arXiv:2305.03943v1 [cs.DC] 6 May 2023

内存解聚：进展与开放挑战

Hasan AlMaruf，Mosharaf
Chowdhury 密歇根大学 SymbioticLab

摘要

在传统数据中心中，每台服务器内的计算与内存紧密耦合。大型数据中心运营商已发现，这种耦合是整个服务器集群资源利用率低下以及总体拥有成本（TCO）不断上升的根本原因之一。随着超高速网络和缓存一致性接口的出现，内存解聚 已成为一种潜在的解决方案，使应用程序能够利用服务器边界之外的可用内存。

本文从软件角度总结了内存解聚领域不断扩展的研究版图，并介绍了在当前及未来硬件趋势下实现其实用化所面临的挑战。我们还回顾了 SymbioticLab 七年来的探索历程：通过超高速网络构建全面的解聚内存系统。最后，我们探讨了利用新兴缓存一致性互连构建下一代内存解聚系统所面临的一些开放挑战。

1 引言

现代数据中心应用程序——低延迟在线服务、大数据分析以及 AI/ML 工作负载——通常都属于内存密集型应用。随着用户数量增加，以及我们在云数据中心收集更多数据，这些应用程序对内存的总体需求持续上升。尽管内存密集型应用具有性能优势，但当其工作集无法完全装入可用内存时，性能往往会出现*disproportionate*损失。例如，我们对一系列内存密集型应用进行测量后发现，如果其工作集有一半无法装入内存，性能可能下降 8× 至 25× [22]。

应用程序开发者通常会通过过量分配内存来规避此类灾难，但普遍存在的过量分配不可避免地会导致数据中心规模的内存利用率低下。事实上，许多超大规模云服务商的内存利用率徘徊在 40%–60% 左右 [1, 22, 25, 40]。在公有云上运行的服务提供商（例如 Snowflake）报告称，平均有 70%–80% 的内存未得到充分利用 [48]。由于动态随机存取存储器是基础设施成本和功耗的重要驱动因素 [33]，过度的利用率低下会导致总体拥有成本（TCO）居高不下。

与此同时，提升每台服务器的有效内存容量和带宽，以容纳不断增大的工作集，也颇具挑战。事实上，如今内存带宽比内存容量更容易成为瓶颈，因为前者的增长速度较慢。例如，Meta 在其数据中心将内存带宽提升 3.6×，需要将容量提升 16× [33]。为了提供充足的内存容量和/或带宽，传统服务器平台中的计算和网络资源会遭到闲置，最终导致整个机群的资源利用率降低，并增加总体拥有成本（TCO）。

内存解聚 从软件和硬件层面解决与内存相关的资源配置不当问题。应用程序可以根据需要分配内存，不受服务器边界限制。如果服务器只需要增加内存容量或带宽，就不必被迫添加更多计算和网络资源。通过将所有服务器上未使用的内存作为内存池提供给所有内存密集型应用，内存解聚既能提升应用程序级性能，也能提高整体内存利用率。多家硬件供应商和超大规模云服务商预测，通过（机架级）内存解聚，TCO 最多可节省 25%，且不会影响应用程序性能。</final>

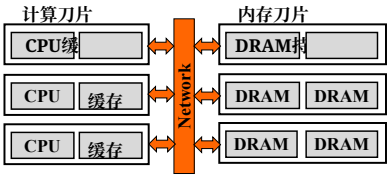
尽管利用远程机器的内存这一理念已有数十年历史 [15, 18, 20, 29, 31, 35], ，但直到过去几年，内存与通信技术之间的延迟和带宽差距才缩小到足以使其变得切实可行。最早的解聚内存[1] 解决方案（Infiniswap [22] 及其他方案）利用 InfiniBand 或以太网上的远程直接内存访问，但其速度比本地内存慢一个数量级。为了弥合这一性能差距，并解决性能隔离、韧性、可扩展性等实际问题，我们构建了一套全面的软件解决方案。最近，随着缓存一致性计算快速链路 (CXL) [3]互连和硬件协议的兴起，这一差距进一步缩小。我们正处于一个关键时刻，即将迈出一大步，走向下一代软硬件协同设计的解聚内存系统。

这篇短文既是一份快速教程，也是对 Infiniswap 项目的回顾，总结了七年的研究成果，并基于迄今为止的经验列出了一份并不详尽的未来预测。

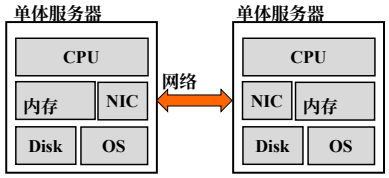
2 内存解聚

简单来说，内存解聚将远程位置可用的内存作为内存池提供，并通过网络在多台服务器之间共享。它将可用的计算资源与内存资源解耦，使集群能够独立分配资源。服务器的

[1]远程内存和远端内存通常与解聚内存一词互换使用。



（a）物理解耦



（b）逻辑解耦

图1：物理解耦与逻辑解耦架构。

本地内存和远程内存共同构成其物理内存总量。应用程序的内存访问局部性使服务器能够利用高速本地内存来维持高性能，而远程内存则提供了更大的容量，其访问延迟虽有所增加，但仍比访问持久化存储（如 HDD、SSD）快几个数量级。操作系统和/或应用运行时提供必要的抽象层，以开放集群中的全部可用内存，同时隐藏设置和访问远程内存的复杂性（如连接建立、内存访问语义、网络数据包调度等），并提供韧性、隔离、安全性等保障。

2.1 架构

内存解耦系统有两种主要的集群内存架构。

物理解耦。在物理解耦架构中，计算节点与内存节点彼此分离，计算刀片集群通过网络（例如 PCIe桥接器）连接到一个或多个内存刀片 [30]（图1a）。内存节点可以是计算资源较少但内存容量较大的传统单体服务器，也可以是网络附加DRAM。为获得更好的性能，计算节点通常会配备少量内存，用于缓存用途。

逻辑解耦。在逻辑解耦架构中，承载计算和内存资源的传统单体服务器通过网络（例如 InfiniBand、RoCEv2）彼此连接（图1b）。这是构建解聚内存系统的一种常见方法，因为无需更改现有硬件架构；只需加入适当的软件以提供远程内存接口即可。在这种设置中，通常每台单体服务器都有自己的

表1：部分内存解耦方案。

抽象层	系统	硬件透明	OS 透明	应用程序透明
虚拟内存管理（VMM）	全局内存 [19]	Yes	No	Yes
	内存刀片 [30]	No	No	Yes
	Infiniswap [22]	Yes	Yes	Yes
	Leap [32]	Yes	No	Yes
	LegoOS [44]	Yes	No	Yes
	zSwap [25]	Yes	No	Yes
	Kona [14]	Yes	No	Yes
	Fastswap [12]	Yes	No	Yes
	Hydra [27]	Yes	Yes	Yes
虚拟文件系统（VFS）	内存分页器 [31]	Yes	Yes	No
	远程区域 [11]	Yes	Yes	No
定制 API	FaRM [17]	Yes	Yes	No
	FaSST [24]	Yes	Yes	No
	Memtrade [34]	Yes	Yes	No
编程运行时	AIFM [42]	Yes	Yes	No
	Semeru [49]	Yes	Yes	No

操作系统。在某些情况下，操作系统本身也可以跨多个主机进行解聚 [44]。主机本地内存通常优先用于运行本地作业。远程机器上未使用的内存可以汇集起来，并作为远程内存提供给集群 [14, 22, 27, 32, 34, 42, 44]。

混合方案。诸如 CXL 这类缓存一致性互连，为构建可组合的异构内存系统创造了机会，使其能够结合逻辑解聚和物理解聚方案。多个单体服务器、计算设备、内存节点或网络专用设备可以通过互连结构或交换机连接，软件栈则可提供以缓存行为粒度或基于传统虚拟内存的解聚内存抽象。

2.2 抽象与接口

访问解聚内存的接口可以对应用程序透明，也可以要求对应用程序进行少量乃至彻底的重写（表1）。前者适用范围更广，而后者可能性能更好。

应用透明接口。无需大幅重写应用程序即可访问远程解聚内存，通常依赖两种主要机制：解聚虚拟文件系统（VFS） [11],，将远程内存作为文件提供；以及用于远程内存分页的解聚虚拟机监视器（VMM） [22, 27, 32, 44]。在这两种情况下，数据都以小块或内存页（通常为 4KB）的形式传输。对于作为文件的远程内存，内存页在写入远程内存或从中读取之前，会经过文件系统。对于远程内存分页和分布式操作系统，缺页异常会促使虚拟机监视器将内存页写入远程内存或从中读取。远程内存分页更适合传统应用程序，因为它不需要软件或硬件修改。

非透明接口。另一种方法是通过自定义 API（键值存储、远程内存感知库或系统调用）直接公开远程内存，并

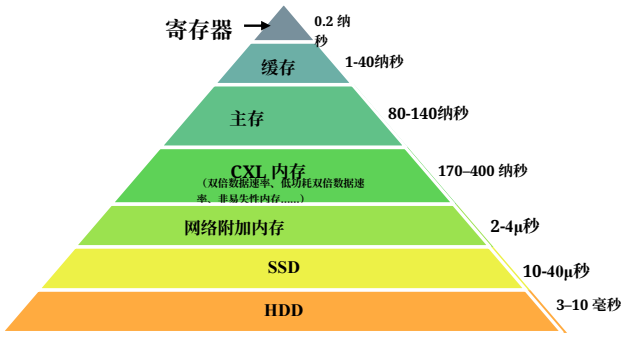


图2：不同内存技术的延迟概况。

修改应用程序以集成这些特定的 API [17, 24, 34, 41, 42, 49]。所有内存分配与释放、事务、同步等操作均由这些 API 的底层实现处理。缓存、本地与远程数据放置、预取等性能优化通常由应用程序处理。

2.3 实际内存解聚中的挑战

仅仅依赖高速网络或互连，不足以实现实用内存解聚。全面的解决方案必须应对多个维度上的挑战：

• **高性能**。 解聚内存系统在其远程内存路径中涉及网络，其速度至少比连接到 CPU 和动态随机存取存储器的内存通道慢一个数量级（80–140 纳秒与微秒级相比；参见图2）。硬件引起的远程内存延迟十分显著，并会影响应用程序的性能 [22, 32, 33]。根据抽象层的不同，软件栈也可能引入显著的开销。例如，通过现有的虚拟机监视器（VMM）进行远程内存分页，可能会使一个 4KB 页面增加数十微秒的延迟 [32]。

•**性能隔离**。 当具有不同性能需求的多个应用程序（例如，对延迟敏感与对带宽敏感的工作负载）竞争使用分离式内存时，具体情况取决于应用程序的运行位置和远程内存的位置；它们可能会在硬件侧争用服务器、网络接口控制器和网络中的资源，也可能会争用应用运行时和操作系统中的各种资源。如果存在多个延迟与带宽特性各异的内存层级，这种情况会进一步加剧。

• **内存异构性**。 服务器内的内存层次结构本身已经具有异构性（图2）。解聚内存——包括网络连接内存和新兴的 CXL内存 [21, 28, 33] ——进一步加剧了延迟与带宽特性方面的异构性。在这种设置下，仅向应用程序分配内存是不够的。何时在何种层级分配多少内存等决策也至关重要。

•**对扩展故障域的韧性**。 依赖远程内存的应用程序容易受到新的故障场景影响，例如远程机器发生独立故障和相关故障、远程内存被驱逐或损坏，以及网络分区。此外，由于网络拥塞和后台流量，它们还会受到掉队节点或远程响应延迟到达的影响 [16]。这些不确定性可能导致灾难性故障，并违反服务级目标（SLO）。

• **效率与可扩展性**。 解聚内存系统本质上是分布式的。随着内存服务器数量、解聚内存总量以及应用程序数量的增加，在大型集群中查找未分配的远程内存、在不违反应用程序特定服务级别目标（SLO）的前提下将其分配给应用程序，以及内存管理相应的元数据开销，都会变得更加复杂。要实现较高的总体利用率，就必须能够大规模高效匹配。

• **安全性**。 尽管在私有数据中心环境中，解聚内存的安全性问题往往被忽视，但对于公有云中的内存解聚而言，这却是一项重大挑战。远程内存中的数据可能被未获适当授权的实体读取，也可能因意外或恶意行为遭到破坏，因此必须保护远程内存的机密性和完整性。其他顾虑还包括网络上的侧信道攻击和远程 Rowhammer 攻击 [45, 46], 、分布式协调攻击、数据缺乏机密性和完整性，以及 CPU 绕过操作期间缺乏客户端可追责性（例如，使用远程直接内存访问进行内存解聚时）。

3 Infiniswap：回顾

据我们所知，Infiniswap 是首个针对上述所有挑战提供全面且协调一致的解决方案集的内存解聚系统。它从主机级、网络级和端到端层面，解决了基于远程直接内存访问（RDMA）的实际内存解聚问题。总体而言，Infiniswap 提供了一种基于分页的远程内存抽象，无需修改即可适用于任何应用程序，同时为远程解聚内存提供高性能、弹性、隔离且安全的数据路径。

引导。 我们的探索始于 2016年，当时我们只是专注于构建一个对应用程序透明的远程内存访问接口，这些远程内存分布在多台服务器上。Infiniswap [22] 通过分页机制，以透明的方式暴露远程解聚内存，无需对各个服务器上的应用程序、硬件或操作系统进行任何修改。它向虚拟机监视器（VMM）提供了一个采用传统 I/O 接口的块设备。该块设备将整个地址空间划分为更小的内存块，并将它们透明地映射到多台服务器上。

远程内存。Infiniswap 在运行时捕获 4KB 页面错误，并使用远程直接内存访问将其重定向到远程内存。

从一开始，我们就希望设计一个能够扩展且不会在后续失去效率的系统。为此，我们设计了去中心化算法，用于识别空闲内存、分配内存块，以及为内存回收驱逐内存块。这消除了集中式元数据管理的开销，同时保持了效率。

提升性能。Infiniswap 基于块层的分页在远程内存访问期间造成了较高的延迟开销。这是因为 Linux 虚拟内存管理器并未针对微秒级操作进行优化。我们放弃了一定的自由度，并于 2018年设计了 Leap [32]——通过识别并移除分页期间的非关键操作，针对远程内存数据路径优化操作系统。

即使采用最精简的数据路径，响应式页面获取系统也必须在关键路径上承受微秒级的网络延迟。Leap 引入了远程内存预取器，主动将正确的页面加载到本地缓存中，从而在缓存命中时提供亚微秒级延迟（与本地页面访问的延迟相当）。

提供韧性。 Infiniswap 最初依赖本地磁盘来容忍远程故障，这导致故障恢复速度缓慢。维护多个内存副本也不可行，因为这实际上会使总容量减半。我们开始探索将纠删码作为一种节省内存的替代方案。具体而言，我们将每个页面划分为k 个分片，生成 r 个编码校验分片，并将这（$k+r$）个分片分布到（$k+r$）个故障域中——随后只需（$k+r$）个分片中的任意 k个即可解码出原始数据。然而，纠删码传统上应用于大型对象 [38]。到 2019/20 年，我们构建了 Hydra [27] ，其经过精心设计的数据路径能够以个位数微秒级的尾延迟执行在线纠删码。Hydra 还引入了 CodingSets，这是一种新的数据放置方案，在提升可用性的同时实现了负载均衡；即使发生大规模相关故障，也能将数据丢失的概率降低一个数量级。

多租户问题。早在 2017年 左右，我们就观察到，通过共享网络访问远程内存会受到网络接口控制器中的竞争以及网络内部竞争的影响[54]。虽然 Leap 和 Hydra 中经过优化的数据路径能够解决主机内部的一些挑战，但无法延伸到远程直接内存访问网络接口控制器中的资源竞争（RNIC）。我们于2020年设计了 Justitia [56] ，通过透明地监控每个应用程序的延迟特征并提供性能隔离，来改善 RNIC 中的网络瓶颈。最近，我们也开始研究如何提升网络内部的服务质量（QoS）[55]。

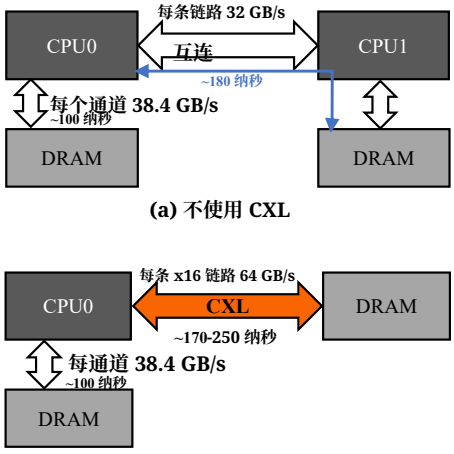


图 3：CXL 系统与双路服务器的对比。

扩展到公有云。 虽然 Infiniswap 及相关项目是为协作式私有数据中心设计的，但公有云中的内存解聚面临着其他问题。2021年，我们完成了 Memtrade[34] 的设计，用于回收虚拟机（VM）中的所有闲置内存——无论是未分配的，还是已分配给应用程序但很少使用的内存——并将其开放至解聚内存市场。Memtrade 允许生产者虚拟机在有限时间内将其闲置的应用程序内存租给远程消费者虚拟机，同时确保机密性和完整性。它采用经纪器在生产者与消费者之间进行匹配，同时满足性能约束。

一路上的探索。 在这段历程中，我们还合作开展了一些支线项目，例如利用 RDMA 原语[51], 在网络内部使用可编程交换机进行元数据管理 [53], 细粒度计算解耦 [52] 等。我们曾尝试设计硬件支持，但不少尝试很早便夭折，往往是因为找不到合适的合作伙伴。回过头来看，考虑到行业如此迅速地转向 CXL，也许我们算是幸运的。

总结。 Infiniswap 及其所有扩展，即使在应用程序工作集有 75%（有时甚至更多）位于远程内存中的情况下，也能在存在故障、负载不平衡和多租户的环境中，以对应用程序和硬件透明的方式，为大多数内存密集型应用程序提供近内存性能。七年之后，我们在 2022年为这一篇章画上了句号。

4 HardwareTrend：缓存一致性互连

尽管 InfiniBand 和以太网等网络技术持续改进，但它们的延迟仍然相当高。

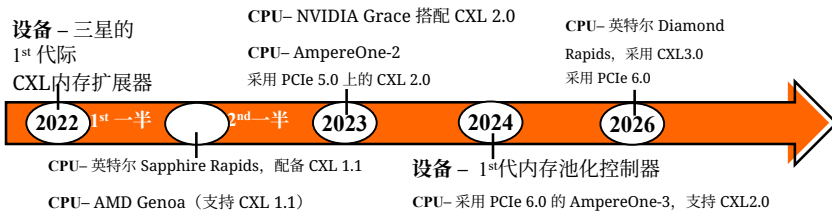


图4：CXL 路线图　　为下一代数据中心设计中的内存池化与解聚铺平道路　　.

要在解聚内存设备之间提供缓存一致性内存地址空间，要求很高。CXL（计算快速链路）[3] 是一种新的处理器到外设/加速器的缓存一致性互连协议，它以现有 PCIe 协议为基础并对其进行扩展，使连接的设备之间能够进行一致性通信。[2] 它在同一物理地址空间中提供字节可寻址内存，并支持使用标准内存分配 API 透明地分配内存。它还允许以缓存行粒度访问所连接的设备，而底层硬件则负责维护缓存一致性和一致性。采用 PCIe 5.0 时，CPU 到 CXL 互连带宽与双路机器上的跨插槽互连带宽相近（图 3）[57]。CXL 内存访问延迟也与 NUMA 访问延迟相近。与普通 DRAM 访问相比，CXL 会额外增加约 50–100 纳秒的延迟。

CXL 路线图。如今，支持 CXL 的 CPU 和内存设备支持 CXL 1.0/1.1（图4），可在 CPU 与加速器内存之间，或 CPU 与内存扩展器之间建立点对点链路。CXL 2.0 规范支持单跳交换，可将多个不带内存（*Type-1* 设备）或带内存（*Type-2* 设备）的加速器配置到同一台主机，并使其缓存与 CPU 保持一致。它还允许通过内存扩展设备（*Type-3* 设备）在多台主机之间实现内存池化。CXL 交换机配备结构管理器（可内置或外置），负责设备地址空间管理。设备可以热插拔至交换机。虚拟 CXL 交换机可对 CXL 内存进行分区，并隔离多台主机之间的资源。它会提供各个连接设备的负载遥测数据，以实现负载均衡和 QoS 管理。

CXL 3.0 新增了多跳分层交换，通过级联和扇出，可构建任意复杂类型的网络。这扩大了连接设备的数量以及互连结构的复杂性，使其能够支持非树形拓扑，例如脊叶架构、网格和环形架构。CXL 3.0 支持 PCIe 6.0（64 GT/s，即 x16 双工链路的吞吐量最高可达 256 GB/s），并拓展了复杂多样的可组合机架级服务器设计的前景。

[2]该领域此前的行业标准，例如 CCIX 和 [2], OpenCAPI [8],Gen-Z [5] 等都汇聚在 CXL 联盟旗下。尽管也有一些相关的研究提案（例如 [26]），但在撰写本文时，CXL 是行业事实标准。

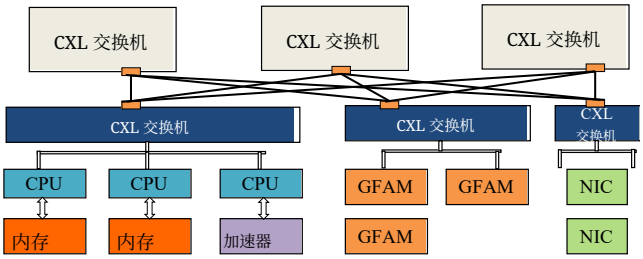


图 5：CXL 3.0 支持采用复杂网络和可组合内存层次结构的机架级服务器设计。

内存技术（图 5）。新的基于端口的路由（PBR）功能提供了可扩展的寻址机制，支持多达 4,096 个节点。每个节点可以是现有的三种设备类型之一，也可以是新的全局互连结构附加内存（GFAM）设备；该设备支持在单个设备中集成不同类型的内存（即持久内存、闪存、动态随机存取存储器及其他未来的内存类型等）。除了内存池化之外，CXL 3.0 还支持多个终端设备上的多个主机之间进行内存共享。连接的设备（即加速器、内存扩展器、网络接口控制器等）可以绕过主机 CPU 进行点对点通信。

本质上，CXL 3.0 支持构建由大量内存设备组成的大型网络。这将推动软硬件协同设计的内存解聚方案蓬勃发展，这些方案不仅能简化并更好地实现上一代解聚方案（例如 Infiniswap），还将开辟新的可能性。

5 服务器内 CXL 解聚

随着新硬件技术的出现，我们有机会重新思考并审视以往的设计决策，CXL 也不例外。此前基于远程直接内存访问的内存解聚软件方案，尚未针对 CXL 进行充分优化，因为 CXL 的延迟上限低得多；对于延迟为数百纳秒的服务器内 CXL（CXL 1.0/1.1）而言，这一点尤为明显。近期利用服务器内 CXL 1.0/1.1 的研究，重点关注分层内存池化 [28, 33]，因为数据中心应用程序的大部分工作集可以卸载到速度较慢的内存层，而不会损害性能 [25, 33, 34]。我们最近围绕这一背景下的两项基础性挑战开展了研究。

内存使用特征分析。数据中心应用程序对内存访问延迟和带宽的要求各不相同。对不同内存页面类型的敏感度

在不同应用程序之间也可能有所差异。了解并表征这些行为，对于设计异构分层内存系统至关重要。Chameleon [33] 是一款轻量级用户空间内存访问行为表征工具，可轻松部署于生产环境，无需中断正在运行的应用程序或修改操作系统。它利用现代 CPU 的性能监控单元（PMU）中的精确事件采样（PEBS）机制，收集与内存访问相关的硬件级性能事件。随后，它会生成不同页面类型的内存使用热力图，并深入揭示应用程序在采用多个温度层级时的预期性能。

内存管理。 根据应用程序的页面特征，TPP [33] 提供一种操作系统级的透明页面放置机制，以便在分层内存系统中高效放置页面。TPP 包含三个组件：**(a)** 一种轻量级回收机制，用于将较冷的页面降级到慢速层；**(b)** 解耦多NUMA系统中的分配与回收逻辑，以便在快速层维持一定的空闲内存页余量；以及 **(c)** 一种响应式页面提升机制，可高效识别滞留在慢速内存层级中的热页面，并将其提升到快速内存层级以提高性能。此外，TPP 还支持在不同内存层级中进行页面类型感知分配。

6 机架级及更大规模的 CXL解聚内存：开放性挑战

尽管延迟高于服务器内部的 CXL 延迟，但采用 CXL 交换机（CXL 2.0）的机架级 CXL 系统，其延迟将远低于基于远程直接内存访问的内存解聚。对于采用 CXL 3.0 的配置，经过少数几跳后，延迟最终将达到数微秒，与当今基于远程直接内存访问的解聚内存系统相近。对于运行在这两个极端之间（即机架级及稍远范围内）的下一代内存解耦系统，仍存在许多开放挑战。我们甚至可能需要重新审视过去的一些设计决策（§2）。在此，我们根据自身经验列出一些挑战，但并未穷尽所有问题。

6.1 抽象

内存访问粒度。CXL 支持通过互连设备进行缓存行粒度内存访问，而现有操作系统虚拟内存管理模块则是为页面粒度（通常为 4KB 或更大）的内存访问而设计的。在整个生命周期中，应用程序通常只会写入每个页面的一小部分；一般仅写入 64 [14] 中的 1–8 个缓存行。页面粒度访问会导致严重的脏数据放大和带宽浪费。相反，在大型内存池中进行细粒度内存访问，则会产生很高的元数据管理开销。根据应用程序的内存访问模式，远程内存抽象应支持对内存访问粒度进行透明、动态的调整。

内存QoS接口。 传统的内存页管理方案侧重于跟踪（部分）内存页并统计访问次数，以确定页面温度，然后将内存页迁移到其他位置。虽然这足以提供热页与冷页两级服务质量，但无法涵盖页面温度的完整范围。潜在的解决方案包括为以下对象分配 QoS 级别：(1) 整个应用程序；(2) 单个数据结构；(3) 单个 mmap()调用；甚至 (4) 单次内存访问。这些方法各有利弊。极端情况下，为整个应用程序分配 QoS 级别或许很简单，但无法捕捉大型、长期运行的应用程序中随时间变化的页面温度。另一方面，为单次内存访问分配 QoS 级别，则要求所有现有应用程序都重新编译，还需要繁琐的手动分配，容易导致 QoS 分配错误。结合上述方法，或许能在减轻开发者负担的同时，提供足够的灵活性来实现时空内存QoS管理。

6.2 管理与运行时

内存地址空间管理。从 CXL 2.0 开始，设备可以热插拔到 CXL 交换机中。设备附加内存会映射到系统的一致性地址空间，并可由主机通过标准写回语义访问。CXL 设备上的内存可以映射为主机管理的设备内存（HDM）或私有设备内存（PDM）。若要更新连接设备的内存地址空间，以便将其分配给不同的主机设备，则需要重置系统；在重置期间，必须停止发往该设备的流量，才能更改设备地址映射。避免系统重置的另一种方案是在 CXL 设备加入系统时，将整个物理地址空间映射到每台主机。CXL 交换机中的虚拟机监视器或结构管理器负责在地址空间管理期间维持隔离。如何将整个地址空间划分为大小合适的内存块，以便在大型内存网络中高效地进行物理地址到虚拟地址的转换，是一个值得探讨的挑战 [26, 53]。

用于计算解聚的统一运行时。 CXL Type-2 设备（带内存的加速器）与 CPU 保持缓存一致性。CPU 和 Type-2 设备可以互换使用彼此的内存，双方均能从中受益。例如，在 CPU 上运行的应用程序可以受益于现在能够访问带宽极高的 GPU 内存。同样，对于 GPU 用户而言，即使 CPU 内存的读写带宽较低，扩展容量仍然有益。在这种设置中，远程内存抽象应跟踪计算核心的可用性，并

高效执行近存计算，以提升整体系统吞吐量。

未来的数据中心可能会配备大量领域专用计算资源/加速器。在这样的系统中，可以借用某项计算资源的空闲核心执行额外计算，以提高整体系统吞吐量。支持可伸缩进程的统一运行时能够将进程立即拆解为更小的部分，并将其卸载到任何可用的计算节点，从而提升应用程序和集群的吞吐量 [41, 52]。

6.3 分配策略

异构 NUMA 集群中的内存分配。为了获得更好的性能，最热门的页面需要放置在速度最快的内存层级中。然而，受不同层级内存容量的限制，始终使用速度最快或性能最高的内存层级可能并不可行。要确定应用程序在生命周期的不同阶段需要在特定内存层级中分配多少内存，才能维持所需性能，是一项挑战。当多个应用程序共存时，这一问题更加棘手。必须在机架规模上高效地将不同热度的页面提升或降级到不同内存层级。可以考虑扩展 TPP，引入一种轻量而有效的算法，在选择迁移目标时综合考虑节点与 CPU 的距离、CPU-内存总线的负载、不同内存层级的当前负载、网络状态，以及迁移候选页面的服务质量要求。

内存带宽扩展的分配策略。对于内存带宽受限型应用，CPU 到动态随机存取存储器的带宽往往会成为瓶颈，并增加平均内存访问延迟。CXL 提供的额外内存带宽可通过将内存分布到顶层节点和远程节点来发挥作用。理想的解决方案不应只将带宽消耗较低的冷页放入 CXL 内存，而应将适量带宽需求高、对延迟不敏感的内存页放入 CXL 内存。识别此类工作集理想比例的方法甚至可能需要硬件支持。

内存共享与一致性。CXL 3.0 支持在多个设备之间共享内存。借助增强的一致性语义，多个主机可以持有共享段的一致副本，并通过反向失效实现同步。内存共享可以减少不必要的数据移动，从而提升应用程序级性能并提高内存利用率。然而，共享大规模内存地址空间会给系统带来显著的开销和复杂性，这正是经典分布式共享内存（DSM）方案所面临的问题 [36]。此外，在硬件空间中跨多个设备共享内存，会增加存在恶意应用运行时的安全威胁。我们认为，解聚内存系统应谨慎对待内存共享，除非特定场景确有必要，否则应避免共享。

6.4 机架级目标

机架规模内存温度。要了解应用程序在多个温度层级下的预期性能，就必须掌握该应用程序内存使用情况的热图。现有的热页识别机制（包括 Chameleon）仅限于单个主机操作系统或用户空间机制。它们要么使用基于访问位的机制 [4, 6, 47], ，要么使用基于特殊 CPU 功能（例如 Intel PEBS）的工具 [39, 43, 50], 或操作系统特性 [7, 33]，以确定单台服务器内的页面温度。迄今为止，还没有能够确定集群范围相对页面温度的分布式机制。如何整合所有操作系统工具或用户空间工具的数据，并协调这些工具以找出机架级热页，是一个重要问题。CXL 结构管理器或许是获取集群范围视图的理想位置，可查看每个 CXL 设备的负载、命中和访问相关硬件计数器。可以设想，将 Chameleon 扩展至机架规模环境，以便观测每个应用程序的设备级内存温度。

面向更优生态系统的软硬件协同设计。 在机架级部署中，硬件特性可以进一步提升解聚系统的性能。在 CXL ASIC 或交换机上配备内存侧缓存及其关联的预取器，或许有助于降低 CXL 内存的有效延迟。支持在不同内存层级之间移动数据，有助于降低页面迁移开销，尤其是在本地内存容量极小、CXL 内存容量较大的激进配置系统中。此外，CXL 交换机的结构管理器应实施公平排队、拥塞控制、负载均衡等策略，以改善网络管理。在 CXL 交换机设计中融入 Leap 的预取器和 Hydra 的纠删码韧性方案，可以提升系统整体性能。

能耗与碳排放感知型内存解聚。数据中心是能源消耗和碳排放的重要来源，且这一影响仍在不断扩大 [13]。一些估算认为，数据中心占全球总用电量的 1%–2% [23, 37]。为降低总体拥有成本（TCO）和碳足迹，并延长硬件的预期寿命，数据中心机架会设定物理能耗预算或功率上限。机架级内存分配、降级和提升策略可以将能耗感知纳入决策过程。总体而言，我们可以在软件栈中引入能耗感知，使其在解聚集群中管理计算、内存和网络资源。

7 结论

2016年，我们怀着内存解聚势在必行的信念启动了 Infiniswap 项目，当时手中只有少数几个数据点，暗示这一目标或许已触手可及。2023年，在结束本文之际，我们已成功构建了一个基于超高速 RDMA 网络的全面软件解聚内存解决方案，可为大多数内存密集型应用提供无缝体验。随着各种缓存一致性互连最终汇聚于 CXL 旗下，整个行业（包括我们自己）正处于迈向新一代软硬件协同设计解聚系统的门槛上。加入我们吧。内存解聚已然到来，并将长久存在。

致谢

Juncheng Gu、Youngmoon Lee 和 Yiwen Zhang 与作者共同领导了 Infiniswap 项目的不同方面。感谢 Yiwen Zhang 对本文提出的反馈。特别感谢众多合作者、贡献者、用户和云资源提供方（即 CloudLab、Chameleon Cloud 和 UM ConFlux），正是他们让 Infiniswap 得以成功。我们在探索下一代内存解聚解决方案的过程中，受益于与 Meta 的合作。我们的研究部分由美国国家科学基金会的资助项目（CCF-1629397、CNS-1845853 和 CNS-2104243）以及 VMware 和 Meta 的慷慨捐赠支持。

参考文献

[1] 阿里巴巴集群追踪 2018年。https://github.com/alibaba/clusterdata/blob/master/cluster-trace-v2018/trace 2018.md._[2] CCIX。https://www.ccixconsortium.com/。 [3] 计算快速链路 (CXL)。https://www.computeexpresslink.org/.[4] DAMON：数据访问监控框架，用于趣味和内存管理优化。https: //www.linuxplumbersconf.org/event/7/contributions/659/attachments/503/1195/damon_ksummit_2020.pdf。 [5] Gen-Z。https://genzconsortium.org/。 [6] 基于空闲页跟踪的工作集估算。https://lwn.net/ Articles/460762/.[7] NUMA 平衡 (AutoNUMA)。https://mirrors.edge.kernel.org/pub/linux/kernel/people/andrea/autonuma/autonuma_bench-20120530.pdf。 [8] OpenCAPI。https://opencapi.org/。 [9] Yahoo! 的机架级计算。http://www.intel.com/content/dam/www/public/us/en/documents/presentation/idf15-yahoo-rack-scale-computing-presentation.pdf。 [10] 腾讯使用英特尔机架级架构 (Intel RSA) 探索数据中心资源池化。http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/rsa-tencent-paper.pdf。 [11] M. K. Aguilera、N. Amit、I. Calciu、X. Deguillard、J. Gandhi、S. Novaković、A. Ramanathan、P. Subrahmanyam、L. Suresh、K. Tati、R. Venkatasubramanian 和 M. Wei。远程区域：远程内存的简单抽象。发表于 *USENIX ATC*，2018年。 [12] E. Amaro、C. Branner-Augmon、Z. Luo、A. Ousterhout、M. K. Aguilera、A. Panda、S. Ratnasamy 和 S. Shenker。远端内存能否提高作业吞吐量？发表于 *EuroSys*，2020年。

[13] T. Anderson、A. Belay、M. Chowdhury、A. Cidon 和 I. Zhang。Treehouse：碳感知数据中心软件的案例。载于 HotCarbon，2022年。[14] I. Calciu、M. T. Imran、I. Puddu、S. Kashyap、H. A. Maruf、O. Mutlu 和 A. Kolli。重新思考解聚内存的软件运行时。载于 ASPLOS，2021年。[15] H. Chen、Y. Luo、X. Wang、B. Zhang、Y. Sun 和 Z. Wang。面向虚拟机的透明远程分页模型。载于虚拟化技术国际研讨会，2008年。[16] J. Dean 和 L. A. Barroso。规模化系统中的尾延迟。Communications of the ACM，56(2)：74–80，2013年。[17] A. Dragojević、D. Narayanan、O. Hodson 和 M. Castro。FaRM：快速远程内存。载于 NSDI，2014年。[18] S. Dwarkadas、N. Hardavellas、L. Kontothanassis、R. Nikhil 和 R. Stets。Cashmere-VLM：软件分布式共享内存的远程内存分页。载于 IPPS/SPDP，1999年。[19] M. J. Feeley、W. E. Morgan、E. Pighin、A. R. Karlin、H. M. Levy 和 C. A. Thekkath。在工作站集群中实现全局内存管理。载于 ACM SIGOPS Operating Systems Review，第 29 卷，第 201–212 页。ACM，1995年。[20] E. W. Felten 和 J. Zahorjan。远程内存分页系统实现中的问题。技术报告 91-03-09，华盛顿大学，1991年3月。[21] D. Gouk、S. Lee、M. Kwon 和 M. Jung。直接访问：借助 DirectCXL 实现高性能内存解聚。载于 USENIX ATC，2022年。[22] J. Gu、Y. Lee、Y. Zhang、M. Chowdhury 和 K. G. Shin。借助 Infiniswap 实现高效内存解聚。载于 NSDI，2017年。[23] N. Jones。如何阻止数据中心吞噬全球电力。Nature，561：163–166，2018年。[24] A. Kalia、M. Kaminsky 和 D. G. Andersen。FaSST：借助双端（RDMA）数据报 RPC 实现快速、可扩展且简单的分布式事务。载于 OSDI，2016 年。[25] A. Lagar-Cavilla、J. Ahn、S. Souhlal、N. Agarwal、R. Burny、S. Butt、J. Chang、A. Chaugule、N. Deng、J. Shahid、G. Thelen、K. A. Yurtsever、Y. Zhao 和 P. Ranganathan。仓库规模计算机中的软件定义远端内存。载于 ASPLOS，2019年。[26] S.-s. Lee、Y. Yu、Y. Tang、A. Khandelwal、L. Zhong 和 A. Bhattacharjee。MIND：面向解聚数据中心的网络内存管理。载于 SOSP，2021年。[27] Y. Lee、H. A. Maruf、M. Chowdhury、A. Cidon 和 K. G. Shin。Hydra：具有弹性和高可用性的远程内存。载于 FAST，2022年。[28] H. Li、D. S. Berger、S. Novakovic、L. Hsu、D. Ernst、P. Zardoshti、M. Shah、S. Rajadnya、S. Lee、I. Agarwal、M. D. Hill、M. Fontoura 和 R. Bianchini。Pond：面向云平台的基于 CXL 的内存池化系统。载于 ASPLOS，2023年。[29] S. Liang、R. Noronha 和 D. K. Panda。通过 InfiniBand 将数据交换至远程内存：一种使用高性能网络块设备的方法。载于 IEEE 国际集群计算会议，2005年。[30] K. Lim、J. Chang、T. Mudge、P. Ranganathan、S. K. Reinhardt 和 T. F. Wenisch。刀片服务器中用于扩展和共享的解聚内存。SIGARCH，2009年。[31] E. P. Markatos 和 G. Dramitinos。可靠远程内存分页器的实现。载于 USENIX ATC，1996年。[32] H. A. Maruf 和 M. Chowdhury。借助 Leap 有效预取远程内存。载于 USENIX ATC，2020年。[33] H. A. Maruf、H. Wang、A. Dhanotia、J. Weiner、N. Agarwal、P. Bhattacharya、C. Petersen、M. Chowdhury、S. Kanaujia 和 P. Chauhan。TPP：面向 CXL 的分层内存的透明页面放置。载于 ASPLOS，2023年。[34] H. A. Maruf、Y. Zhong、H. Wong、M. Chowdhury、A. Cidon 和 C. Waldspurger。Memtrade：面向公有云的解聚内存市场

云。载于 *SIGMETRICS*，2023年。[35] T. Newhall、S. Finney、K. Ganchev 和 M. Spiegel。Nswap：面向 Linux 集群的网络交换模块。载于 *Euro-Par*，2003年。[36] B. Nitzberg 和 V. Lo。分布式共享内存：问题与算法综述。*Computer*，24(8)：52–60，1991年。[37] F. Pearce。能源消耗大户：能否让全球大型数据中心更加高效? *Yale Environment*，2018年。[38] K. Rashmi、M. Chowdhury、J. Kosaian、I. Stoica 和 K. Ramchandran。EC-Cache：借助在线纠删码实现负载均衡、低延迟的集群缓存。载于 *OSDI*，2016年。[39] A. Raybuck、T. Stamler、W. Zhang、M. Erez 和 S. Peter。HeMem：面向大数据应用和真实非易失性内存的可扩展分层内存管理。载于 *SOSP*，2021年。[40] C. Reiss、A. Tumanov、G. R. Ganger、R. H. Katz 和 M. A. Kozuch。大规模云环境的异构性与动态性：Google 追踪分析。载于 *SoCC*，2012年。[41] Z. Ruan、S. J. Park、M. K. Aguilera、A. Belay 和 M. Schwarzkopf。Nu：借助逻辑进程实现微秒级资源可替代性。载于 *NSDI*，2023年。[42] Z. Ruan、M. Schwarzkopf、M. K. Aguilera 和 A. Belay。AIFM：高性能、应用程序集成的远端内存。载于 *OSDI*，2020年。[43] H. Servat、A. J. Peña、G. Llort、E. Mercadal、H.-C. Hoppe 和 J. Labarta。自动化混合内存系统中的应用数据放置。载于 *IEEE* 国际集群计算会议，2017年。[44] Y. Shan、Y. Huang、Y. Chen 和 Y. Zhang。LegoOS：面向硬件资源解聚的分散式分布式操作系统。载于 *OSDI*，2018年。[45] A. Tatar、R. K. Konoth、E. Athanasopoulos、C. Giuffrida、H. Bos 和 K. Razavi。Throwhammer：网络上的 Rowhammer 攻击及防御。载于*ATC*，2018年。[46] S.-Y. Tsai、M. Payer 和 Y. Zhang。Pythia：面向大众的远程预言机。载于 *USENIXSecurity*，2019年。

[47] Vladimir Davydov。《空闲内存跟踪》。https://lwn.net/Articles/ 639341/.[48] M. Vuppalapati、J. Miron、R. Agarwal、D. Truong、A. Motivala 和 T. Cruanes。基于解耦存储构建弹性查询引擎。载于 *NSDI*，2020年。[49] C. Wang、H. Ma、S. Liu、Y. Li、Z. Ruan、K. Nguyen、M. D. Bond、R. Ne- travali、M. Kim 和 G. H. Xu。Semeru：内存解耦托管运行时。载于 *OSDI*，2020年。 [50] K. Wu、Y. Huang 和 D. Li。Unimem：基于非易失性内存的异构主存中的运行时数据管理。载于超级计算大会，2017年。 [51] D. Y. Yoon、M. Chowdhury 和 B. Mozafari。基于远程直接内存访问的分布式锁管理：无饥饿的去中心化。载于 国际数据管理大会，2018年。[52] J. You、J. Wu、X. Jin 和 M. Chowdhury。传输计算还是传输数据？为什么不能两者兼顾？载于 *NSDI*，2021年。 [53] Z. Yu、Y. Zhang、V. Braverman、M. Chowdhury 和 X. Jin。NetLock：使用可编程交换机实现快速集中式锁管理。载于 数据通信特别兴趣组大会，2020年。[54] Y. Zhang、J. Gu、Y. Lee、M. Chowdhury 和 K. G. Shin。远程直接内存访问中的性能隔离异常。载于 *ACM SIGCOMM KBNets*，2017年。[55] Y. Zhang、G. Kumar、N. Dukkipati、X. Wu、P. Jha、M. Chowdhury 和 A. Vahdat。Aequitas：数据中心性能关键型远程过程调用的准入控制。载于 *ACM SIGCOMM*，2022年。[56] Y. Zhang、Y. Tan、B. Stephens 和 M. Chowdhury。Justitia：硬件内核旁路网络中的软件多租户。载于 *NSDI*，2022年。 [57] W. Zhao 和 J. Ning。Tioga Pass 项目修订版 0.30：服务器 Intel 主板 V4.0 规格。https://www.opencompute.org/documents/facebook-server-intel-motherboard-v40-spec。