关于改善数据中心内存效率的研究综述

郭韩飞[1)]

[1)](华中科技大学 计算机科学与技术, 武汉市 中国 430074)

摘 要 随着计算机硬件和软件技术的不断发展，带来了内存容量的增长并不足以跟上计算能力的增长的问题，也就是内存资源成为了数据密集型计算系统性能的瓶颈，并且当今数据中心应用程序如图形计算、数据分析和深度学习等对访问大量内存的需求越来越大，在服务器下的内存资源负担也越来越重并影响应用程序的性能，并且诸如内存重复数据删除等内存优化内核特性也加剧了这个情况，这也自然地使得数据中心必须寻找能够超出本地机器所能提供的内存资源，即使用远程内存。本文主要介绍一种利用智能网卡来卸载内存优化内核特性从而减少内存负担的方法，以及采用分离式内存架构来扩展内存并提高内存效率的一些方法和一种利用新兴 CXL 技术来实现内存池化的方法。

关键词 智能网卡；分离式内存；数据中心

A Review of Research on Improving Memory Efficiency in Data Center

GUO Han-Fei[1)]

[1)](Department of Computer Science and Technology, Huazhong University of Science and Technology, WuHan 430074)

Abstract With the continuous development of computer hardware and software technology, the problem of insufficient memory capacity to keep up with the growth of computing power has emerged, which means that memory resources have become a bottleneck in the performance of data-intensive computing systems.Moreover, today's data center applications such as graphic computing, data analysis, and deep learning have an increasing demand for accessing large amounts of memory. The burden of memory resources on servers is also increasing, which affects the performance of applications. Additionally, memory optimization kernel features such as memory deduplication have exacerbated this situation. This naturally makes it necessary for data centers to find memory resources that can exceed what local machines can provide - using remote memory. This article mainly introduces a method of using smart network cards to unload memory and optimize kernel features to reduce memory burden, as well as some methods of using a separate memory architecture to expand memory and improve memory efficiency, and a method of using emerging CXL technology to achieve memory pooling.

Key words Smart Network Interface Card; Disaggregated Memory; Data Center

1 引言

现代数据中心应用程序，如图形计算、数据分析和深度学习，对访问大量内存的需求越来越大。不幸的是，由于引脚、空间和功率限制，服务器面临着内存容量墙。并且由于 DRAM 技术的扩展速度明显放缓，导致每 GB DRAM 成本的下降停滞不前；而且数据中心服务器所需的 DRAM 容量迅速增长，不仅用于应用程序，还用于软件包、分析、日志记录和其他有效部署应用程序所需的支持功能(即数据中心内存负担)。

而现代操作系统提供了各种内核特性，可以提高系统的整体利用率和/或性能。其中，内存优化内

收稿日期：2024年1月1日

核特性，如内存重复数据删除、压缩缓存等致力于更有效地利用系统有限的 DRAM 容量。而具体来说，这些功能不经常被调用，但它们经常执行内存和/或 cpu 密集型操作，因此会明显地破坏应用程序的执行，密集地消耗服务器 CPU 的周期并污染其缓存，从而会造成严重的干扰，特别是在服务器 CPU 核心和缓存上共同运行内存密集型/延迟敏感的应用程序时。这导致数据中心中应用程序的高百分位数延迟大幅增加，因此为了改善内存利用率，可以进行内存优化内核功能的卸载操作，在不会降低内核功能的有效性的同时，而对协同运行应用程序的性能的干扰大大减少。

另一方面就是既然 DRAM 资源有限，数据中心需要寻求本地内存之外的内存资源，由此产生了对远程内存的需求，也就是对分离式内存研究的探索，也就是将计算资源和内存资源进行分离，让计算节点和内存节点之间通过网络互连或者 CXL 技术互连，核心思想就是通过计算和内存的解耦来弹性扩展分离的数据中心的计算和内存资源，来大大提高资源利用率、硬件异构性、资源弹性和故障处理能力。而现在的研究可以主要分为三类：首先是依赖虚拟内存系统来实现基于传统分页式的分离式内存，这种方法拥有作为缓存的本地内存，主要是依赖页面故障来进行页面交换，将远程的分离内存池作为交换内存区域；然后是一种基于对象的方式进行分离式内存管理和使用，比如使用键值存储内存数据库来处理分离式内存，这种方法可以解决地址转换所带来的挑战（例如，页面错误、上下文切换和写放大），但它需要重大的源代码级修改和接口更改；还有利用新兴的 CXL 互连技术来将内存池直接连接到计算节点端来扩展内存容量，主机端的用户可以直接用访存指令来访问。

本文主要是对一种利用智能网卡来卸载内存优化内核功能、以及对基于虚拟内存系统和 KV 存储内存数据库和 CXL 直连内存池的分离式内存管理方法进行介绍。

2 关于 STYX

这一部分主要对 STYX[1]进行介绍，它是一种基于智能网卡进行内核功能卸载的框架。

它主要是利用 SNIC 的计算资源和内存资源对原本在服务器 CPU 和缓存以及内存上执行的内存优化内核功能进行卸载操作。

文章中主要对 ksm 和 zswap 内核功能进行研究，首先介绍一下这两个功能：ksm 是一种重复数据删除功能，通常用于基于内核的虚拟机(KVM)，通过在多个虚拟机之间共享具有相同内容的页面，在给定的物理内存容量内快速整合更多虚拟机，它会定期扫描两个或多个正在运行的进程的页面，以识别具有相同内存内容的进程，ksm 的开销和收益都取决于每次调用 ksm 的扫描页面数、扫描页面的频率和合并页面的最大数量；zswap 充当 Linux 页面交换守护进程（kswapd）的压缩后端，用于对换出的页面进行压缩，然后存放在 DRAM 中动态分配的内存池（即 zpool）中；当 zpool 的大小达到阈值时，zswap 会从 zpool 中获取 LRU 页面，将其解压缩并重新存放到备用交换设备，并从 zpool 中释放压缩的页面；为了解决页面故障，zswap 首先检查 zpool，以确定页面是否被逐出到备用交换设备；如果在 zpool 中找到该页面，则只需解压缩该页面并由 zswap 返回。

这两个功能都是计算和内存密集型的操作，会严重消耗服务器 CPU 周期和降低最后一级缓存命中率。服务器中的 SNIC 可以通过 RDMA 访问服务器的本地内存，因此论文中提出了一个 STYX 卸载框架选择将这些操作的计算和所用到的数据都卸载到智能网卡上，来减少这些负增益对服务器系统性能的降低。设计 STYX 的基础是一个关键观察，即内存优化内核功能类似于网络应用程序，可以分解为控制和数据平面。然后，我们将内核功能的 CPU 和内存密集型操作分配给数据平面，并让 SNIC 的 CPU 处理数据平面。这有助于 STYX 在不损害其优点的情况下显著降低部署内核功能的成本。

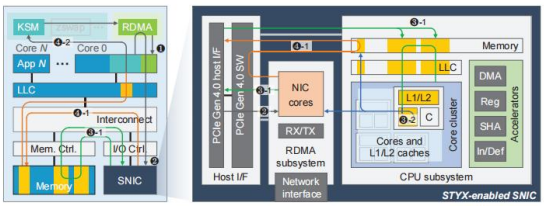

图 1 使用 styx 框架卸载内核功能的工作流过程

利用 STYX 卸载操作的工作流如图 1 所示，主要分为了四个步骤：①：STYX 首先确定将被卸载到 SNIC 中的函数，STYX 通过在服务器和 SNIC 之间建立 RDMA 连接来建立它们之间的通信接口。②：服务器上的 STYX 先更新与要执行的函数相关的描述符（为执行特定卸载功能的 SNIC 提供必要

信息）；然后通过执行 RDMA send 请求来执行拷贝服务器内存区域到 SNIC 的内存区域的操作；在服务器获得 SNIC 的函数结果之前，将暂停内核功能的执行，并且可以释放 CPU，让其他应用程序使用。③：在收到来自②步骤的服务器的 RDMA send 请求后，SNIC 开始执行拷贝服务器的内存区域到对应的 SNIC 的内存区域的操作，一旦复制操作完成，SNIC 上的 STYX 就会创建一个工作线程来执行要在 rdma 复制的内存区域上操作的函数。④：在完成函数的远程执行之后，SNIC 上的 STYX 将结果发送到服务器的 STYX，服务器的 STYX 接收到结果后，使内核功能从服务器内存读这些结果，然后继续执行内核功能。

并且修改并实现了两个功能的算法流程，代码逻辑就是将原本的函数调用修改为基于 STYX 执行的函数调用；并且在开启基于 STYX 实现的内核功能后同时运行内存密集型/延迟敏感的应用程序，来证明了 STYX 的有效性，在 ksm 和 zswap 执行功能的有效性和性能的相比原本情况下基本不影响功能执行的同时，发现使用基于 styx 的 ksm 和 zswap 的系统的延迟值比使用传统 ksm 和 zswap 的系统分别都有有效的降低。基于 STYX 的 zswap 内存优化内核功能在保留了原本功能优势的同时，也显著减少了由它引起的对协同运行应用程序的干扰。

3 MIND：在网络上进行内存管理

文章主要是考虑到此前的分离式内存的工作为了减少内存一致性维护的流量而降低性能，从而将分离式内存的共享限制在单个计算节点极大约束的资源的弹性使用。而网络中集中内存管理允许带宽和延迟高效地实现网络内缓存一致性协议，因此将内存管理逻辑放在可编程交换机上来实现共享内存抽象，MIND[2]就是一种将分离的内存管理逻辑和相关元数据放置在可编程交换机中来实现分离式内存管理和不同计算结点内应用的内存共享。

它的关键思想是将内存管理的逻辑和元数据放在网络结构中，因此集中放置的网络处理设备如可编程网络交换机，承担了内存管理单元 MMU 的角色，以实现高性能的共享内存抽象。它主要关注机架规模上的内存分解，其中内存和计算节点由单个可编程交换机连接。论文工作的范围限制在部分内存分解，也就是大多数内存是网络连接的，但 CPU 节点拥有少量（几个 GBs）的本地 DRAM 作为缓存。

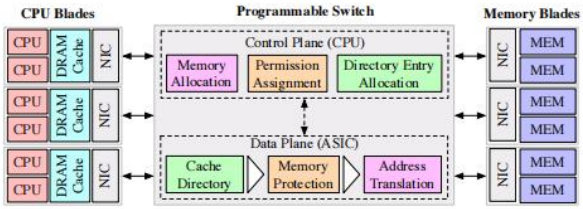

图 2　MIND 架构图

图 2 展示了 MIND 设计的概述图，划分控制平片功能和数据平面功能，并且将地址翻译和内存保护进行解耦、缓存访问粒度和缓存一致性执行粒度进行解耦。

来自用户进程的所有内存分配都在 CPU 刀片上拦截，并转发到交换机控制平面。控制平面拥有系统的全局视图，它利用这个视图来执行内存分配、权限分配等，并响应用户进程。所有来自用户进程的内存 LOAD/STORE 操作由 CPU 刀片缓存处理。缓存是虚拟寻址的，并存储缓存页面的权限，以实施内存保护。如果页面没有本地缓存，则 CPU 节点会触发页面故障，并使用 RDMA 请求从内存节点中获取该页，必要时清除其他缓存的页面（也就是基于页面级别的交换，把分离的内存当做了交换区来访问）。由于 CPU 节点不存储内存管理元数据，因此 RDMA 请求是针对虚拟地址的，并且不包含保存该页面的内存刀片的端点信息（例如，IP 地址）。因此，交换机数据平面拦截了这些请求。然后，它执行必要的缓存一致性逻辑，包括对缓存目录的查找/更新，以及在其他 CPU 节点服务器上的缓存无效。并且 MIND 利用单边 RDMA 操作来避免在分解的内存节点上的对 CPU 处理的需要。

关于地址翻译和内存保护的解耦

MIND 的翻译是基于内存结点的，而保护是基于 vma 的。MIND 避开了基于页面的保护，但跨所有进程使用单一的全局虚拟地址空间，允许在它们之间共享翻译条目。它将虚拟地址空间划分到不同的内存节点上，这样整个虚拟地址空间就会映射到一个连续的物理地址空间范围（也就是在特定的内存结点上虚拟地址和物理地址是一对一的）。这允许为每个内存节点使用一个转换条目：任何在其范围内的虚拟地址都可以直接路由到该内存刀片，从而最大限度地减少在交换机数据平面上所需的存储。在 MIND 中，只有当新内存结点连接、旧结点退出或内存在结点之间移动时，此映射才会更改（此外，MIND

为对应于静态虚拟地址或迁移内存的物理内存区域维护单独的基于范围的地址转换）。内存分配的结果是一个虚拟内存区域（vma），由该区域的基本虚拟地址和长度标识。

由于 MIND 解耦了地址翻译和内存保护，它使用一个单独的表来存储数据平面中的内存保护条目，因此，应用程序可以为任何大小的 vma 分配访问权限，这个保护表的大小与 vma 的数量成正比，并且使用保护域和权限等级的抽象来实现细粒度，灵活的内存保护。

缓存访问的粒度与执行缓存一致性的粒度解耦

缓存访问粒度和执行缓存一致性的粒度进行解耦，这允许在更细的粒度上执行内存访问，而在更粗的粒度上跟踪目录条目。具体来说，在 CPU 刀片上访问本地 DRAM 缓存，甚至在 CPU 缓存和内存刀片之间的数据移动，都发生在精细的页面粒度上；然而，一致性协议以更大的、可变大小的区域粒度跟踪目录条目（存储在交换机数据平面上）——当 4 KB 页面缓存在 CPU 刀片上时，MIND 为包含该页面的区域创建一个目录条目。该区域的失效会触发该区域中所有脏页的失效。

区域大小仍然暴露了一致性的性能问题（例如，由于更大的区域大小导致更大的错误无效计数）和目录存储效率（例如，由于较小的区域大小导致更多的目录条目）之间的紧张关系。为了适当地调整区域，MIND 利用交换机控制平面的内存流量的全局视图。简单地说，MIND 用一个非常大的内存区域设置初始时的每个目录条目；而当由于错误无效而导致的开销很高时，它将使用一个有界分割算法来对半分割该区域，并创建一个新的目录条目，重复这类分割操作，直到开销低于预定的阈值，或者区域大小达到 4 KB，即页面大小。

由于访问粒度和内存分配粒度的不匹配，实现缓存一致性时存在错误无效行为，有界分割算法的实现，其实就是为了应对设计的交换机以大于页面粒度的区域粒度跟踪（缓存一致性目录协议）目录条目方式而造成的大量错误无效的过大开销问题，对错误无效量过大的区域进行对半分割，增加缓存区域目录条目来降低错误无效区域的开销。

4 关于 Clio 分离式内存系统

与前面的 MIND 类似的是，Clio[3]也是提供分离式内存作为远程内存池，不同的是 MIND 的远程内存访问是基于页面故障的页面交换方式，对应用透明；但是 Clio 是提供对应用不透明的显式 API 调用（计算节点端的名为 CLib 的用户空间库）来让应用直接分配和访问内存结点上的分离式内存。

Clio 是一个软硬件联合设计的分离式内存系统，但是它基于对单一交换机可能成为性能瓶颈和扩展瓶颈的考虑，在内存结点上添加硬件来增加其计算能力（定制设备），因此在内存结点上的执行远程内存分配（包括虚拟和物理地址的分配）和内存访问的地址翻译功能。

即使线程和进程不在同一个计算结点上，它们可以共享数据；并且应用程序可以选择缓存从内存结点处读取的数据，在远程虚拟内存地址空间（RAS）中共享数据的不同进程可以在不同的计算结点（CN）上拥有自己的缓存副本。但是 Clio 不会使这些缓存的副本自动具有一致性，并允许应用程序选择它们自己的一致性协议，也就是把缓存数据一致性的维护让编程人员负责（因为每次读/写上的自动缓存一致性会给商品以太网基础设施带来高性能开销，而应用程序语义可以降低这种开销）。

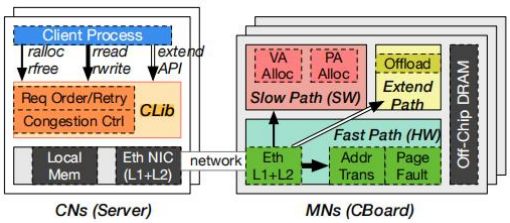

图 3 Clio 架构图

正如图 3 所示，CN 是常规服务器，每个服务器都配备了常规的以太网网卡，并连接到机架顶部（ToR）的交换机。MN 是我们直接连接到 ToR 交换机的定制设备，应用程序运行在基于用户空间库 Clib 上的 CN 上。MN 用 FPGA 实现的快速路径来处理所有数据访问的硬件逻辑、用 ARM 处理器实现处理元数据和控制操作的慢速路径，以及以及一个托管应用程序计算卸载的 FPGA（即扩展路径）。一个传入的请求到达 ASIC（专用集成电路），并通过标准的以太网物理层和 MAC 层，以及一个匹配和操作表（MAT），该表根据请求类型决定请求应该访问的三条路径中的哪一条。如果请求是一个数据访问（快速路径），它将停留在 ASIC 中，并通过一个基于硬件的虚拟内存系统，该系统在同一管道中执行三个任务：地址转换、权限检查和页面故障处理。然后，通过存储器控制器执行实际的存

储器访问，并通过网络堆栈发送响应。诸如内存分配等元数据操作被发送到慢速路径。

这种提供显式 API 调用的远程分离式内存的内存管理是以有计算能力的内存结点为中心，但与传统的虚拟内存系统类似，使用固定大小的页面作为地址分配和转换单元，而数据访问则以字节为粒度。

考虑到如果每个进程都有自己的页表，那么 MN 将需要缓存和查找许多页表根，从而导致额外的开销；其次，多级页表设计需要在发生 TLB 丢失时进行多次 DRAM 访问。在内存分离环境中，TLB 缺失将更为常见，因为有更多应用程序共享一个 MN，总的工作集大小将比单服务器设置中的大得多，而 MN 中的 TLB 大小将与单个服务器的 TLB 相似甚至更小(出于成本考虑)。更糟糕的是，每次访问 DRAM 对于 RDMA NIC 这样的系统来说成本更高，因为 RDMA NIC 必须穿过 PCIe 总线才能访问主存中的页表。因此文中提出了一种新的基于无溢出散列的页表设计，它根据物理内存大小和边界地址转换设置总页表大小为最多一次 DRAM 访问。具体来说，就是将所有进程中的所有页表条目（pte）存储在一个哈希表中，该表的大小与 MN 的物理内存大小成正比。慢速路径上的软件执行分配请求，计算此地址范围内的虚拟页面的哈希值，并检查将它们插入到页表是否会导致任何散列溢出。如果是这样，它将进行另一次搜索可用的 VA。重复这些步骤，直到找到一个不会导致散列溢出的有效 VA 范围（缺点是不能保证一个特定的 VA 可以被插入到页面表中）。

对于内存结点的初始访问页故障处理，为了让其延迟更低，提出了一种异步设计方法，以使 PA 分配偏离性能关键路径。具体来说，在异步缓冲区中维护一组空闲物理页码，ARM 通过查找空闲物理页地址并保留它们而不实际使用页面来不断填充。在页面故障期间，页面故障处理程序只需获取一个预先分配的物理页面地址。（即使单个 PA 分配操作有一个重要的延迟，但生成 PA 和填充异步缓冲区的吞吐量也高于网络线速率，因此，快速路径总是可以及时地在异步缓冲区中找到自由的 PA）。

关于虚拟内存系统的工作方式，例如：第一步（远程内存分配）由慢路径处理，该慢路径通过在散列页表中找到一组可用的插槽来分配 VA 范围。慢速路径将新的 pte 转发到快速路径，该快速路径将它们插入到页表中。此时，PTE 无效。此 VA 范围将返回给客户端。当客户端执行第一次写入时，请求将转到快速路径，将会有一个 TLB 的错误，然后是一个获取的 PTE；由于 PTE 无效，因此将触发页面故障处理程序，它从异步缓冲区获取一个空闲的 PA，并建立有效的 PTE；之后，它将执行写入，更新页表，并插入 PTE 到 TLB，并继续原始的故障请求。

并且论文中为了降低成本，设计了一个为内存分离定制的非对称网络。主要思想就是只在 CN 上维护传输逻辑、状态和数据缓冲区；以及放松传输的可靠性，而是在内存请求级别强制排序和丢失恢复，以便 MN 的硬件流水线可以在数据单元到达时立即处理它们。

5 FUSEE：完全内存分离 KV 存储

与前面的工作不一样，这篇文章的工作不再是基于虚拟内存系统（靠页故障来访问内存），而是使用内存数据库技术，也就是在分离式内存架构上实现比如键值存储来进行分离式内存的管理和使用。而这需要编程人员以一种不同于传统的基于分页式的内存访问方式来访问内存进行编程，并且系统的性能因应用程序的语义而异。

FUSEE[4]就是一个基于分离式内存架构的完全内存分离的 KV 存储系统，也就是分离的思想不仅仅在于数据的分离，而且也在于索引和内存管理元数据的分离，也就是元数据应该被存储在内存池中，并直接由客户端而不是元数据服务器进行管理，以此消除元数据服务器的计算瓶颈问题。同样内存结点具有少量计算能力，计算结点具有少量作为缓存的内存。

它在不同的 MN 上复制多个索引副本来容忍 MN 节点的故障问题。而为了保证复制的索引的一致性，需要应对来及客户端的对索引的读写冲突和写写冲突，文章中提出采用快照复制协议来解决。简单来讲，就是要解决读写冲突，快照将复制的散列索引分割为一个主副本和多个备份副本，并使用备份副本来解决写冲突，因此，写入冲突期间的不完整状态只出现在备份副本上，并且主副本始终包含正确和完整的值；读者可以简单地阅读主副本中的内容，而不感知不完整的状态。对于写写冲突，快照采用了一种类似于共享寄存器协议的最后写者赢的冲突解决方案，根据在备份副本中冲突的写

者写入的值定义了三个冲突解决规则，这使客户端能够协作地决定写入冲突下的最后一个写者。

FUSEE 通过一个两级内存管理机制来解决负责在 MN 上的 KV 对分配、复制和释放内存空间的内存管理问题。关键思想是将以服务器为中心的内存管理任务拆分为在 NM 和客户机上运行的轻计算的粗粒度管理和重计算的细粒度管理。FUSEE 首先在多个 MN 上复制和分区 48 位内存空间，也就是它将内存空间分割到 2GB 的内存区域，并以一致的散列将每个区域映射到 r 个 MN 上。第一级是粗粒度的 MN 端内存块分配。每个 MN 将其本地内存区域划分为粗粒度的内存块，例如，16 MB，并在每个区域的前面维护一个块分配表。对于每个内存块，块分配表记录一个分配它的客户端 ID。客户端通过向 MN 发送分配请求来分配内存块。在接收到分配请求时，MN 从其一个主内存区域中分配一个内存块，将客户端 ID 记录在主区域和备份区域的块分配表中，并将内存块的地址回复给客户端。因此，粗粒度的内存分配信息在 r 个 MN 上复制，并可以承受 MN 故障。第二级是细粒度的客户端对象分配，它分配小对象来保存 KV 对；具体来说，客户端管理从 MN 分配的块，将内存块分割为不同大小类的对象，然后从适合它的最小大小类中分配一个 KV 对。

对于索引损坏问题，它采用常用的操作日志方式来恢复崩溃的客户端导致的部分索引修改问题。然而，构造操作日志会在 DM 上产生很高的日志维护开销，因为写入日志条目会增加对 KV 请求的关键路径上的远程内存访问。因此，它实际上是采用一种嵌入式操作日志的方法，将日志项嵌入到 KV 对中。嵌入式日志条目与其对应的 KV 对一起编写，并使用一个 RDMA_WRITE 操作，因此，就消除了持久化日志条目所需的其他 RTT。但是这样又会导致不能保持 KV 请求的执行顺序，因为日志项不再是连续的。为了解决这个问题，嵌入式操作日志方案维护每个大小类的链接列表，以按照 KV 请求的执行顺序组织客户端的日志条目。

6 关于基于 CXL 的分离式内存

这篇文章的工作是基于 CXL 技术，也就是英特尔主导的计算快速链接技术，来构建了一个可以直接用访存指令访问的分离的内存池架构 DIRECTCXL[5]，即直接将远程内存资源连接到主机的计算综合体，并允许用户通过纯粹的 load/store 指令来访问它们。

前面的分离式的内存管理架构仍然需要在远程内存节点端的计算资源，这是因为所有的 DRAM 模块及其接口都被设计为被动外设。而 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。因此，文中将 CXL 设备设计并实现为纯无源模块，每个模块都能够有许多 DRAM 内存和自己的硬件控制器。其 CXL 设备采用多个 DRAM 控制器，在传统的 DDR 接口上连接 DRAM 内存，而 CXL 控制器然后通过许多 PCIe 通道将内部的 DRAM 模块暴露给 FlexBus。设备的 CXL 控制器解析传入的基于 pcie 的 CXL 数据包，称为 CXL flits，将它们的信息（地址和长度）转换为 DRAM 请求，并使用 DRAM 控制器从底层的 DRAM 提供服务。

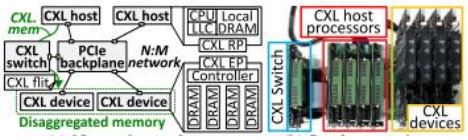

图 4 使用 CXL 的集群网络拓扑和实现

在主机端需要支持 CXL 技术的专用设备（比较昂贵并且现在商品化的推出还在施行中）。文中主要是利用 CXL 技术将设备内存整合到系统内存空间，并设计 CXL 网络交换机，以及设计软件运行时/驱动程序来管理底层的 CXL 设备，并在应用程序的内存空间中公开它们的地址空间。

相比于前面基于 RDMA 互连的分离式内存系统（页面交换或者 KV 存储），DirectCXL 没有大量数据移动，也不需要对某些工作负载的源代码级别进行修改，不需要与 RNIC 控制和相应的数据管理相关的硬件和软件的干预和开销。因此相比之下，没有网络传输带来访问数据和元数据的较高延迟影响，有天然的技术优势带来的性能优势。

但是这种内存扩展是基于单个计算节点扩充的，不同的 host 不可以共享同一个设备的内存空间，也就是没有在 host 登记的设备内存空间对 host 是不可见的，而前面的基于 RDMA 的分离式内存的地址空间对所有计算结点都可见。

7 总结与讨论

以上介绍的诸多工作都是为了解决数据中心内存资源不足以及利用率不高的问题，针对内存进行优化的内核功能来利用有限的内存资源是一种方法，提高内存利用率的是这些功能，而对这些功能进行卸载到网卡执行，是为了减少这些功能对系统性能的降低，也就是一种补充性优化系统性能的工作，并非是解决数据中心内存负担的方法，而且内存优化功能并不能本质上解决数据中心的大数据应用等对内存容量的大量需求问题。

而分离式内存思想是可以有效的对计算和内存资源进行弹性扩展，从而以一种形成内存池的方式来提升内存容量并且相比于传统的计算和内存紧密耦合的方式更能提高内存利用率，提升进程的可用内存地址空间。但是由于计算和内存分离，也就是二者“离得更远”，不可避免的带来传输延迟从而降低内存访问性能；以及与传统计算机的架构不同，在分离式内存架构下如何进行内存管理也是极具挑战性的，以往的内存管理方法不能很好地适用分离式内存场景，需要考虑新的解决办法。文中提到的三种类型的分离式内存的管理方式，都以不同的出发点来考虑分离式内存的管理，彼此都存在有不同的限制和要解决的问题且各有优劣。而现在主流就是基于 KV 存储或者新兴的 CXL 技术来实现分离式内存架构和进行相应的内存管理逻辑设计，分离的计算和内存结点之间连接方式也就是分别基于 RDMA 和 CXL 互连的方法，相信未来一定会出现在各自场景下解决相应问题的方法，来实现更加高效的分离式内存管理方法，以此来在尽量减少访问性能降低的同时用分离式内存池的大容量和可扩展性来满足数据中心对内存的不断增加的需求。

致　谢　*感谢施展老师、胡燏翀老师、童薇老师给我们讲授这门数据中心技术课程，课堂教授和论文汇报使我学到很多关于数据中心课程的知识概念，让我对数据中心有了一个更加清晰的认识，并且让我对数据中心所涉及的诸如对象存储、纠删码、固态存储技术有了进一步的了解，提升自己关于论文阅读和研究以及做汇报和讨论的经验。

参 考 文 献

[1] Houxiang Ji, Mark Mansi, Yan Sun, et al. Styx: exploiting smartnic capability to reduce datacenter memory tax: //Proceedings of the 2023 USENIX Annual Technical Conference. Boston, USA, 2023: 619-633.

[2] Seung-seob Lee, Yanpeng Yu, Yupeng Tang, et al. MIND: In-Network Memory Management for Disaggregated Data Centers: //Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. New York, USA, 2021: 488-504.

[3] Zhiyuan Guo, Yizhou Shan, Xuhao Luo, Yutong Huang, and Yiying Zhang. Clio: a hardware-software co-designed disaggregated memory system: //Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA, 2022: 417-433.

[4] Jiacheng Shen, Pengfei Zuo, Xuchuan Luo, et al. FUSEE: a fully memory-disaggregated key-value store: //Proceedings of the 21st USENIX Conference on File and Storage Technologies. Santa Clara, USA, 2023: 81-98.

[5] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, Myoungsoo Jung. Direct access, high-performance memory disaggregation with DirectCXL: //Proceedings of the 2022 USENIX Annual Technical Conference. Carlsbad, USA, 2022: 287-294.

附录X.

关于课程问题与回答内容。

Q：关于SmartNIC卸载操作的安全性问题，若是有人选择攻击网卡怎么处理？会不会产生数据泄露？

A：这确实是文章并没有考虑到的地方，不如说这其实是用网卡进行卸载操作的普遍存在的问题，而现在的研究其实主要关注性能方面的提升问题，因此安全问题就成了一个坑；因为网卡内存独立于操作系统管理的内存，智能网卡本身就是一个完整的系统，可以看做是一个单独的节点，确实收到攻击会存在数据泄露的问题，一个可能的方法就是在网卡上内置一些访问许可检测逻辑比如许可密钥等一些提高访问安全保护的方法。

Q：为什么要针对这两个内核功能？调用这两个功能既然会因此性能下降，那可不可以不用？

A：既然会引起性能下降，确实可以选择不用这两个优化功能，而且这两个功能中的zswap功能是默认关闭的，但是由于为了更有效地利用系统有限的DRAM容量，这些内核特性对谷歌、亚马逊、Meta和微软等拥有超大规模数据中心的厂商很有吸引力，因此这些功能对于它们来说是会选择开启的，而基于这种情况从而产生了对这些功能进行卸载到智能网卡上的想法，从而来提升系统的性能和减少系统延迟；而另一个分离式内存的架构思想可以扩展有限的容量，从本质上解决有限内存和内存利用率不高的问题，但是性能相比本地要低很多，因此怎么很好的进行分离式内存管理是极具挑战性的工作。