分布式共享内存研究进展

摘 要 分布式共享内存(distributed shared memory)是并行处理发展中出现的一种重要技术。DSM提供给程序员一个逻辑上统一的地址空间，任何一台处理机都可以对这一地址空间直接进行读写操作，具有分布式内存结构可扩充性的优点，也具有共享内存结构通用性好、可移植性、编程容易的优点。DSM热点技术包含复制问题、存储一致性模型等。另外远程直接内存访问(RDMA)技术正在大数据领域被越来越广泛的应用，它支持在对方主机CPU不参与的情况下实现远程读写异地内存，并提供高带宽、高吞吐和低延迟的数据传输特性，所以RDMA的出现进一步激励了人们对DSM 系统的兴趣。本文介绍三种DSM 实现，首先介绍CoRM，它是一个RDMA 加速的共享内存系统，提供单侧RDMA访问，同时支持内存压缩并确保严格的一致性。其次介绍Gengar，一个RDMA 受限的分布式共享混合内存（DSHM）池，具有简单的编程API，用于在全局内存空间中查看远程NVM和DRAM。最后介绍Concordia，与前两者是基于RDMA技术相比，Concordia是一种由可编程交换机支持的具有快速网络内缓存一致性的DSM，核心是充分利用交换机和服务器的协作

关键词 DSM；RDMA；交换机；分布式共享内存

Research Progress of Distributed Shared Memory

Abstract

Distributed shared memory is an important technology emerging in the development of parallel processing. DSM provides programmers with a logically unified address space. Any processor can directly perform read and write operations on this address space. It has the advantages of scalability of distributed memory structure and good versatility of shared memory structure. , Portability, and easy programming advantages. DSM hotspot technologies include replication issues, storage consistency models, etc. In addition, remote direct memory access (RDMA) technology is becoming more and more widely used in the field of big data. It supports remote reading and writing of remote memory without the participation of the other host's CPU, and provides high bandwidth, high throughput and low latency. Data transmission characteristics, so the emergence of RDMA has further stimulated people's interest in DSM systems. This article introduces three DSM implementations. First, CoRM is introduced. It is an RDMA-accelerated shared memory system that provides single-sided RDMA access, while supporting memory compression and ensuring strict consistency. Next, we introduce Gengar, an RDMA-limited distributed shared hybrid memory (DSHM) pool with a simple programming API for viewing remote NVM and DRAM in the global memory space. Finally, Concordia is introduced. Compared with the first two based on RDMA technology, Concordia is a DSM with fast network cache consistency supported by programmable switches. The core is to make full use of the cooperation of switches and servers.

Keywords DSM;RDMA;switch;distributed shared memory

1 引言

具有远程直接内存访问(RDMA) 的快速低成本网络的广泛使用促使现代数据库管理系统采用RDMA 来提高查询性能。RDMA 已经支持复制、索引结构、分布式事务和分析工作负载的处理。RDMA 的出现进一步激发了对分布式共享内存(DSM) 系统的研究兴趣，DSM将互连节点的内存组合为共享的远程可访问内存空间。内存数据库、缓存服务和临时存储只是此范例的一些示例。然而，虽然DSM 系统提供全局内存视图和资源分解，但它们需要有效地处理诸如内存分配和内存碎片整理等管理任务。支持RDMA 的网络接口卡(RNIC) 使系统能够访问远程对等方的主内存，与传统的TCP/IP 网络相比，主机CPU 提供高达22倍的较短延迟和高达20 倍的吞吐量。但是，使用RDMA 会阻止内存优化策略，例如内存压缩。事实上，远程对象是通过在远程主机上指定它们的虚拟地址来访问的：如果远程主机重新定位一个对象，它的虚拟地址可能会改变，需要将此更新传播到其他节点。为了避免这个问题，一些RDMA 系统会选择不公开存储对象的虚拟地址，而是将对象的句柄分发给客户端。虽然这种间接性隐藏了更新重定位对象指针的过程，但由于指针追逐开销，它会显著阻碍性能。另外内存碎片也是一个严重问题，û有压缩的传统内存分配器可能会遭受灾难性的内存碎片。此外，虽然碎片将内存数据存储的内存使用率提高了69

因此第一篇文章提出了CoRM[1]，这是一种共享内存系统，它利用RDMA 进行快速远程访问并支持内存压缩，CoRM 的压缩是RDMA 安全的，同时为了便于压缩，CoRM 中的对象与在对象分配时随机生成的块本地对象ID 相关联。另外压缩算法是概率性的：两个内存块只有在它们û有冲突的情况下才能被压缩为一个，即两个块中的对象不具有相同的ID。CoRM 旨在为RDMA 加速的DSM 系统（例如FaRM）提供内存压缩，而不会影响严格的一致性和单侧RDMA 访问。对于合成工作负载，它可以将FaRM 的内存占用减少多达6 倍，对于实际应用程序，它可以减少多达2.9 倍的内存占用。

字节可寻址非易失性存储器(NVM) 技术，例如相变存储器(PCM) [7]和英特尔傲腾DC 持久性存储器模块(DCPMM) [8]提供高存储器密度、低成本和接近于零待机功耗。它们可以在单台机器中提供数十TB 的主内存。然而，由于与DRAM 相比，NVM 具有有限的写入耐久性和相对较低的读/写性能，因此它们经常与DRAM 一起使用以形成混合存储系统[7],[9]-[11]。以前对混合内存系统的大多数研究主要集中在单机环境中的性能改进[12],[13]和降低能耗[14]，关于在分布式共享混合内存(DSHM) 系统中有效利用DRAM 和NVM 的研究相对较少。最近，NVM 的出现激发了一些关于基于RDMA 的分布式持久内存(PM) 和文件系统的研究。然而，目前还不清楚如何充分利用DSHM 系统的高性能RDMA 硬件。同时在DSHM 系统中有效管理NVM 和RDMA 仍然存在一些挑战，首先，尽管DRAM 缓冲方案已被广泛用于加速单个服务器中的NVM 访问，但它们不再适用于DSHM 系统。由于RDMA 操作绕过远程服务器的操作系统(OS)，传统的OS 支持的内存监控机制很难应用于启用RDMA 的DSHM 环境。其次，大多数RDMA 操作本质上是异步的，这方面有改进性。RDMA 的异步通信模式为优化RDMA 网络的不同构建块提供了大量机会，例如，DaRPC 和FaSST 重新设计了RPC 以减少RDMA 通信的软件开销。

所以在第二篇文章中提出了Gengar，一个支持RDMA 的DSHM 池。Gengar 利用RDMA 读/写动词（即单边RDMA 模型）来实现低延迟远程内存访问，还提供了一组简单的API 来促进DSHM 池的RDMA 编程。关键技术设计有三点，首先，提出了一种有效的DRAM 缓冲方案来加速远程NVM 访问。其次，通过重新设计RDMA 写原语提出了一种新颖的RDMA 写模式。第三，Gengar提供简单的API来保证DSHM 系统中对象共享的一致性。此外，Gengar 利用租约分配机制来保证元数据的一致性，并利用轻量级写锁来保证数据/元数据的一致性。

然而，分布式缓存总是伴随着一致性问题，这是设计中的关键问题。一致性极其复杂并且需要很大的开销。具体而言，同步会导致过度通信以进行协调，例如在验证和序列化冲突请求时跟踪缓存副本的分布（请求启动同一缓存块），从而严重影响整体吞吐量。可编程交换机的出现改变了改变了这种现状，使得在DSM中利用可定制的交换机来减少缓存一致性的开销是有希望的。首先，由于交换机是服务器间通信的中心枢纽，因此将其重新配置以处理缓存一致性可以显著减少服务器之间的协调。其次，最新的交换机通常ÿ秒可以处理数十亿个数据包，使它们能够快速处理一致性请求。第三，交换机拥有一个片上内存，允许在交换机中存储缓存块元数据。此外，片上存储器可以支持原子读修改写操作[33]，从而简化同步冲突一致性请求的工作。然而，简单地将所有缓存逻辑塞入交换机是不切实际的。首先，缓存一致性协议对于可编程交换机来说过于复杂，由于其严格的编程模型和苛刻的时序要求，可编程交换机的表达能力有限。

其次，片上内存通常很小（例如10-20 MB），因此不太可能容纳整个缓存集的元数据。第三，故障处理可能很棘手。缓存一致性协议通常使用状态机来执行状态转换。因此，如果在交换机上部署，则常见故障（如数据包丢失）可能会导致交换机进入错误状态（例如，由于重新传输而重复锁定导致死锁）。

因此第三篇文章中提到了Concordia，它是一种具有网络内缓存一致性的高性能机架级DSM。Concordia的核心是FLOWCC（in-Flow Cache Coherence），这是一种由交换机和服务器共同支持的缓存一致性协议。在FLOWCC中，交换机作为数据平面，将冲突的一致性请求序列化并将它们多播到目的地，从而减少服务器之间的协调。另一方面，服务器充当执行状态转换并向交换机发送相应更新的控制平面。另外Concordia还设计了所有权迁移机制来管理缓存块的元数据。该机制根据一致性流量动态地在交换机和服务器之间移动缓存块的所有权，即只保留交换机中的热缓存块。为了解决数据包丢失问题，使服务器和交换机中的所有操作都是幂等的，从而保证只使用一次语义。

2 原理和优势

2.1 CoRM

在DSM 系统中，进程的内存可以由本地运行的应用程序任务分配，也可以由远程应用程序任务分配。因此，DSM 节点需要管理并发内存分配，就像在普通的多线程应用程序中一样，而且现在分配请求也可以来自网络。并发内存分配器(CMA) 有两个主要要求：(1) 规模,管理内存的线程数量；(2) 保持低内存碎片,最大限度地提高内存效率。这里将内存碎片定义为操作系统授予进程的内存量与进程有效使用的内存量之间的比率。*Scalability*为了提高可扩展性，大多数CMAs[15],[16]采用两级架构，在模型中，ÿ个线程由一个线程本地分配器提供服务，该分配器具有空闲内存堆：内存分配请求由这个堆提供，不需要全局同步。如果线程本地分配器内存不足，它会从进程范围分配器请求新的内存，进程范围的分配器可以直接从操作系统分配内存。为了避免频繁访问进程范围的分配器，这可能需要同步，线程本地分配器一次获取一个以上的空闲页面。一个对象被分配在可以容纳它的最小尺寸类中。因此，必须仔细选择大小类以限制内部碎片。基于块的方法引入了同步开销和内存效率之间的权衡：块大小越大，访问进程范围分配器的次数越少，因此可扩展性越好。另一方面，如果分配线程û有充分利用分配的块，则较大的块会导致内存效率低下。*Memory Fragmentation*高内存碎片可能是由不规则的分配或特定大小类别的低使用率引起的。当分配器经历了大量分配，然后只有部分释放集时，就会发生分配尖峰。问题是不能保证这些释放会导致内存被释放：事实上，包含至少一个对象的块不能被释放回进程范围的分配器。在这种情况下，线程可能会留下许多很少使用且无法释放的块，从而导致内存碎片。*Memory Compaction*为了减少内存碎片，CMA 系统可以采用内存压缩策略。原则上，这些策略包括采用一组很少使用的块并将它们合并为一个，释放其他块。此过程需要保留压缩对象的可访问性，以便客户端仍然可以访问它们。一个常见的策略是使用一个间接表，将对象键映射到它们当前的内存位置[28]，允许系统通过更新表中的相应条目来在内存中自由移动对象。但是，间接表的使用会导致撤销对存储对象的直接RDMA 访问，从而导致读取吞吐量降低2 倍[13,14]。为了有效地启用共享内存抽象，DSM 系统必须最小化远程内存访问的开销。为了缩小本地和远程访问之间的性能差距，现代DSM 系统采用RDMA 来确保低延迟和高吞吐量[1,2,9,14]。RDMA 是一种机制，允许一台机器通过网络直接访问其他远程机器的内存。发送方的RNIC 直接从发送方的内存中读取数据并注入网络。在接收端，RNIC 接收数据并将其直接写入主机内存，绕过操作系统并最小化端到端延迟。启用RDMA 的主机通过可靠或不可靠的队列对(QP) 进行通信。在这项工作中只考虑可靠的QP，因为它是唯一一种支持单边RDMA 读操作的QP。原则上，所有DSM 操作（即内存分配和释放、读写）都可以作为RDMA 实现方面的操作。然而，它们中的大多数将需要多次往返，从而阻碍了RDMA 提供的性能提升：例如，仅使用RDMA 单侧操作分配内存将需要读取、修改和写回目标节点的分配状态（û有考虑到DSM 的多个节点可以同时针对同一个内存）。同时采用FaRM [27] 的方法，它使用RDMA 加速远程读取，同时使用远程过程调用(RPC) 实现其他操作。

2.2 Gengar

2.2.1 Hybrid Memories and RDMA

Intel傲腾DCPMM [8]的出现终于使NVM 实现了商用。它可以与DRAM（内存总线）一起配备，以构建大容量和节能的混合内存系统。Optane DCPMM 可用于两种模式：应用程序直接模式和内存模式[17]。前一种模式为操作系统和应用程序提供了NVM的持久性功能。操作系统将DRAM和Optane DCPMM 分别作为主内存和持久存储提供给应用程序。Optane DCPMM 用于直接访问(DAX) 命名空间，该命名空间提供由应用程

序通过特殊API 直接访问的字节可寻址持久存储。在内存模式下，DCPMM 的持久性功能被禁用。Optane DCPMM 用作更大容量、更低性能的主内存，DRAM 用作操作系统不可见的缓存，以缩小DRAM 和NVM 之间的性能差距。Optane DCPMM 在显着提高HPC 和数据中心应用程序的性能和能源效率方面具有巨大潜力。但是，其性能远低于DRAM，DCPMM的单条加载指令读取延迟比DRAM高约2.3-4倍，随机写入时DCPMM的写入带宽可低至DRAM带宽的3.4

RDMA 是一种高性能通信机制，它使用户空间应用程序能够直接访问远程服务器中的内存，绕过操作系统内核并消除缓冲区之间的内存复制（零复制）[18]。通常有两种RDMA 操作：使用读/写动词的单侧RDMA 和使用发送/接收动词的双侧RDMA。两侧RDMA 类似于消息传递编程模型。当客户端执行RDMA发送操作时，服务器应该通知客户端存储数据的内存地址。通过避免远程服务器的CPU [19]的参与，单侧RDMA 提供比两侧RDMA 低得多的延迟和更高的带宽。目前的RDMA通信主要包括三种模式：可靠连接（RC）、不可靠连接（UC）和不可靠数据报（UD）。单边RDMA 操作依赖于可靠连接(RC)。RDMA 通信基于客户端和服务器节点之间的一对队列，称为队列对(QP)。QP 包括一组工作队列：发送队列(SQ)、接收队列(RQ) 和完成队列(CQ)。它们负责一起调度RDMA 操作。客户端通过在SQ 上发布工作请求(WR) 来执行RDMA 操作。WR 完成后，将创建工作完成(WC) 事件并将其发布到CQ。当客户端收到WC时，对应的RDMA操作就完成了。

2.2.2 Optimization Opportunities for DSHM

目前，远程NVM 访问的高延迟是在数据中心环境中使用DSHM 的主要问题。例如，256 字节对象的远程访问延迟约为7.3 us [19]，比本地NVM 读取延迟（ 0.7 us）高一个数量级，也远高于本地NVM 写入，因此，减少远程NVM 的读/写延迟仍有很大的空间。RDMA 操作通常涉及多个组件，即本地应用程序、本地内存区域(MR)、本地网络接口卡(NIC)、远程NIC 和远程内存（如NVM）。不同的RDMA 通信模型都允许这些解耦组件单独执行数据传输，从而为优化不同组件中RDMA 通信的性能提供了巨大的机会。例如，MR 重用机制[20]-[22]已被广泛用于降低MR 注册的成本。FaSST [14] 利用协作多任务、NIC 门铃批处理和事件合并机制来优化RPC，从而减少RDMA 网络延迟。DaRPC [15] 通过利用服务器端的完成队列共享/集群来提高RDMA 操作和RPC 处理的并行性。我们在支持RDMA 的DSHM 中观察到以下性能优化机会。

2.3 Concordia

DSMs中使用两种主要协议，首先是基于目录的协议，它保留缓存副本的服务器跟踪。ÿ个数据块都有一个主节点，用于保存缓存副本的状态和位置。主节点向所有缓存副本更新并通知数据块的状态，并序列化冲突的缓存一致性请求。限制是主节点导致额外的往返，并且热数据的主节点处于重载状态。其次是基于侦听的协议，它不需要跟踪缓存副本。相反，它们向所有服务器广播缓存一致性请求。限制是广播很容易使网络不堪重负，并且浪费不包含请求的缓存块的服务器的CPU 周期。此外，由于有序且可靠的广播系统等同于共识[23]，因此在侦听协议中协调冲突的缓存一致性请求是棘手的。

Barefoot Tofino[2]等新兴可编程交换机提供可编程容量。这种交换机遵循可重构匹配表（RMT）体系结构[24]，通常具有多个入口和出口管道。ÿ个管道包含多个阶段，数据包按顺序由这些阶段处理。RMT管道的两个特性，它们可以简化交换机中有状态协议的设计：原子性，由于流水线架构，任何时候在一个阶段只处理一个数据包。换句话说，同一阶段的多个寄存器数组的操作是原子的；有序性，假设有两个数据包A，B，它们分别在SA和SB阶段被处理。如果SA¡SB在时间t（例如，A在阶段1，B在阶段2），在之后的管道中SA¡SB在任何时候都成立。

Concordia有三个关键设计，(1)分离数据平面和控制平面。在concordia中，交换机只做其擅长的事情，即作为数据平面的·由包；它多播缓存一致性请求，并通过网络内锁序列化冲突的请求（锁定可被视为控制冲突请求的·由·径）。相反，服务器作为控制平面，执行状态转换并向交换机发送相应的更新。这种分离简化了交换机的数据平面设计，以克服其有限的表达能力。(2)统一基于交换机和服务器的一致性处理。由于交换机的片上内存容量有限，一致性使交换机只能管理热数据的一致性，并借助服务器处理冷数据。根据缓存块诱导的一致性流量，Concordia在服务器和交换机之间动态迁移其所有权。(3)最小化对交换机状态的修改次数。对于修改交换机状态的ÿ个操作，即存储在交换机寄存器阵列中的值，我们必须处理相应的数据包丢失情况，代价是消耗交换机宝贵的硬件资源并使系统设计复杂化。因此，必须减少开关状态修改的次数。具体地说，在ÿ个同步事件中，即执行缓存一致性请求的过程中，交换机只修改其状态两次：1）请求锁；2）释放锁定并安装目标缓存块的

新元数据。我们使这两个修改幂等，以处理数据包的丢失。

3 研究进展

3.1 CoRM

CoRM 是一种内存管理系统，它利用RDMA来改善内存访问的延迟和吞吐量。使用CoRM可以在RDMA 加速的DSM 系统中启用内存压缩，而无需引入间接以充分利用单边RDMA 操作。像FaRM这样的DSM 系统牺牲内存压缩以使用RDMA 加快远程通信。CoRM 旨在为RDMA 加速的DSM 系统（例如FaRM）提供内存压缩，而不会影响严格的一致性，但需要在对象标头中存储额外的元数据。由于CoRM 的API 模仿了FaRM 的API，并且只添加了一个用于释放未使用的虚拟地址的维护调用，所以此种压缩策略可以毫不费力地集成到FaRM 中。

CoRM 支持不损害客户端对象指针的压缩, 系统利用RDMA 感知内存重新映射在物理内存块之间静默地移动对象，同时保留它们的虚拟地址和RDMA 访问密钥。

1.*Allocation algorithm.*CoRM 使用前面描述的并发内存分配器，类似于大多数内存系统[25],[26]。分配器支持不同的8 字节对齐大小的列表，选择这些列表是为了减少由于向上舍入到最接近的大小类别而导致的平均内部碎片。CoRM 中的进程范围块分配器可以分配大小为4 KiB 倍数的块（即，正常大小的页面）。但是，CoRM 可以很容易地扩展到处理大页面以减少页面数量。

2.*Compaction algorithm.*CoRM 可以压缩属于同一台机器的相同大小类的块。压缩算法的高级思想是找到两个相同类别的低利用率块，并将对象从一个块（源）复制到另一个（目标），一旦所有对象都被复制后，源块可以被释放，它的虚拟地址被重新映射到目标块的物理地址。此时，我们有两个虚拟地址指向同一个物理页。映射也在RNIC 上更新，以保留对源块对象的RDMA 访问。Mesh [26] 提出了一种类似的内存压缩方法。然而，只有当块中û有对象占据相同的偏移量时，才可能在网格中进行压缩。这就带来了一个限制，即Mesh 是C/C++ 程序中malloc 的插件替代品，应用程序可以通过使用带有加载/存储指令的虚拟地址来自由读写内存。在这种情况下，页面虚拟地址可以透明地重新映射（即，转换由MMU 执行）但对象偏移量不能改变。因此，只有当它们的对象在偏移量上û有冲突时，Mesh 才能压缩块。然而，在DSM 系统中，用户总是使用显式读/写函数来访问内存：这些是解析远程指针和启用并发控制所必需的。CoRM 利用了DSM 编程模型并放宽了对压缩对象保持不变的要求压缩后的偏移量。为了实现这一点，CoRM 为块中的ÿ个对象分配标识符（ID）。ID仅在单个区块内唯一，采用均匀分布随机生成。对象在内存中通过区块地址和对象ID唯一标识，对象ID存储在对象的头部。这种设计选择允许CoRM 仅在其中的对象不具有相同ID 时压缩两个块。与Mesh 的压缩条件基于偏移不同，在CoRM中，对象ID 是随机的，并且可以调整ID 大小（默认为16 位），提高了压缩概率。事实上，虽然块也可能在对象ID 中存在冲突，但它们的可能性低于偏移冲突。

3.*Compaction policy.*CoRM 计算ÿ个大小类别的碎片率。如果碎片率超过碎片阈值，则CoRM 会针对ij个大小类触发压缩。可以根据压缩概率为ÿ个大小类调整碎片阈值。当由于内存不足导致分配失败时，CoRM 还可以启动压缩。

4.*Compaction mechanism.*在CoRM 中，ÿ个线程都有自己的私有内存分配器。因此，可以压缩的块可能属于不同的线程，从而阻止了高效的无锁内存压缩。为了解决这个问题，CoRM 选择一个工作线程作为压缩领导者，它在两个阶段执行压缩：块收集和块压缩。在块收集期间，领导者向所有其他线程广播收集请求，要求一定大小级别的足够低占用的块。在第二阶段，所有线程都回复了收集请求后，领导者可以启动压缩算法。我们的两阶段设计消除了对昂贵的并发数据结构的需要，因为CoRM 持有一个不变式，即任何块最多由一个拥有线。CoRM 首先尝试压缩最少使用的块，因为它们具有较少的元素并引起较少的偏移冲突。在压缩两个块的过程中，除了处理对象外，CoRM 还会合并受影响块的元数据。元数据是一个哈希表，它保留了对象ID 和偏移量之间的映射，仅用于快速指针校正。

由于压缩后的对象可以在压缩后在内存中移动，因此用户持有的虚拟指针可能不会直接指向所需的对象。为了避免在其块中搜索对象，ÿ个用户的对象指针都包含对象在块中预期的偏移提示。CoRM 总是乐观地访问在暗示的偏移量处的对象（使用加载指令），然后检查其ID。如果访问对象的ID 与使用的指针中的ID 不匹配，则CoRM 执行搜索以找到请求的对象。一旦找到对象，对象指针内的提示就会更新为新的偏移量，使指针成为直接的。

3.2 Gengar

Gengar 提供基于客户端/服务器模型的远程内存访问。客户端节点运行应用程序并访问服务器节点中的远程内存，其中Gengar 服务器管理共享的NVM 和DRAM 缓冲区，û有集中的元数

据服务器。ÿ个客户端在本地维护一个远程服务器列表，ÿ个服务器将其可用的DRAM/NVM 容量随RDMA 操作的ACK 报告给客户端。Gengar 将所有服务器中的NVM 资源作为共享内存池进行管理，并使用少量（本地或远程）DRAM 作为分布式缓冲区以加速NVM 访问。首先，Gengar 客户端从共享内存池中为应用程序分配NVM。当一个对象被识别为热（经常访问）数据时，Gengar 客户端从内存池请求DRAM 来缓存热对象。我们注意到DRAM 缓存可以驻留在任何服务器节点中，因此当热对象数量增加时Gengar 可以很好地扩展。当一个对象被释放时，Gengar 服务器从DRAM 缓冲区和共享NVM 池中回收其内存。热数据缓存由Gengar 单独执行，不涉及客户端应用程序。程序员甚至不必知道远程内存池的异构性。尽管传统文件系统提供了丰富的接口来轻松使用远程PM [34]，但相对复杂的I/O 堆栈会增加软件层的PM 访问延迟。Gengar 选择基于对象的DSM 设计来直接访问远程共享PM，同时提供了非常简单的API 来通过对象使用内存池。共享NVM 分配和释放是在请求/响应通信模型中实现的。当客户需要更多的远程内存资源，Gengar 客户端向Gengar 服务器发送内存分配请求。分配成功后，服务器节点将远程MR的全局内存地址（64位）返回给Gengar客户端，并将注册的MR的虚拟到物理地址映射存储在哈希表中（称为地址映射表）(AMT)，客户端和服务器都在本地维护一个AMT，用于地址转换和内存管理。客户端中的AMT 只记录客户端和多个服务器之间的地址映射。相比之下，服务器中的AMT 只记录多个客户端与服务器之间的地址映射。一旦NVM 数据缓存在DRAM 缓冲区中，我们还会在地址映射表中维护NVM 到DRAM 的映射。当客户端中的应用程序的数据对象被释放时，Gengar 客户端向Gengar 服务器发送内存释放请求。如果对象也缓存在DRAM 缓冲区中，则Gengar 服务器释放DRAM 缓冲区和共享NVM 池中的内存区域，并且相应的虚拟到物理地址映射也从地址映射表中删除。由于一个对象可以被多个客户端应用程序共享，而Gengar 不知道应用程序级别的语义，因此Gengar 为程序员提供了一种简单有效的机制来更新具有元数据一致性保证的共享对象。为了保证存储在DSHM 池中的所有数据都是持久的，Gengar 中的分布式DRAM 缓冲区被实现为一个直写缓存。如果数据已存在于DRAM 缓冲区中，则将其同时写入DRAM 缓冲区和后备NVM。否则，数据将直接写入NVM，绕过DRAM 缓冲区。这样，Gengar 只将频繁读取的对象缓存在DRAM 缓冲区中，以在保证数据持久性的同时加快RDMA 读取操作。

图2展示了Gengar中RDMA的写模式。由于远程服务器的CPU 和操作系统无法感知单方面的RDMA 操作，因此很难监控服务器端的数据访问。Gengar 利用RDMA 读取操作的语义来识别客户端的热点对象，为地址映射表中的ÿ个对象添加读取计数器。对于ÿ个RDMA读操作，首先查询地址映射表，获取远程内存地址，同时Gengar 客户端顺便更新了对象读取计数器。在固定的时间内，Gengar 客户端能够统计ÿ个对象的RDMA 读取频率。如果读取计数超过给定阈值，则对象被确定为热对象。ÿ个Gengar 客户端单独确定热点对象，然后将这些对象的读取计数发送到服务器，用于全局选择热点对象。由于服务器节点上的可用DRAM 资源可能会随时间变化，因此DRAM 缓冲区的大小也会动态变化。为了最好地利用DRAM 缓冲区的稀缺资源，需要根据DRAM 利用率和DRAM 缓存的优势动态调整热数据的阈值。当DRAM 资源充足时，Gengar 会降低热点对象的阈值，以在DRAM 缓冲区中缓存更多的NVM 数据，从而加快远程内存访问。当DRAM 缓冲区已满时，Gengar 服务器将DRAM 缓冲区中的一些冷对象逐出以缓存较热的对象，以提高DRAM 缓冲区的效率。

最后，为了保证所有客户端元数据的一致性，Gengar利用服务器端租用分配策略[36] 来减轻元数据同步的开销。Gengar 使用时间戳为ÿ个对象设置租约到期时间。对于释放内存和缓存对象的操作，Gengar 客户端需要等待当前租约到期。一旦客户端获得访问对象的租约，它就可以执行RDMA 操作，而无需在多个客户端之间频繁同步元数据。其次为了实现并发控制，Gengar 提供了一个应用级API（设置锁）来支持对共享对象的数据和元数据的并发控制。对于分配/释放/-缓存/驱逐和读/写操作，Gengar 使用一个锁和3 个RTT 来完成这些RPC。

3.3 Concordia

Concordia的核心是一个写失效协议FLOWCC，它在交换机和服务器之间划分一致性责任。交换机序列化冲突请求，并通过锁定前向检查（LCF）管道将它们多播到正确的主/缓存代理，服务器执行状态转换并更新交换机数据平面。应用程序线程（即请求程序）通过向本地缓存发出写/读操作来访问全局内存，并在缓存未命中和缓存逐出的情况下生成缓存一致性请求。在LCF管道中，为ÿ个缓存块存储读写器锁和全局元数据。通过利用这种状态，LCF管道将冲突请求和多播序列化到目标。LCF 管道通过三个步骤处理请求：（i）锁定请求的缓存块以进行并发控制。（ii）检查请求的有效性。（iii）

将请求转发至最终目的地。当一个请求进入LCF管道时，交换机首先将请求的标签字段哈希成索引号，然后使用索引号在TAG 数组中搜索该标签（即检查TAG[hash(tag)] 是否等于标签）。失败时，交换机将请求转发给其主代理，并将field设置为false。找到标签后，如果请求为读取未命中，则交换机尝试获取读锁；否则，将获取写锁。注意使用写锁保护EVICT-SHARED，因为对同一缓存块的两个并发EVICT-SHARED 请求可能会导致不同的状态转换。同时，将相应的全局元数据（即G-STATUS和COPYSET 数组中的值）填充到请求中。当无法获取锁时，交换机向请求者返回失败的ACK。交换机获取锁成功后，将请求传递给检查表，过滤掉并发事件引起的无效请求。让我们考虑一个无效请求的例子：在请求者对缓存块A 发出WRITE-SHARED 请求W 之后，同一节点中的缓存代理立即使A 无效；可能是因为W 被OS 调度程序或网络延迟，当W 进入LCF 管道时，A 的全局状态不再是Shared 或请求者不再持有A。因此，为了确保仅将有效请求转发到目标节点，交换机会检查请求中的全局状态和副本集。如果检查失败，交换机重新提交释放获取的锁的请求，然后向请求者发送一个失败的ACK。与基于服务器的机制相比，这种网络内检查减少了服务器的负载（检查请求的有效性和发送失败的ACK）。如果请求成功通过校验表，则转发表（见表3）根据（类型，全局状态）对将其发送到其最终目的地。具体有以下三种情况：(i) 如果请求的缓存块不存在副本，即全局状态为Unshared，则交换机将请求·由到其主代理以获取全局内存中的数据。(ii) 如果缓存块副本需要失效，交换机将请求多播到副本集中的缓存代理。(iii) 对于驱逐请求，交换机直接将其返回给请求者，因为已经获得了相应的锁。图3展示了Concordia的模型。

4 总结

本文首先介绍了CoRM，它是一种共享内存系统的原型，它采用RDMA 来加速远程读取，同时支持内存压缩以提供高内存效率，即使存在并发压缩，CoRM 的内存访问也是高度一致的。CoRM 在正常操作下显示出与FaRM 等其他RDMA 加速DSM 相同的读取吞吐量，并且它可以快速恢复由于内存压缩而创建的间接指针。在内存碎片化的情况下，CoRM 在压缩内存方面至少与Mesh 一样好，同时节省多达2.8 倍的内存。在大型对象发生分配/解除分配峰值的情况下进行网格划分。CoRM 的新型压缩算法基于对象ID 而不是偏移量，不使用间接表，完全依赖于操作系统虚拟地址转换。总而言之，CoRM 通过避免为了性能而牺牲内存效率的需要，填补了RDMA 加速共享内存系统中的空白。其次介绍了Gengar，一个支持RDMA 的分布式共享混合内存池。Gengar 允许应用程序通过客户端/服务器模型访问大型全局内存空间中的远程DRAM/NVM。在客户端利用RDMA 原语的语义来识别服务器节点中经常访问的（热）数据，并将其缓存在一个小的DRAM 缓冲区中。另外还重新设计了RDMA 通信协议以减少RDMA 写入延迟。最后介绍了Concondia，它是一种具有网络内缓存一致性的机架级DSM，通过在交换机和服务器之间划分一致性责任，以减少交换机功能和资源限制内的一致性开销。

参 考 文 献

[1] Taranov, Konstantin, Salvatore Di Girolamo, and Torsten Hoefler. ”CoRM: Compactable Remote Memory over RDMA.” *Proceedings of the 2021 International Conference on Management of Data*. 2021.

[2] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2016. FaSST: Fast, Scalableand Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs.InProceedings of the 12th USENIX Conference on Operating Systems Design andImplementation(Savannah, GA, USA)(OSDI 16). USENIX Association, USA, 185 – 201.

[3] oungjin Kwon, Hangchen Yu, Simon Peter, Christopher J. Rossbach, and EmmettWitchel. 2016. Coordinated and Efficient Huge Page Management with Ingens.In12th USENIX Symposium on Operating Systems Design and Implementation(OSDI 16). USENIX Association, Savannah, GA, 705 – 721

[4] Chiranjeeb Buragohain, Knut Magne Risvik, Paul Brett, Miguel Castro, WonheeCho, Joshua Cowhig, Nikolas Gloy, Karthik Kalyanaraman, Richendra Khanna,John Pao, Matthew Renzelmann, Alex Shamis, Timothy Tan, and Shuheng Zheng.2020. A1: A Distributed In-Memory Graph Database. InProceedings of the 2020ACM SIGMOD International Conference on Management of Data(Portland, OR,USA)(SIGMOD 20). Association for Computing Machinery, New York, NY, USA,329 – 344.

[5] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G.Shin. 2017. Efficient Memory Disaggregation with INFINISWAP. InProceedingsof the 14th USENIX Conference on Networked Systems Design and Implementation(Boston, MA, USA)(NSDI 17). USENIX Association, USA, 649 – 667.

[6] Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, Ana Klimovic, Adrian Schuep-bach, and Bernard Metzler. 2019. Unification of Temporary Storage in the NodeK-ernel Architecture. In2019 USENIX Annual Technical Conference (USENIX ATC 19).USENIX Association, Renton, WA, 767 – 782

[7] Moinuddin K Qureshi, Vijayalakshmi Srinivasan, and Jude A Rivers. Scalable High Performance Main Memory System Using Phase-change Memory Technology. *ACM

SIGARCH Computer Architecture News*, 37(3):24 – 33, 2009.

[8] Intel Optane DIMMs. https://www.tomshardware.com/news/intel- optane - dimm- pricing- performance, 39007.html.

[9] Gaurav Dhiman, Raid Ayoub, and Tajana Rosing. PDRAM: A Hybrid PRAM and DRAM Main Memory System. In *Proceedings of the 46th ACM/IEEE Design Automation Conference (DAC’09)*, pages 664 – 669, 2009.

[10] Haikun Liu, Yujie Chen, Xiaofei Liao, Hai Jin, Bingsheng He, Long Zheng, and Rentong Guo. Hardware/Software Cooperative Caching for Hybrid DRAM/NVM Memory Architectures. In *Proceedings of the 31st ACM International Conference on Supercomputing (ICS’17)*, pages 26:1 – 26:10, 2017.

[11] Subramanya R. Dulloor, Amitabha Roy, Zheguang Zhao, Narayanan Sundaram, Nadathur Satish, Rajesh Sankaran, Jeff Jackson, and Karsten Schwan. Data Tiering in Heterogeneous Memory Systems. In *Pro- ceedings of the 9th ACM European Conference on Computer Systems (EuroSys’16)*, pages 15:1 – 15:16, 2016.

[12] Sudarsun Kannan, Ada Gavrilovska, and Karsten Schwan. pVM: Persistent Virtual Memory for Efficient Capacity Scaling and Object Storage. In *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys’16)*, pages 13:1 – 13:16, 2016.

[13] Iyswarya Narayanan, Aishwarya Ganesan, Anirudh Badam, Sriram Govindan, Bikash Sharma, and Anand Sivasubramaniam. Getting More Performance with Polymorphism from Emerging Memory Technologies. In *Proceedings of the 12th ACM International Conference on Systems and Storage (SYSTOR’19)*, pages 8 – 20, 2019.

[14] Sudarsun Kannan, Moinuddin Qureshi, Ada Gavrilovska, and Karsten Schwan. Energy Aware Persistence: Reducing Energy Overheads of Memory-based Persistence in NVMs. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation (PACT’16)*, pages 165 – 177, 2016.

[15] EmeryD.Berger,KathrynS.McKinley,RobertD.Blumofe,andPaulR.Wilson. 2000. Hoard: A Scalable Memory Allocator for Multithreaded Applications. In Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (Cambridge, Massachusetts, USA) (ASPLOS IX). Association for Computing Machinery, New York, NY, USA, 117 – 128. https://doi.org/10.1145/378993.379232

[16] EmeryD.Berger,KathrynS.McKinley,RobertD.Blumofe,andPaulR.Wilson. 2000. Hoard: A Scalable Memory Allocator for Multithreaded Applications. In Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (Cambridge, Massachusetts, USA) (ASPLOS IX). Association for Computing Machinery, New York, NY, USA, 117 – 128. https://doi.org/10.1145/378993.379232

[17] Haikun Liu, Di Chen, Hai Jin, Xiaofei Liao, Bingsheng He, Kan Hu,and Yu Zhang. A Survey of Non-Volatile Main Memory Technologies: State-of-the-Arts, Practices, and Future Directions. *Journal of Computer Science and Technology*, 36(1):4 – 32, 2021.

[18] Anuj Kalia, Michael Kaminsky, and David G Andersen. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs. In *Proceedings of the 13rd USENIX Symposium on Networked Systems Design and Implementation (OSDI’16)*, pages 185 – 201, 2016.

[19] Youyou Lu, Jiwu Shu, Youmin Chen, and Tao Li. Octopus: An RDMA- enabled Distributed Persistent Memory File System. In *Proceedings of the 2017 USENIX Annual Technical Conference (ATC’17)*, pages 773 – 785, 2017.

[20] Aleksandar Dragojevi, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. FaRM: Fast Remote Memory. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI’14)*, pages 401 – 414, 2014.

[21] Jiuxing Liu, Jiesheng Wu, Sushmitha P. Kini, Pete Wyckoff, and Dha- baleswar K. Panda. High Performance RDMA-based MPI Implementa- tion over InfiniBand. In *Proceedings of the 17th Annual International Conference on Supercomputing (ICS’03)*, pages 167 – 198, 2003.

[22] Li Ou, Xubin He, and Jizhong Han. Mrrc: A Efficient Cache for Fast Memory Registration in RDMA. In *Proceedings of the NASA/IEEE Conference on Mass Storage Systems and Technologies (MSST’06)*, 2006.

[23] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. Forwarding Metamorphosis: Fast Programmable Match-action Processing in Hardware for SDN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM ’13, pages 99 – 110, New York, NY, USA, 2013. ACM.

[24] BojieLi,ZhenyuanRuan,WencongXiao,YuanweiLu,YongqiangXiong,Andr Putnam, Enhong Chen, and Lintao Zhang. 2017. KV-Direct: High-Performance In-Memory Key-Value Store with Programmable NIC. In Proceedings of the 26th Symposium on Operating Systems Principles (Shanghai, China) (SOSP 17). As- sociation for Computing Machinery, New York, NY, USA, 137 – 152. https: //doi.org/10.1145/3132747.3132756

[25] Bobby Powers, David Tench, Emery D. Berger, and Andrew McGregor. 2019. Mesh: Compacting Memory Management for C/C++ Applications. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (Phoenix, AZ, USA) (PLDI 2019). ACM, New York, NY, USA, 333 – 346. https://doi.org/10.1145/3314221.3314582