍 CXL 技术除了在分布式内存当中的应用探索外，还将深入扩展探讨 CXL 技术在存储领域的前沿探索。

1.1 分布式内存及系统

分布式内存指的是一个计算机系统具有多个处理器，其中每个处理器可以位于不同的计算机上，而且都有自己的私有内存。计算任务只能操作本地数据，如果需要远程数据，计算任务必须与一个或多个远程处理器进行通信。这与提供单一内存空间的共享内存多处理器形成了鲜明的对比。在共享内存多处理器中，处理器不必知道数据所在的位置，但可能会有性能损失，并且要避免竞态条件。

分布式内存系统通常包括处理器、内存和某种形式的互连，允许每个处理器上的程序相互交互。这种互连可以通过点对点链接组织，或者通过单独的硬件提供交换网络。节点之间的链接可以使用标准的网络协议（例如以太网）实现，也可以使用定制的网络链接。

分布式内存的优势在于它可以为大规模计算提供更多的资源，并确保高效的数据访问和处理。例如，在大型科学计算或数据分析任务中，分布式内存可以提供更大的内存容量和更高的并发处理能力。与此同时，分布式内存通过网络等方式将各个计算机的内存集中统一管理调度，做到内存级别的大容量和高并发。

1.2 分布式内存传统优化设计技术

内存数据压缩技术：借助内存数据压缩技术，可以减小分布式计算时的数据传输量，从而提升计算效率。常见的数据压缩算法有 LZ77、Huffman 编码等。

分布式共享内存技术：分布式共享内存技术是将内存地址空间分布在多个节点上，从而形成虚拟的共享内存，不同的处理器可以更高效地访问和共享数据，从而提高计算性能。

数据缓存层：在存储节点上给多块 HDD 配置一块 SSD，再使用开源 BCache 方案，可以增加数据缓存层来降低延迟。这种方式是一种通用的经济实惠解决方案。

硬件升级和优化：例如使用 NVMe 和 RDMA 来优化 IO 路径和通信框架，可以进一步优化每个模块的处理时间。一般存储厂商的通用有效方式是利用增加数据缓存层来降低延迟。

1.3 新型高速互联 RDMA 技术

RDMA（Remote Direct Memory Access）是一种新型的高速互联技术，它允许计算机直接访问其他计算机的内存，而不需要经过处理器的处理。RDMA 技术通过将数据直接从一台计算机的内存传输到另一台计算机，无需操作系统的介入，从而消除了数据传输过程中的延迟和资源消耗。它利用高速的网卡和网络接口实现高速数据传输，从而进一步提高分布式内存系统的性能和效率。

RDMA 技术实现的关键在于将网络层和传输层放到了硬件中，服务器的网卡上来实现。数据报文进入网卡后，在网卡硬件上就完成四层解析，直接上送到应用层软件，四层解析 CPU 无需干预。这种技术可以避免传统 TCP/IP 协议的开销和延迟，从而提高数据传输效率。

RDMA 技术在分布式内存设计优化当中的作用主要体现在以下几方面：

零拷贝：RDMA 技术可以实现零拷贝数据传输，即应用程序可以直接执行数据传输，而不需要将数据从网络软件栈复制到内核空间或从内核空间复制到网络软件栈。这种零拷贝特性可以大大减少数据传输过程中的系统开销，提高数据传输效率。

减少上下文切换：在传统的网络通信中，数据传输需要经过多次上下文切换，即数据的发送和接收都需要进行一系列的系统调用和处理。而 RDMA 技术可以减少上下文切换的次数，从而降低系统开销和延迟。

高速数据传输：RDMA 技术可以利用高速的网

卡和网络接口实现高速数据传输，从而进一步提高分布式内存系统的性能和效率。

适用于大规模并行计算机集群：RDMA 技术适用于大规模并行计算机集群，它可以实现高效的分布式内存管理和数据传输，从而提高整个集群的性能和效率。

1.4 新型高速互联 CXL 技术

CXL（Compute Express Link）是一种新型的高速互联技术，旨在提供更高的数据吞吐量和更低的延迟，以满足现代计算和存储系统的需求。它最初由英特尔、AMD 和其他公司联合推出，并得到了包括谷歌、微软等公司在内的大量支持。

CXL 技术可以有效地解决内存墙和 IO 墙的瓶颈问题。在分布式内存系统中，CXL 技术可以提供高速高效的互联，从而满足高性能异构计算的要求，并且维护 CPU 内存空间和连接设备内存间的一致性。

CXL 技术在分布式内存设计优化当中的作用主要体现在以下几个方面：

高速数据传输：CXL 技术提供了高速的数据传输，有效地解决内存墙和 I/O 墙的瓶颈问题。它通过直接在 CPU 和设备间传输数据，避免了传统内存访问方式的延迟和资源消耗。

内存共享：CXL 技术可以实现内存共享，多个设备可以同时访问相同的内存区域。这可以有效地提高分布式内存系统的并行处理能力，从而提高整体性能。

硬件一致性：CXL 技术可以维护 CPU 内存和连接设备内存之间的一致性。这意味着在分布式内存系统中，不同设备间的数据可以保持同步一致，从而避免了数据不一致的问题。

高扩展性：CXL 技术可以扩展到更多的设备和应用场景。它可以支持多个设备间的互联，从而实现更高效的分布式计算和存储。

下一章将具体介绍 CXL 相关背景和技术前沿。后续章节将针对 CXL 技术应用于分布式内存优化以及其他计算机领域进行探讨和深入，并基于相关技术进行总结分析，提出一些思考和未来可能延伸的方向。

2 CXL 技术背景介绍

本章将针对 CXL 技术进行深入介绍，具体包括该技术提出的背景、设计的主要原则、以及已经发布的历史版本和设备类型，最后还会针对 CXL 技术当前研究现状进行简单概述。

CXL 是一个开放的行业标准，它定义了 CPU 和设备之间的一系列互连协议。CXL 协议跨越了整个计算堆栈，并涉及到计算机科学的许多分支。作为一种通用的设备互连协议，CXL 对设备进行了广泛定义，包括图形处理单元（GPU）、通用图形处理单元（GP-GPU）、现场可编程门阵列（FPGAs），以及各种专门构建的加速器和存储设备。CXL 还适用于传统上通过双数据速率（DDR）并行接口连接到 CPU 的内存。

2.1 CXL 的提出背景

虽然 PCIe 和 DDR 是各种设备互联的良好接口，但它们也有一些固有的局限性。这些限制导致了以下挑战，从而促使了 CXL 的开发和部署。

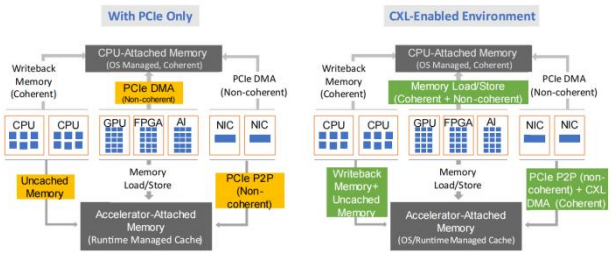


图 1 仅使用 PCIe 以及支持 CXL 的互联环境结构

挑战 1：对系统和设备内存的一致访问。系统内存通常通过 DDR 连接，并通过 CPU 缓存层次结构进行缓存。相比之下，从 PCIe 设备对系统内存的访问是通过非一致性的读/写方式进行的，如图 1 所示。设备内存不能映射到可缓存的系统地址空间。

但在人工智能、机器学习和智能网络接口卡等模型中，设备寻求使用设备本地缓存与 CPU 同时访问相同数据结构的部分，而不来回移动整个数据结构。这一挑战也出现在内存中处理（PIM）领域，目前还没有标准化方法使用 PIM 设备一致地访问 CPU 缓存层次结构中的数据。这导致了繁琐的编程模型，阻碍了 PIM 的广泛采用[1][2]。

挑战 2：内存可扩展性。对内存容量和带宽的需求与计算量的指数增长成比例地增长。DDR 内存无法满足这一需求，限制了每个 CPU 的内存带宽。这种缩放不匹配的一个关键原因是并行 DDR 接口的固定效率低下。通过添加 DDR 通道进行扩展又会显著增加了平台成本，并引入了信号完整性挑战。而 PCIe 不支持相干性，并且设备连接的内存不能映射到相干内存空间，PCIe 也一直无法取代 DDR。

挑战 3：由于资源搁浅而导致的内存和计算效率低下。由于资源搁浅，数据中心效率低下。根本原因是资源的紧密耦合，其中计算、内存和 I/O 设备只属于一个服务器。因此，每个服务器都需要过度提供内存和加速器，以处理具有峰值容量需求的工作负载。另一方面，工作负载使用的所有核心服务器通常都存在内存未使用，这样的资源搁浅导致低功率[3]、高成本[4]和持续性影响[5]。

挑战 4：分布式系统中的细粒度数据共享。分布式系统经常采用细粒度同步。底层更新通常很小，并且对延迟敏感。例如，在网络规模的应用程序中的分区/聚合设计模式系统中，查询更新通常低于 2kb。以如此细的粒度共享数据，意味着典型数据中心网络中的通信延迟主导了更新等待时间，并减慢了这些重要的用例[6][7]。而一个连贯的共享内存实现可以帮助减少通信延迟到亚微秒。

2.2 CXL 设计的主要原则

CXL 采用了**非对称的方法来实现一致性、向后兼容和开放，以实现多样化和开放的生态系统，促进广泛的部署。CXL 相干性与主机特定的相干性协议细节解耦**。主机处理器还负责协调缓存一致性，以便简化在设备中实现一致性。设备的缓存代理使用一个小的命令集强制执行一个简单的 MESI 一致性协议。CXL 通过提供多个复杂性不同的协议来支持多个用例，而设备只能实现协议的一个子集。

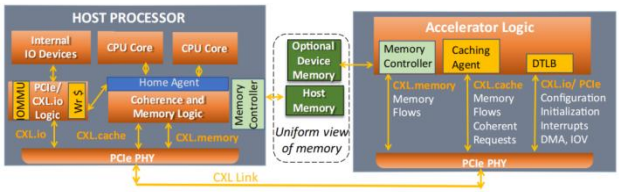


图 2 利用 CXL 在 PCIe PHY 上的动态多路复用结构图

CXL 运行在 PCIe PHY 上，设备可以**插入任何 PCIe 插槽**。CXL 的**向后兼容发展**以及它与 **PCIe 的互操作性**确保了各大公司在 CXL 上进行投资，并保证与上一代 CXL 设备以及任何 PCIe 设备的互操作性。PCIe 基础设施的重用降低了进入障碍，因为相同的 IP 构建块、平台、通道以及软件基础设施都可以被重用。从 SoC 的角度来看，一个支持多协议的 PCIe PHY 如图 2 所示，有助于减少硅面积、引脚计数和功率。

2.3 CXL 主要版本与设备类型

CXL 的开发是为了解决 2.1 当中提到的挑战和其他挑战，实现对系统和设备内存的一致访问、提升内存可扩展性、解耦合服务器各种资源避免资源搁浅问题、分布式数据共享通信延迟等。

表 1 CXL 各版本规范、速度和用例概述

Generation	Year	Scope	Speed	Use case
CXL 1.0 & CXL 1.1	2019	Single machine	32 GT/s	Accelerators (Challenge 1) Bandwidth and capacity expansion (Challenge 2)
CXL 2.0	2020	2-16 machines (single switch)	32 GT/s	Small-scale resource **pooling** (Challenge 3)
CXL 3.0	2022	100s machines (multiple switches)	64 GT/s	Large-scale resource pooling and **sharing** (Challenges 3 and 4)

自 2019 年首次发布以来，CXL 已经进化了三代。每一代都指定互连和多个协议，同时保持完全向后兼容。表 1 概述了当前 CXL 各代版本和关键用例示例。

CXL1.0：该规范在 PCIe 上增加了一致性和内存语义。通过允许 CXL 设备缓存系统内存，这解决了挑战 1 和挑战 2。这也标准化了一个连贯的接口，以促进 PIM 系统和编程模型的广泛采用。CPU

还可以缓存设备内存，这解决了异构计算的细粒度数据共享问题。附加到 CXL 设备的内存可以映射到系统可缓存内存空间，这促进了异构处理，并有助于解决内存带宽和容量扩展挑战。CXL 1.0 还继续支持 PCIe 的非连贯的生产者-消费者语义[8][9][10]。

CXL2.0：该规范通过跨多个主机启用资源池来解决挑战 3。我们使用主机是指在单个操作系统或系统管理程序的控制下的单套接字或多套接字系统。池化通过随着时间的推移将资源（例如内存）重新分配给不同的主机，克服了资源搁浅和碎片化，而不必重新启动这些主机。CXL 协议通过引入 CXL 交换机构建包含主机和内存设备的小型网络来实现池化。

CXL3.0：该规范依靠更大规模多级别的 CXL 切换解决挑战 3。这使得在机架甚至 pod 级上构建动态可组合系统。此外，CXL 3.0 通过启用跨主机边界的细粒度内存共享，解决了挑战 4。

CXL 设备可以根据其所支持的协议分为三种类型。

类型 1 设备，保证基本的缓存一致性，并支持 CXL.io 和 CXL.cache 协议。

类型 2 设备，支持所有三个协议 CXL.io、CXL.cache 和 CXL.mem，具有完全一致的缓存和它们自己的内存连接到设备本身。与只能覆盖原子操作的 1 类型设备相比，2 类型设备可以覆盖更复杂的操作，因为它们的加速器和设备连接的内存之间有巨大的带宽。

类型 3 设备，它只支持 CXL.io 和 CXL.mem 协议，并被用作主机系统的内存扩展器。这种类型的 CXL 设备不会通过 CXL.cache 发出任何请求，所以它只在 CXL.mem 上操作从主机 CPU 发送的服务请求。

2.4 CXL 当前研究现状概述

浏览当前 CXL 技术相关论文，可以发现目前 CXL 相关研究领域主要集中在基于 CXL 的分布式内存系统设计和性能探索（云服务下基于 CXL 实现内存池化的性能权衡需求、基于 CXL 技术在云性能要求下提出的内存池化系统、DirectCXL 内存分离技术等）。

同时大量研究集中在基于 CXL 缓存一致性协议进行的内存技术和运用探索（如基于 CXL 内存的页面置换方法设计 TPP、基于 CXL 协议优化廉价内存、基于 CXL 连接的软硬件组合设计方案等）。另外还有部分研究，探索基于 CXL 内存扩展技术调研大数据下深度学习加速、机器学习算法优化等内容，具体应用于基因组分析、商业云服务器数据分析等。

3 基于 CXL 的分布式内存系统设计

本章将结合当前高层次论文，了解目前分布式内存优化应用场景当中对于 CXL 技术所提出的要求以及设计前沿。具体包括，云服务下基于 CXL 实现内存池化的性能权衡需求、基于 CXL 技术在云性能要求下提出的内存池化系统、DirectCXL 内存分离技术等，对于 CXL 技术的分布式系统运用需求和前景更进一步理解。

3.1 云服务平台下 CXL 内存池化的性能权衡

DRAM 是公共云服务器性能和成本的关键驱动因素。与此同时，由于跨服务器的分散使用，大量的 DRAM 没有得到充分利用。像 CXL 这样新兴的互联协议提供了一条通过内存池来提高利用率的途径[11]，且相关成本远远低于 DRAM。我们可以通过 CXL 使用缓解云服务下硬件成本昂贵、延迟和性能要求较高、缓解内存搁浅导致的内存开销和闲置。

但是可以发现，基于 CXL 的内存系统的设计空间很大，其关键问题在于内存池的大小、覆盖范围和拓扑结构。同时，使用内存池化需要围绕性能、虚拟化和管理导航复杂的设计约束。CXL 内存访问延迟取决于整个系统的设计，特别是池的大小和拓扑结构。而每个 CXL 组件都增加了系统成本，这

必须与内存搁浅节省相平衡，即本文所谓的性能权衡。

一文[12]针对上述问题进行了提出和讨论，基于 Azure 探讨了云服务工作负载的的特征，具体探讨了 Azure 的内存资源搁浅问题、Azure 虚拟负载内存利用情况以及工作负载对于内存延迟的敏感性，可以发现内存资源搁浅严重影响系统性能、虚拟负载适合于池化但存在性能奇偶挑战、内存延迟以及带宽变化均表现敏感。关注了通用云计算两个设计方面：1)是通过 CXL 交换机提供连接，还是通过 CXL 多头设备（MHDs）提供连接；2)构建的池应该有多大，以提高 ROI。对于 MHD 设计，最重要的权衡是传入的 CXL 端口和 DDR 通道的数量。

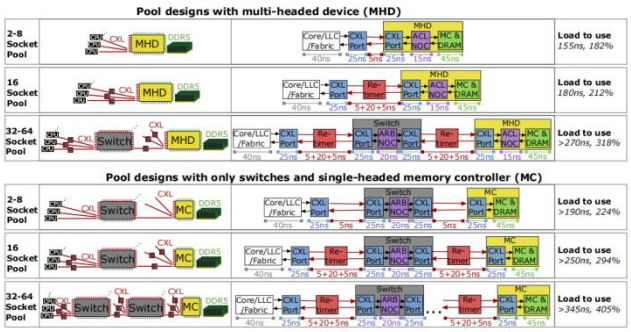


图 3 内存池大小和性能延迟的权衡

从图 3 来看，大规模的池化会严重影响性能，因此基于 CXL 的内存池的大小很可能是机架大小的一个子集，以最小化访问延迟的性能影响。而后本文针对池化大小与 DRAM 节省情况，进行权衡分析如图 4，发现池化大小增加可以有效减少 DRAM 的使用，由 32 个套接字组成的小池足以显著降低总体内存需求。但当池化到一定大小后，减少趋势明显放缓。

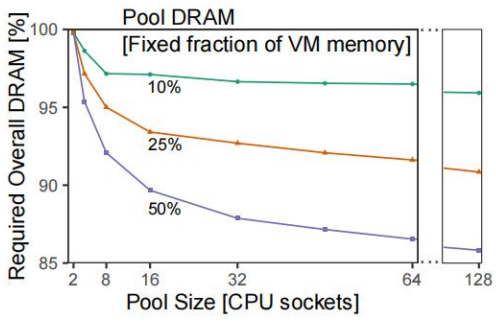


图 4 池大小对于总体内存需求的影响

最后，针对池化大小和系统成本进行了分析，参数的示例性值大致基于硅面积以及支持存储池所需的连接性和基础设施的估计，结果如图 5。正投资回报要求池设计者在池大小、拓扑结构和节省之间进行复杂的权衡，这依赖于工作负载。基础设施开销可能成为采用基于 CXL 的池化的一个主要障碍，因为昂贵设计的配置不能提供有利的投资回报率。

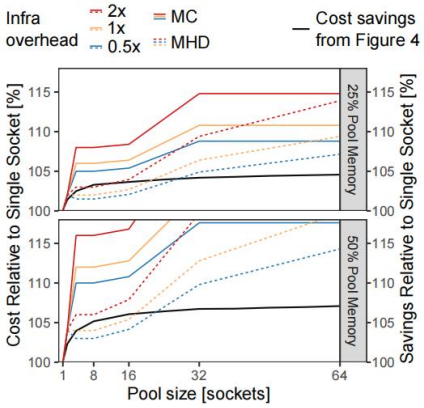


图 5 池大小与系统成本之间的权衡关系

本文引入 CXL 技术在云服务当中实现内存池化，对于这样的实现作者针对各种指标进行了权衡和探讨，包括 DRAM 的节省、系统整体搭建的效益、内存访问延迟和用户体验、基于 CXL 的内存池化大小等。可以发现，通过内存控制器与 CPU 套接字解耦，可以更快地探索和部署新的控制器特性。而面向云服务，云提供商可以基于 CXL 需要提升可靠性、可用性和可服务性功能，通过性能权衡挖掘，提升正投资回报。

3.2 在云性能要求下提出的 CXL 内存池化系统

3.1 节我们具体探讨了云性能要求下对于 CXL 进行内存池化所需要进行的性能权衡问题，具体包括内存池的大小、覆盖范围和拓扑结构因素等，各个部分都会最终影响系统搭建后正向回报比（DRAM 节省、性能均衡、硬件成本），可以发现进行 CXL 内存池化系统搭建是一件多因素影响多方权衡的工作。有研究[4]完成了内存池化系统设计，视主内存为云公共厂商性能和成本的一个关键驱

动因素。

该文提出了第一个既满足云性能目标，又能显著降低 DRAM 成本的内存池系统 Pond。Pond 建立在 CXL 标准上，以加载/存储访问池内存。通过对云生产跟踪的分析表明，跨 8-16 个套接字的池化可以保证大部分优势，使得低访问延迟的小池设计成为可能。其次，可以创建机器学习模型，准确预测分配给虚拟机的本地内存和池内存大小，逼近相同 NUMA 节点内存性能。

本文基于在 Azure 工作负载下的的测量和观察（具体见 3.1 节），以下述目标进行池化系统设计，性能逼近 NUMA-本地 DRAM、兼容虚拟化加速器、兼容不透明 VM 和不变的客户操作系统/应用程序、保证主机资源开销小。在硬件层次上，Pond 池中的主机具有单独的缓存一致性域，并运行单独的管理程序。Pond 使用一个所有权模型，其中池内存在主机之间显式地移动。外部内存控制器（EMC）ASIC 使用多个 DDR5 通道，如图 6，EMC 是多头的，允许连接多个 CXL 主机和 DDR5 DIMM。通过一个以 PCIe 5 速度运行的 CXL 端口集合来实现该池。进行性能权衡研讨，相对于本地 NUMA DRAM，8 个和 16 个插座池只增加了 70-90ns，Pond 池减少了 1/3 的延迟。

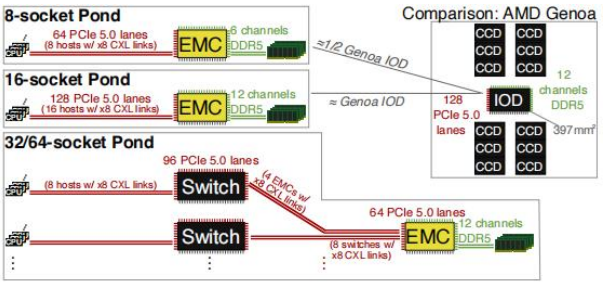


图 6 外部内存控制器（EMC）

系统软件层上，包括了 Pond 池管理、故障管理、将池化内存暴露给虚拟机、内存分配的重新配置、不透明 VM 的遥测（收集与内存性能相关的硬件计数器、跟踪虚拟机的未访问的页面）。分布式控制平面层上，实现两项主要任务——在 VM 调度期间分配内存的预测和 QoS 监控和解析，如图 7。而这两项任务，具体使用了机器学习算法进行实现，非本文的重点不再赘述。Pond 池使用 16 个插座，假设 CXL 增加 222%的延迟，则会减少 7%所需的 DRAM。这意味着云服务器的成本总体上降低了 3.5%。

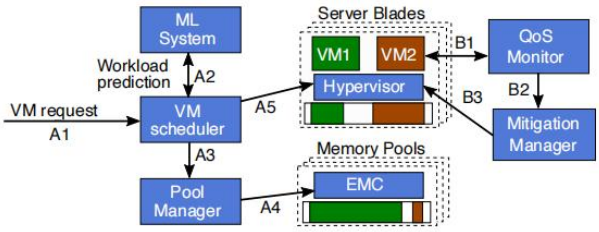


图 7　Pond 池控制平面工作流程

可以发现，当前 CXL 前沿研究，已经将 CXL 技术运用于了内存池化领域当中，并且经过性能权衡分析构筑 CXL 基础设施，搭建起了一个符合公共云应用的内存池化系统，呈现出了 CXL 技术用于分布式内存系统优化的前景。仔细分析，CXL 技术的特点在于其高速数据传输、内存共享、硬件一致性和扩展性等方面，这些特性使得 CXL 成为构建内存池化系统的理想选择。Pond 系统的设计评估，充分利用 CXL、机器学习等领域技术辅助提升内存池化正向收益，进一步证明了 CXL 技术在构建高效内存池化系统中的潜力。

3.3 直接访问内存分离技术 DirectCXL

内存分离方法主流有三类，一是借助 RDMA[13][14]，二是利用交换池（基于页面的内存池）[13][15]，三是 KVS（基于对象的内存池）[16][17]。但是分离式内存方法归根结底都需要通过 RDMA 将数据从远程内存移动到主机内存（或类似的细粒度网络接口）。此外，它们甚至需要管理主机或内存节点中的本地缓存数据。但数据移动及其伴随的操作引入了冗余内存副本和软件结构干预，这使得分离内存比本地 DRAM 访问延迟高多个数量级。

DirectCXL 一文[18]提出了直接访问的内存分离技术，即通过 CXL 内存协议（CXL.mem）直接连接主机处理器远程复杂的内存资源。本文探索了一

种基于CXL的内存分离实际设计，还提供了在CXL软件运行时，允许用户通过纯粹的加载/存储指令利用底层的分离内存资源，它简单地将HDM的地址空间分割成多个段（CXL名称空间）。然后，直接CXL运行时允许应用程序作为内存映射文件（mmap）访问每个CXL名称空间。

具体来说，首先提出了在CXL上分离内存方式，并将分离后的内存集成到处理器端系统内存中。这包括实现在远端使用多个DRAM模块的CXL控制器。CXL.mem允许主机计算资源通过PCIe总线（FlexBus）直接访问底层内存。它的工作原理类似于本地DRAM，连接到它们的系统总线。然后，提出了一组网络基础设施组件的原型，如CXL交换机，以便使分离内存以可扩展方式连接到主机。

文中通过CXL设备和控制器以及模拟的CXL网络交换机，并将设备整合入系统当中如图8，引入了CXL控制器、BAR、HDM等结构，CXL控制器将基于PCIe产生的CXL数据包转化为DRAM请求进行处理。主机端内核驱动程序首先通过PCIe事务查询CXL设备的基本地址寄存器（BAR）及其内部内存的大小，并让底层CXL设备知道它们的BAR和HDM（基本地址）映射在主机的系统内存中的位置，进而转化为CXL设备内DRAM进行相应处理。本文还说明了DirectCXL如何使用一个或多个和CXL设备从主机上分离内存资源，结构如图9a，显示了在其中的CXL交换机组织如图9b。

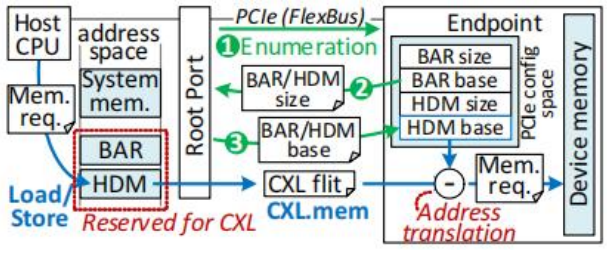
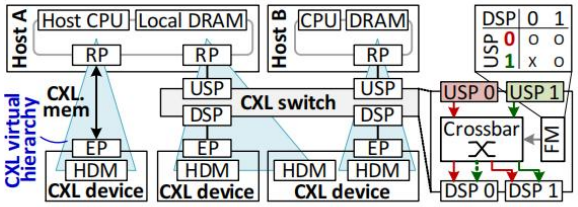


图8 DirectCXL的连接方式

（CXL设备内部DRAM通过CXL被映射到主机内存空间）



(a) CXL virtual hierarchy. (b) CXL switch.

图9 DirectCXL的网络和交换机结构

本文进一步设计了CXL网络拓扑结构，并在实际系统中进行了相应的实现。实际系统评估结果表明，当工作负载可以享受主机处理器的缓存时，DirectCXL的分离内存资源可以表现出类似DRAM的性能。对于现实世界的应用程序，它比基于RDMA的内存分离系统平均好3倍的性能。

本节针对分离式内存CXL前沿技术进行了详细阐述，可以发现CXL技术适配于分离式内存系统设计，同时有自洽的实现逻辑。相对于基于RDMA的内存分离系统内存分离访问延迟影响大，它基于资源分离和硬件异构能力分离式内存系统表现出了类似于DRAM的性能，性能大大提升。后续可以考虑正在扩展有效的CXL内存管理的内核，并考虑将SoC硅作为直接访问内存分离技术的未来工作。

4 CXL内存技术前沿与运用探索

本章将针对CXL内存技术前沿和运用探索进行阐述，具体来说，有文章提出基于CXL内存的页面置换方法TPP（CXL内存置于分层内存系统中）、启用廉价闪存的CXL设计、使用CXL内存扩展器的异构系统的软件定义分层内存等，探索CXL技术在非分布式内存系统当中的运用。

4.1 基于CXL内存的页面置换方法

数据中心程序对于内存的需求日益增长[19]。CXL可以对于昂贵的主存进行丰富扩展，甚至构建异构主存系统，为上述问题的有效解决途径，即将CXL技术应用于分层内存系统扩展优化当中。但没有有效的内存管理机制，这样的系统只会降低性能。TPP[20]一文基于Chameleon描述在生产服务器组中运行多年的内存绑定应用程序的观察结果，提出了

一种支持 CXL 的内存操作系统级应用程序透明页面放置机制（TPP）。

TPP 机制的关键思想包括轻量级回收迁移、解耦分配和回收、CXL 节点的页面提升（陷入的热页面的适当识别）、页面类型感知分配。

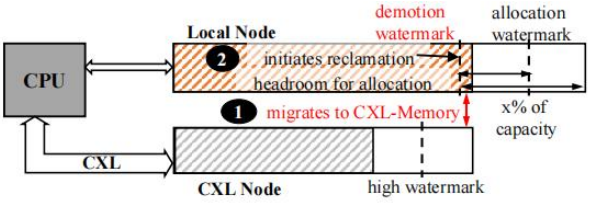


图 10 TPP 解耦本地内存节点的分配逻辑和回收逻辑（以及页面迁移至 CXL 内存的过程）

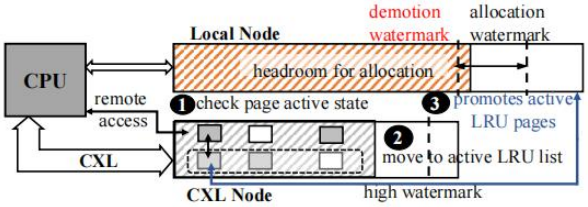


图 11 TPP 机制提升陷入 CXL 内存的页面

本文关键是通过设置 demotion_watermark 和 allocation_watermark 来解耦并控制分配和回收，且前值设置得比后面低，以侵占式回收保证大量空闲空间。同时又可以使用工作负载监视工具来动态地调整回收阈值(回收的侵占性)，如图 10。另一个是从 CXL 节点进行页面提升，如图 11。对于被提升的界面随机性进行了改进，采用 LRU 表，仅对活跃的进行提升；针对非活跃的，标记活跃下次在被抛出时才被提升，大大降低了提升率。为本地结点留出了空闲空间，提高了提升的成功率，避免了提升的盲目性。

整体而言，本文提出的 TPP 机制为 CXL 进行内存扩展的良好实践，满足昂贵主存廉价化扩展，还可以构建异构存储系统充分利用不同存储器件特性，构筑了一套性能较好的页面放置分配机制，显著降低对于系统性能的影响。

4.2 启用廉价闪存的 CXL 设计

计算系统中计算能力和内存容量需求之间日益不平衡已经发展成为一个被称为内存墙的挑战。有研究表明，可以基于 CXL 技术利用闪存的方式克服内存墙的问题（CXL 技术保证了优秀的可扩展性）。CXL 类型 3 设备作为内存扩展的重点设备，虽然 CXL 目前只考虑将 DRAM 和 PMEM 作为主要的内存扩展设备，但由于 CXL 的一致性内存存取特性，它可以使用 SSD。基于闪存的 SSD 的高容量和更好的缩放，通过在 3D 水平中叠加[21]和存储多个位[22]，可以有效地解决现代数据密集型应用程序面临的内存墙。这也正说明了通过使用 CXL 内存扩展到闪存 SSD 上，是有效的同时可能避免内存墙问题。

一文[23]针对使用闪存作为 CPU 可访问主存的主要的挑战进行了提出，内存请求和闪存之间存在粒度不匹配、闪存仍然比 DRAM 慢几个数量级、闪存持久性有限（重复写入会产生磨损）。该文通过合成工作负载，展示了通过集成各种系统设计技术如缓存和预取，有效地减少 CXL 闪存延迟。并使用真实世界的工作负载，分析了当前预取器的局限性，并建议对未来的 CXL 闪存进行系统级更改，以实现接近 DRAM 的性能，特别是设备低μs 延迟。

本文不仅对 CXL 闪存架构的设计空间进行了探索（不同闪存类型生命周期、性能评估），同时还评估高级缓存和预取策略（预取数据能够提升 CXL 闪存性能，解决延迟问题）。对于本文采用和评估的预取相关策略不再赘述。

虽然该工作没有直接研究内存分离系统，但使用 CXL-flash 作为分离式内存有助于克服内存墙问题。同时 CXL 技术引入内存扩展当中，还为机器学习部分领域带去了新的搭建硬件，如 NLP 领域需要的内存大对带宽不敏感的设备。总结相关 CXL 扩展闪存结构，所提出的设计并未考虑到闪存内部任务，如垃圾回收和磨损均衡等。此外，主机系统可能不能完全反映 CXL 所引入的新系统特性。因此，在 CXL-flash 研究领域做更多的工作是需要。

4.3 使用 CXL 内存扩展器的异构系统的软件定义分

层内存

新兴的应用程序，如人工智能、大数据和边缘计算，需要高密度和高带宽的内存解决方案，以获得更好的性能和系统利用率。当前外围 I/O 基于 PCIe 协议连接，导致低带宽、服务器与系统网络延迟以及不一致问题。CXL 作为一种开放标准的互连协议，通过有效地扩展内存容量和带宽，克服了架构上的限制。因此有部分研究去探讨将高性能的 I/O 总线架构 CXL 技术运用在内存系统当中进行扩展，提出一个高效且经济的解决方案，提升当前数据任务如人工智能、传统内存数据库等的吞吐量。

SMT[24]就是这样一个高效经济的解决方案，其包括 CXL 附加的内存硬件和一个软件套件。内存模块硬件集成了双数据速率（DDR）动态随机存取内存（DRAM）和 CXL 控制器，扩展了每秒数十 GB 的带宽，并增加了数 TB 的内存容量。与传统的仅使用 DDR 的内存系统相比，所提出的 CXL 解决方案将内存数据库（IMDB）和人工智能应用的吞吐量分别提高了 1.5 倍和 1.99 倍。

SMT 建立了一个内存扩展器原型，它可以使用 CXL 控制器来提高系统的内存容量和带宽，用 ASIC 控制器构建了 CXL 内存扩展器模块，完成了对 HDM 的读/写操作的验证，并在商业服务器系统环境中执行了应用程序评估。同时为异构内存系统开发了一个具有内存扩展器的新软件套件 SMDK，具体架构上分为 SMDK 分配器和 SMDK 内核，如图 12 所示。SMDK 可以在一个带有内存扩展器的异构内存系统上执行内存分层、管理和智能分配，实现了软件定义内存（SDM）的概念。由于传统 DRAM 和 CXL DRAM 设备之间固有的硬件差异，SMDK 内核还扩展了 Linux VMM，以更有效地管理这两种混合内存类型，其继承了 Linux VMM 层次结构的设计。

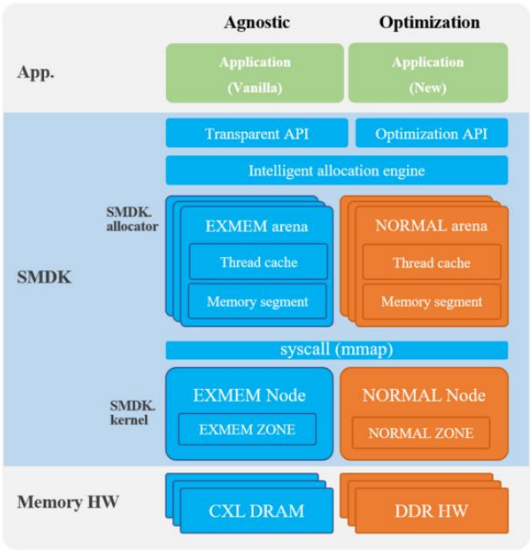


图 12 SMDK 结构图

本文提出的 CXL 内存扩展逻辑器件以及扩展软件套件 SMDK，两者相互紧密地支持，其脱离于了常用的摆脱内存控制器和通道数量限制问题的 HCI 和扩展架构解决方案。在 CXL 技术的前沿，扩大 SMDK 的垂直和水平覆盖范围，有望成为高密度和高带宽新的内存解决方案，为新一代的超大型数据仓库做出贡献。

5 总结与展望

对于本文内容进行总结梳理，本文以分布式内存系统设计作为出发点，探究了分布式内存系统具体背景以及当前传统优化方式以及新型高速互联技术（RDMA、CXL）引入产生的优化新方式。由此，本文进入综述重点 CXL 缓存一致性协议的研讨工作。

第二章对于 CXL 协议的提出背景、设计主要原则、主要版本和类型设备，简单阐述了其在计算机系统领域内的研究前景和当前方向。第三章，回归到第一章内容如何使用新型高速互联技术 CXL 进行分布式内存系统优化。分析研讨云运用场景下，使用 CXL 技术进行内存池化的性能权衡问题，并结合目前的有效模拟实践，探讨了 Pond 池的 CXL 内存池化构想。同时进入到分离式内存构建当中，CXL 也为相关实践注入了新的活力，产生了一些分

离式内存技术研究，如 DirectCXL。

视野放宽，可以发现 CXL 技术不仅能够运用于分布式内存系统、分离式内存系统当中，还可以扩展进入到分层内存系统上，为当前热门的机器学习、人工智能、传统内存数据库领域带去新的硬件支持——依赖于 CXL 协议支持的异构内存系统扩展、缓存一致性协议等良好特性。比如将 CXL 内存引入到目前的存储系统当中，如何放置 CXL 内存的位置，如何进行页面置换等，提出了 TPP 协议；基于 CXL 技术考虑扩展到 SSD 闪存上，缓解当前存在的内存墙问题（计算能力和内存容量需求之间日益不平衡）；也有研究着重进行基于 CXL 的内存扩展软件套件研究，如 SDMK 的设计，实现了软件定义内存（SDM）的概念。

对于 CXL 技术进行展望，目前的相关理论研究存在对 CXL 协议理解不充分、设备模拟无法精确、许多指标无法进行量化等问题，等后续 CXL 设备实现量产后，相信还能进一步带动 CXL 相关研究领域蓬勃发展。正如 CXL 技术于 2019 年才被正式提出，仍然在开发期，相关新兴领域融合正在逐步推进，无论是步入分布式内存系统、分离式内存系统设计，还是单机内存系统，都在不断吸收 CXL 技术带来的良好特性，做好性能权衡和融合创新，得到性能和成本效益相对最佳的系统设计。

CXL 需要支持复杂的内存共享结构，必须有一套新软件支撑，包括内存调度管理器、内存高级数据特性，包括内存压缩、快照、克隆、备份等，以及内存安全防护。这些都是 CXL 技术运用当中可以考虑挖掘的点。同时继续深挖 CXL 技术引入带给深度学习、机器学习领域的效益，在硬件层面上助力算法层面发展。

CXL 技术在分布式内存系统优化以及其他领域具有重要的应用前景。随着技术的不断发展和完善，CXL 将成为构建高效、可靠、快速的计算和存储系统的关键技术之一。同时，随着应用领域的不断拓展，CXL 技术将为未来的计算和存储系统带来更多的创新和变革。

致　谢　十分感谢数据中心技术课程以及施展老师给予了我这个机会完成一篇关于 CXL 用于内存发展的综述。通过收集整理论文资料、挖掘和探索各篇文章当中的亮点和设计思路，探索了 CXL 技术用于未来内存以及分布式系统当中的良好前景。同时也让我有这样一个机会，深入了解 CXL 相关领域，扩宽了我的知识面，为未来进行 CXL 缓存一致性协议相关研究做好了铺垫。

参考文献

[1] Mutlu, O., Ghose, S., Gómez-Luna, J., & Ausavarungnirun, R. (2022). A modern primer on processing in memory. In Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann (pp. 171-243). Singapore: Springer Nature Singapore.

[2] Singh, Gagandeep, et al. "Near-memory computing: Past, present, and future." Microprocessors and Microsystems 71 (2019): 102868.

[3] Zhang, Chaojie, et al. “Flex: High-availability datacenters with zero reserved power.” 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2021.

[4] H. Li et al, “Pond: CXL-Based Memory Pooling Systems for Cloud Platforms”, ASPLOS ’23, 2023

[5] Berger, Daniel, et al. “Research Avenues Towards Net-Zero Cloud Platforms”. NetZero 2023: 1st Workshop on NetZero Carbon Computing

[6] Alizadeh, Mohammad, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. "Data center tcp (dctcp)." In Proceedings of the ACM SIGCOMM 2010 Conference, pp. 63-74. 2010.

[7] Zhang, Irene, Amanda Raybuck, Pratyush Patel, Kirk Olynyk, Jacob Nelson, Omar S. Navarro Leija, Ashlie Martinez et al. "The demikernel datapath os architecture for microsecond-scale datacenter systems." In Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles, pp. 195-211. 2021.

[8] D. Das Sharma, “Compute Express Link”, white paper, Compute Express Link Consortium, March 2019

[9] CXL Consortium, “Compute Express Link 1.1 Specification”, July 2, 2019

[10] D. Das Sharma, “Compute Express Link®: Enabling Heterogeneous Data-Centric Computing with Heterogeneous Memory Hierarchy”, IEEE Micro, Mar-Apr 2023

[11] CXL Specification. Available at https://www.computeexpre sslink.org/download-the-specification, accessed December 2020, 2020.

[12] Berger, Daniel S., et al. "Design Tradeoffs in CXL-Based Memory Pools for Public Cloud Platforms." IEEE Micro 43.2 (2023): 30-38.

[13] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. Efficient memory disaggregation with infiniswap. In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), pages 649 – 667, 2017.

[14] Marcos K Aguilera, Nadav Amit, Irina Calciu, Xavier Deguillard, Jayneel Gandhi, Stanko Novakovic, Arun Ramanathan, Pratap Subrahmanyam, Lalith Suresh, Kiran Tati, et al. Remote regions: a simple abstraction for remote memory. In 2018 USENIX Annual Technical Conference (USENIX ATC 18), pages 775 – 787, 2018.

[15] Chenxi Wang, Haoran Ma, Shi Liu, Yuanqi Li, Zhenyuan Ruan, Khanh Nguyen, Michael D Bond, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. Semeru: A memory-disaggregated managed runtime. In 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), pages 261 – 280, 2020.

[16] Jacob Nelson, Brandon Holt, Brandon Myers, Preston Briggs, Luis Ceze, Simon Kahan, and Mark Oskin. Latency-tolerant software distributed shared memory. In 2015 USENIX Annual Technical Conference (USENIX ATC 15), pages 291 – 305, 2015.

[17] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K Aguilera, and Adam Belay. Aifm: High-performance, application-integrated far memory. In 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), pages 315 – 332, 2020.

[18] Gouk, Donghyun, et al. "Direct access,{High-Performance} memory disaggregation with {DirectCXL}." 2022 USENIX Annual Technical Conference (USENIX ATC 22). 2022.

[19] S.-H. Lee. Technology scaling challenges and opportunities of memory devices. In 2016 IEEE International Electron Devices Meeting (IEDM), 2016.

[20] Maruf, Hasan Al, et al. "TPP: Transparent page placement for CXL-enabled tiered-memory." Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3. 2023.

[21] K. Parat. and A. Goda. Scaling trends in NAND flash. In 2018 I EEE International Electron Devices Meeting (IEDM), pages 2.1.1 – 2.1. 4, 2018. https: //ieeexplore.ieee.org/document/8614694.

[22] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. Operating Systems: Three Easy Pieces. Arpaci-Dusseau Books, 1.00 edition, August 2018

[23] Yang, Shao-Peng, et al. "Overcoming the Memory Wall with {CXL-Enabled}{SSDs}." 2023 USENIX Annual Technical Conference (USENIX ATC 23). 2023.

[24] Kim, Kyungsan, et al. "SMT: Software-Defined Memory Tiering for Heterogeneous Computing Systems With CXL Memory Expander." IEEE Micro 43.2 (2023): 20-29.

附录 CXL-ANNS论文汇报记录.

问题 1 本文为什么会想到将CXL应用于ANNS近似最近邻搜索任务改进当中？

CXL是一种开放式硬件互连标准，它允许底层内存具有高度可扩展性和可组合性，且成本低。CXL内存可以作为中间层弥补主存和NVMe之间的速度差异gap。CXL技术的异构内存特性可以实现多种不同存储的整合使用，将DRAM使用CXL内存池从主机当中分离出来，将可以保证满足大规模数据集存储需要的同时，减少之前的访问SSD/PM等持久化内存的长延迟情况，同时CXL内存池可以有效解决当前改进方案的不足。

面对大规模数据集实现最近邻搜索，采用传统KNN算法，成本高昂，延迟情况严重。ANNS应运而生，将查询向量限制为仅搜索最有可能成为最近邻的邻域子集，但是ANNS仍然无法解决内存需求和压力问题。目前存在的两种主流改进方法是基于有损压缩的方法（损失精度，且压缩能力有限，仅能压缩嵌入表）以及使用持久存储的层次化方法（将全数据集存放在SSD/PM当中，将搜索任务分为低精度任务和高精度任务。对于高精度任务，仍然需要在SSD/PM当中进行读取数据，延迟较高）。而如果使用CXL内存扩展池，异构地将DRAM从主存当中分离出来，保证访问介质仍然为DRAM延迟较低，同时通过CXL的高扩展性连接多个CXL EP，可以实现对于全数据集完全载入，保证搜索精度不会出现明显下降，达到BigANN提到的 90%的搜索准确性。

同时CXL当中Type 3 设备正好适用于实现这样的内存扩展，并且提供EP内存地址空间暴露给CXL CPU，实现物理地址转化，进而实现虚拟地址转换。可以通过传统的load/store命令对于底层CXL EP的内容进行访问读取。因此引入CXL内存池来装载大规模数据集（如十亿级节点）是完全可行的，能够保证精度要求，同时满足时延较低的问题。虽然仍存在时延相对于拥有无限DRAM的Oracle系统增加的问题，但是作为本文出发点来说，后续会有很多改进。其中在CXL EP端使用DSA领域加速器进行数据向量提前运算，将数据传输从数据向量变为一个距离标量，将由于数据传输量大造成的时延明显问题进行解决，同时削弱CXL CPU的工作负担，将任务分担给了CXL EP端。

问题 2 本文在图相关领域提出了哪些优化，你认为这些关于图搜索相关的亮点足够新颖吗？

本文的图相关内容主要针对的是大规模十亿级节点的图，对于其中每个图节点又映射到了一个高维向量（比如Macrosoft当中一个图结点对应一个 100 维的向量）。主要完成了以下优化工作，首先是通过SSSP算法统计图节点距离ANNS算法当中的入口节点的跳数，将跳数少的节点和数据向量置入本地DRAM当中，将跳数较高的节点和数据向量置入CXL EP当中，充分利用访问的空间局部性原理，将频繁访问的内容放置在本地内存当中，减少延迟开销。另外还加入了预取机制，设计查询调度程序在比实际图遍历子任务需要前就预取图信息。预取的内容通过引用候选数组来推测要访问的节点（由统计验证，在测试的所有数据集下一次迭代的图遍历中，总访问节点的 82.3%来自候选数组）。另外还对于ANNS图算法当中的候选更新操作，进行细粒度划分为紧急和非紧急任务，紧急任务在图遍历前执行，而将候选数组更新、候选数据节点排序子任务放在和距离计算任务时并行完成，减少CXL CPU空闲等待时间。

其实预取策略、节点放置策略、细粒度划分机制等，对于图搜索来说都是较为普遍的做法了，在许多文献当中都存在使用预取策略减少主机CPU等待数据读取的情况，利用空间局部性原理和图算法访存行为进行节点数据乃至数据向量的放置。而本文主要是整体结构基于CXL上，将过去较为普遍的图优化方法加载到了CXL内存池当中，进行了相应的复用。本文对于相关硬件设备协同使用和部署安排显得更加可贵，算是在图算法领域结合CXL技术的先驱代表。

问题 3 你认为本文实现最高的门槛在哪？

正如问题 2 回答的那样，本文最高的门槛是对于CXL设备以及协议的相关理解工作，以及将图算法领域和CXL结合起来的使用，到最后搭建CXL-ANNS软硬件系统，以及全系统模拟。图算法的优化其实已经趋于饱和，存在很对针对不同算法、不同结构的优化方法，比如本文的预取策略、节点放置策略。本文也是基于其中一个方向进行的扩展——即海量大规模数据节点情况进行最近邻搜索任务。本文其实优化效果最为显著的部分，是使用DSA进行领域协同加速，充分利用了CXL EP当中的PE，PE可以进行角度距离或者欧式距离计算，将高维数据向量的传输转变为了距离标量的传输，大大减少了CXL远内存特性以及传输延迟造成的影响，提升了QPS吞吐。同时还使用多个EP对于实际图搜索当中的高维海量数据进行切片操作，避免CXL后台DRAM带宽带来的瓶颈。

本文实验部分，鉴于缺乏公开可用、功能齐全的 CXL 系统，在实际运行系统中构建并验证了 CXL-ANNS 软件和硬件。还建立了一个代表 CXL-ANNS 的硬件验证的全系统模拟器用于评估，提升灵活性。进而顺利完成吞吐测试、任务分解、CPU使用情况、可扩展性测试等。这些优化和测试都需要作者对于CXL相关技术和硬件有较深的理解和运用实践基础，如果不具备CXL相关的基础知识，将难以实现CXL与图算法优化的交叉，无法搭建CXL内存池、实现DSA协同计算等。