高技术通讯 2023 年 第33 卷 第10 期:1009-1019

doi:10.3772/j.issn.1002-0470.2023.10.001

软硬件协同的远端内存系统研究①

李海锋②*** 刘 珂* 陈明宇***

(*中国科学院计算技术研究所 北京 100190)

(**中国科学院大学 北京 100049)

摘 要 针对当前远端内存系统中页面预取与页面替换因操作系统与应用程序之间语义鸿沟导致的局限性问题,本文提出一个软硬件协同的远端内存系统。通过在内存控制器中增加热点页面提取表,将实时访存的热点页面信息通过内存中的缓冲区传送给操作系统。同时,通过对访存信息的学习,构建了高精度的异步预取框架与替换框架,降低应用关键数据路径的开销,提升远端内存系统的性能。本文利用内存跟踪工具构建了一个原型仿真系统。实验证明,在拥有全局实时访存信息后,预取框架可以实现超过90%的准确率与覆盖率,与谷歌提出的远端内存系统 Fastswap 相比,性能提升59%。相比于内核默认替换框架,替换框架使应用性能提升30%。

关键词 远端内存系统;软硬件协同;预取框架;替换框架

0 引 言

虚拟内存子系统(virtual memory system,VMS)一直是操作系统(operate system,OS)的重要组成部分,它作为中间层使用户可以创建并访问大量的虚拟内存空间。如今,VMS 在基于微秒级网络的远端内存系统中发挥着至关重要的作用。例如,云服务供应商利用 VMS 将冷页面,即不常用的页面,透明地交换到廉价的慢速存储设备当中,以节省内存成本[1-2];基于内核的远端内存系统与 VMS 紧密结合在一起,从而实现远端内存的访问[3-5]。此外,随着新兴内存(如持久内存[6])与互连内存(如新的计算互连标准[7-8])的出现,基于多层内存的系统利用 VMS 进行数据迁移或交换。

这些系统通常依赖于3 种核心 VMS 机制:(1)请求分页,即系统拦截内存访问,然后从慢速设备(如远端内存)读取页面,将其缓存在本地内存中;(2)页面交换,即系统将本地页面替换到慢速设备为其他进程腾出空闲内存;(3)页面预取,即系统预测未来页面使用情况,并在进程使用页面之前从慢速设备中读取页面到本地内存。

然而,现有的 VMS 数据路径存在高昂的开销,这是因为其使用了粗粒度的锁,且在关键路径上存在同步的函数调用。尽管一些工作在努力优化数据路径,但它们都没有办法解决这些根本问题[9]。因此,先前的工作专注于通过优化页面预取和收回算法,来降低应用在关键路径上访问远端的次数,从而尽可能减少关键数据路径的开销[5,9]。然而,这种方式仍然存在不足,其优势被第2 个限制所抵消,即页面预取和回收无法精确地确定要预取与回收的页面。这是因为它们只能用有限的访问信息来训练算法。为了让操作系统获取更加完整的访问信息,一种可能的解决方法是使用守护进程定期扫描页表中的访问位,以近似显示用户访问历史[1]。但是,这种方法不仅会造成由于地址转换后援缓冲器击落和

① 国家重点研发计划(2022YFB4500401),国家自然科学基金(62072439),北京市自然科学基金(212028)和山东省自然科学基金联合基金(ZR2019LZH004)资助项目。

② 男,1993 年生,博士生;研究方向:计算机系统结构;联系人,E-mail:lihaifeng@ict.ac.cn。

(收稿日期:2022-10-27)

固定扫描线程而带来的开销,而且产生的访问信息不是实时的,是粗粒度且高延迟的。

这种限制源于操作系统和硬件之间存在语义鸿沟:处理器仅能通过页表向操作系统传递粗粒度和陈旧的访问信息,因此操作系统对其运行的应用程序的内存访问信息了解有限,必须使用高开销的软件方法来近似获取应用程序访问信息。本文认为,源自内存层次结构中较高层的丰富访问历史记录可以使较低层构更好地分析内存访问模式,从而建立更好的驱逐或预取模型。例如,如果操作系统知道关键的最后一级缓存缺失信息,则 OS 可以构建精确的替换和预取模型。

因此,本文提出通过内存控制器(memory controller, MC)捕获应用实时访存信息,并将它们高效且实时地提供给操作系统。这样,VMS 获得了足够的实时内存访问信息,就可以改进其页面替换和预取算法。具体来说,本文在内存控制器中部署了热点页面提取表。热点页面提取表分析缓存块粒度的访存请求并生成页粒度的访存请求。反向页表通过软件的方式进行构建,负责将物理页面翻译为虚页面和进程号。

本文构建了 2 个样例来展示软硬件协同框架带来的好处。首先,改进了谷歌发布的远端内存系统[3],将其默认的顺序预取策略替换为软硬件协同框架支持的基于流的预取算法。其次,本文改进了默认的内核替换算法,将其基于访问位的替换算法改为软硬件协同框架支持的替换算法。除了本文提供的使用样例,其他场景也可以使用软硬件协同框架所提供的访存信息进行优化。例如,用户可以使用这些信息指导页面的迁移等。

由于 MC 无法直接修改,本文在 x86 服务器上构建了一个原型仿真系统,用于验证和评估。本文利用了基于硬件的内存跟踪工具 HMTT (hybrid memory trace tool)[10-11],同时在用户空间中构建处理访存信息的软件。本文认为新增的硬件设计是轻量级和通用的,因此可以轻松集成到中央处理器(central processing unit,CPU)芯片或各种 CPU 平台的 MC。

本文使用许多大规模内存应用程序来评估软硬件协同框架和这 2 个用例。总体而言,软硬件协同框架有助于实质性地改进基于 VMS 的系统。例如,当应用程序的一半内存被分配到远端时,软硬件协同框架驱动的预取器将基于谷歌的远端内存系统的应用性能提高 59%,同时,预取器的精度和覆盖率超过 90%。此外,软硬件协同框架驱动的页面回收机制降低了关键路径访问远端内存的次数,与内核默认的回收机制相比,最多可以降低 30%。

本文组织如下。第 1 节介绍研究背景与相关工作;第 2 节介绍软硬件协同框架的设计细节;第 3 节介绍预取与替换框架实现;第 4 节介绍测试平台及系统实现;第 5 节介绍实验评测方法及结果;第 6 节对全文进行了总结。

1　背　景

本节首先介绍了现代服务器架构与软硬件协同框架的背景。然后讨论了可以从使用软硬件协同框架中受益的几个用例。最后介绍了几种其他类型的远端内存系统。

1.1　现代服务器架构

图 1 显示了典型的服务器架构,最后一级缓存(last level cache, LLC)、内存控制器(MC)和高速串行计算机扩展总线标准(peripheral component ihterconnect express, PCIe)控制器通过处理器内部总线互连。MC 通过标准双倍速率(double data rate, DDR)总线与动态随机存取存储器(dynamic random access memory,DRAM)芯片通信。PCIe 控制器通过 PCIe 总线与 I/O 设备通信。当 LLC 发生缺失时,它将通过处理器总线向 MC 发送请求。随后,MC 发出 DDR 请求。显然,MC 知道所有 CPU 的全部内存访问。由于直接向DRAM写入访存信息消耗额外的

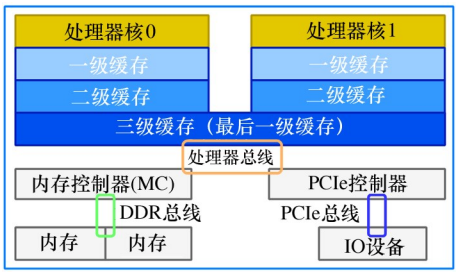

图 1　典型的服务器架构图

内存带宽,并且在操作系统读取访存信息时需要将数据从 DRAM 读到 LLC。所以,本文设计了软硬件协同框架,在 MC 中增加硬件逻辑,将缓存块粒度的访存请求转化为页粒度后,再通过内存传递给操作系统,降低传输时的带宽消耗。

1.2 基于内核的远端内存系统

这类远端内存系统依靠虚拟内存系统来透明地访问远程内存。因此,应用程序无需修改自身代码,直接使用远程内存。但这种透明不是没有代价的。缺页会引发高的软件延迟,甚至高过网络延迟,这使得软件的开销成为访问远程内存的瓶颈。

作为案例研究,本文评估了 Fastswap[3],这是谷歌在 2020 年基于 Linux 内核的 Frontswap 接口[12]提出的远端内存系统。研究发现 Fastswap 使用单核从远程读取页面大约需要 6 μs。图 2 显示了当增加线程数量时的延迟趋势,并详细介绍了 Fastswap 中的各种组件(暂时禁用预取)。本次测试将本地内存限制为 1 GB,而程序使用的总内存为 8 GB。线程固定在不同的内核上,每个内核从 4 kB 页面读取 8 字节整数并将所有值相加(即每页 512 次加法)。很明显,Fastswap 无法随着线程数量的增加而扩展。这是由于现有 VMS 中固有的粗粒度锁定和同步函数调用。

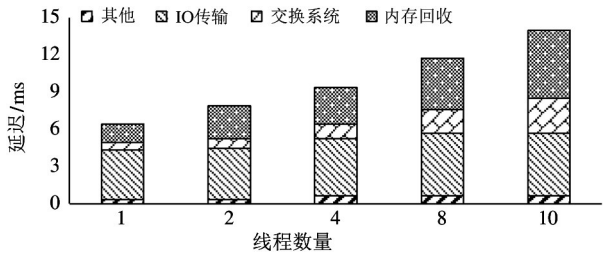

图 2 当线程数量变化时,关键路径上延迟的变化

克服上述 VMS 开销的一种方法是通过预取远端内存页面到本地内存,从而尽可能避免 VMS 数据路径中的 I/O 传输时间。Fastswap 使用默认的内核预取策略来读取出错页面[3]之后的后续 7 个页面。假设,如果处理器运行一个顺序扫描内存区域的程序,上述简单策略应该具有完美的准确率和覆盖率(定义见第 4 节)。然而,如图 3 所示,随着并行度的增加,这 2 个指标都会下降。使用 10 个线程,准确率下降到 85%,而覆盖率仅为 81%。优化的预取解决方案 Leap[5]尝试使用基于监测窗口选择最适预取偏移的在线预取算法来改进远端内存系统。然而,它的算法不能充分利用预取的好处。根本原因在于,Fastswap 和 Leap 都只从缺页或扫描页表中的访问位中获知访问信息,这些信息是不频繁、不准确、陈旧的且获取成本高昂。

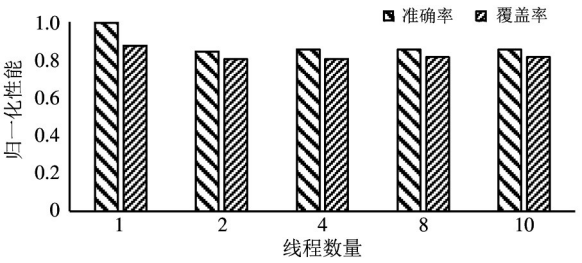

图 3 当线程数量变化时,准确率与覆盖率的变化

为了展示软硬件协同框架的优势,本文将其与 Fastswap 集成。从软硬件协同框架的角度来看,Fastswap 是一个应用程序。特别是,本文将 Fastswap 的默认顺序预取策略替换为基于流的策略,该策略使用软硬件协同框架提供的丰富应用程序访问信息进行训练。为了克服 Fastswap 中的可扩展性瓶颈,本文在单独的异步数据路径中运行软件协同框架和新算法,独立于传统 VMS 路径。本文选择优化 Fastswap 是因为它在大多数情况下比 Leap 具有更好的性能。

1.3 页面交换和驱逐

页面交换在现代数据中心中起着关键作用。例如,谷歌和 Meta 都部署了依赖内核页面交换来识别冷内存的远端内存系统[1-2]。在 Linux 中,页面交换使用近似最近最少使用(least recently used,LRU)列表来选择要驱逐的页面。通常,内核在 2 种情况下更新这些列表:(1)内核在被某个内核函数(例如写时复制)使用时主动更新页面访问标志位;(2)内核调用守护线程(即 kswapd)定期扫描页表中的访问位,然后相应地调整它们在列表中的顺序。然而,这 2 种方法都会产生不频繁、不准确、陈旧的页面访问信息。因此,使用此类信息排序的 LRU 列表无法捕获实际的内存访问模式。

本文通过使用软硬件结合框架的丰富内存访问

信息替换基于访问位的方法来改进默认 LRU 列表。此外,当有替换请求时,替换框架会优先考虑具有流式访问模式的页面,因为与随机访问的页面相比,它们不太可能被再次使用。

1.4　其他类型的远端内存系统介绍

应用集成的远端内存系统。这类系统需要通过修改应用来达到访问使用远端内存的目的,一些研究通过构建定制的应用接口或数据结构[13-14],让用户更方便且更加灵活地使用远端内存。然而,与基于内核的远端内存系统相比,这种方式丧失了透明性与通用性。用户不但需要重构代码,而且引入了复杂的远端数据管理问题。

虚拟化的运行时远端内存系统。这类系统改变了其中的内存管理组件,例如 Java 虚拟机(Java virtual machine, JVM)的垃圾收集[15],使它们对远程内存更加友好。尽管如此,它们被限定在特定语言(如 JVM 的用户),因此通用性有限。

基于总线扩展的远端内存系统。这种系统通过一致性协议扩展来实现远程内存的访问[16-18]。尽管这种方式有望完全实现透明性、通用性和高性能的远端内存系统,并引起极大的关注[16-17]。但是,它有 2 个限制:(1)它无法利用本地内存作为缓存来减轻应用使用远端内存造成的性能损失;(2)由于同步 load/store 指令可能经过多个节点进行转发,其延迟高达几微秒,因此每个未完成的内存访问都需要至少持有一个硬件资源,直到操作完成[18]。因此,数据中心运营商更倾向于在一个机柜搭建基于总线扩展的远端内存系统[7]。

2　软硬件协同框架系统设计

图 4 描述了软硬件协同框架的系统架构。本文在 MC 中部署了一个热点页面提取表。该单元截获所有 LLC 未命中的内存请求,将缓存块粒度信息转化为页粒度信息,然后其将热点页面信息发送到保留的缓冲区。MC 向缓冲区尾部写入访存信息,同时更新写指针。而专用的软件线程不断从缓冲区头部读取访存信息,然后通过反向页表将物理页面转化为虚拟页面与进程号,给替换模块与预取模块使用。

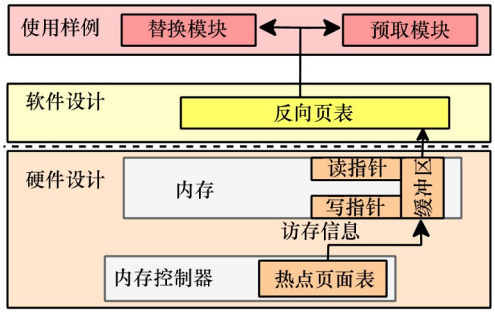

图 4　软硬件协同的框架结构图

2.1　热点页面表

内存控制器可以收到所有缓存缺失信息。简单地将所有这些信息提供给操作系统会消耗过多的内存带宽。另外,如果软件负责过滤信息,则增加了软件复杂度和计算资源。本文希望提供尽可能多的内存访问信息以接近应用实时页面访问行为。为了在不消耗过多内存带宽的情况下实时输出内存信息,本文添加了一个轻量级的热页检测模块,在 MC 中将基于缓存块粒度的访问转换为热点页面级别的访问信息。

热点页面表过滤了写类型的请求,仅处理读类型请求,原因有 2 个:(1)任何读操作都会立即生成读请求,而写操作会首先生成读请求并将数据写入处理器缓存,写内存的请求会在缓存发生替换的时候产生;(2)远程直接数据存取(remote direct memory access, RDMA)网卡(network interface card, NIC)使用 DMA 写入将页面从远程获取到本地内存中时,导致大量写入访问。没有简单的方法将它们与应用程序生成的正常访问区分开来。

为了要识别热点页,一种直接的方法是跟踪所有物理页的访问次数,然后在一个时间窗口(例如,100 μs)内找到被访问过 N 次的物理页。然而,这种方法需要大量的硬件资源,这些资源无法放入 MC 中。因此,本文构建了一个热点页面提取表来仅跟踪 M 个页面的访问计数。在图 5 中,热点页面表被组织为具有 LRU 替换($M=64$)的 16 路 4 组关联缓存。表中的每个条目都记录了物理页面号(physical page number, PPN)、读取访问次数、一个发送位表示该条目被识别为热页并被提取,以及一个用于替换的 LRU 位。PPN 的最低 2 位用作集合索引。

物理页面号	访问次数	发送位	LRU位	
8001	8	1	1	重复监测
9001	7	0	1	没有足够访问次数
7340	8	0	1	发现热点页面

图 5　热点页面表结构图

现在可以详细描述整个流程。当热点页面表接收到内存访问请求时,将其缓存块对齐的地址转换为物理地址,并使用物理页面号来查询表条目。如果没有找到,则将其插入表中。如果找到,检查发送位是否被置 1,如果是,不进行处理。否则,增加这一项的访问次数。一旦访问超过阈值 N,就提取物理页号并将其标记为热页。热点页面表中的项数越多,可以同时跟踪的物理页就越多。本文使用 4 组的热点页面表,即最多可以同时跟踪 64 个不同的物理页面。

N 的影响。一个 4 kB 的页面包含 64 个缓存块,因此 N 的范围是 1 ~ 64。如果 N 太小,例如 N = 2,则会提取到更多的热页面,并且提取的内存请求更接近应用程序的实时页面访问。但是,它包括对同一页面的更多重复提取,从而导致消耗更多的内存带宽。表 1 显示,当 N 小于 8 时,提取的热点页数显著增加,传输过程中更多的内存带宽被消耗,例如 PageRank,因此应用程序性能下降(3% for N = 4)。如果 N 太大,例如,N = 32,一个热点页面可能在被访问 N 次之前就被替换,对提取的内存信息造成缺失,影响预取覆盖率和应用性能。为了在内存带宽消耗和页面提取的及时性之间取得最佳平衡,本文选择 N = 8 作为默认值。

表 1　N 变化时,提取的热点页面数量占访存请求个数占比(%)

N	2	4	8	16	32
K-means	1.72	1.63	1.59	1.56	1.54
PageRank	11.72	4.45	1.55	1.07	0.84
CC	5.18	2.16	1.48	1.19	1.02
LP	3.96	2.42	1.84	1.47	1.26
BFS	4.01	2.36	1.77	1.44	1.23

多个内存通道的影响。当多个通道设置为 interleave 模式时,同一物理页面的不同缓存块被分配到不同的通道中。在这种情况下,热点页面表需要减少 N。虽然这可能会导致重复的热页面提取,但可以在软件中对它们进行去重。当多个通道没有设置为 interleave 时,从不同的 MC 中提取不同的热页,可以在软件中将它们合并。

2.2　反向页表

系统软件持续监控内存中缓冲区的热点页面信息,然后由反向页表对这些访存记录进行处理。这些访存记录仍然是物理地址。然而,虚拟内存系统需要虚拟地址以便于更好地发现访存规律[19]。Linux 内核中的有默认的反向映射表就是为此目的而设计的,但默认的反向映射表至少需要 4 次内存访问才能完成一次转换,这样的开销过大,因此需要设计一个简易且快速的反向页表。

本文构建了反向页面表,这是一种为软硬件协同框架的需求量身定制的轻量级翻译机制。反向页表被组织为一个由物理页号为索引的数组。这个数组存储在操作系统预留的内存区域,为了避免一致性问题,需要将这块内存区域的属性设置为不经过缓存。图 6 显示了每一项反向页表的具体结构,它包含 16 位的进程号与 40 位的虚拟页面号。每次查询物理页面对应的虚拟页面号与进程号时,反向页表首地址与物理页面号相加的地址就是当前物理页面号反向页表项的地址。

图 6　反向页表项结构

3　使用样例

基于软硬件协同框架的完整访存信息,本文构建了 2 种使用样例展示软硬件协同框架如何提升虚拟内存系统的性能,即一个新的基于完整访存信息的 LRU 的页面替换系统和一个基于 RDMA 的远端内存的预取系统。

3.1　页面替换

Linux 内核中现成的 LRU 页面列表是根据页表访问位以及其是否属于活跃链表进行排列的。本文构造一个由软硬件协同框架驱动的页面替换系统来

改进它。首先,为了绕过内核复杂的数据路径,本文构建了一个单独的数据驱逐路径。一旦替换系统检测到整个服务器存在内存不足的情况,它将选择一组要收回的页,并指示内核完成收回过程。其次,替换系统基于 LRU 规则维护了一组页面列表,该列表根据完整的来自软硬件协同框架的访问信息构建。最后,替换系统针对不同的访问模式,使用不同的替换策略。当选择驱逐的页面时,本文优先对流访问之后的页面进行驱逐,这是因为流访问模式的页面与遵循随机访问模式的页面相比,有更小的再次被访问的可能性。本文使用页面流表来标识哪些页面属于流(详见 3.2 节)。

3.2 页面预取

基于软硬件协同框架提供的完整访存信息,本文构建了一个基于流的预取器,称为软硬件协同框架驱动的预取器。这个预取器能够指导 Fastswap 从远程异步地预取(无需等待缺页发生)。同时,本文还将上述软硬件协同框架驱动的替换系统集成到 Fastswap 中。

预取器中的一个核心思想是预取页面流,即一个固定步幅的访问序列。一个应用程序可以同时具有多个页面流。预取器维护一个流训练表,每个条目代表一个页面流。如图 7 所示,流训练表有 64 个条目,这在大多数情况下就足够了。使用 LRU 策略管理条目。此外,一个流训练条目最多可以记录 5 个虚拟页号。

图 7 流训练表结构图

预取器工作流程如下:当新的{虚页号,进程号}到达时,首先找到与新进程号相等的所有流训练表中的条目;然后将新虚页号与条目中虚页号数组的第一项进行比较,如果两者差值小于一定阈值(设置为 64),那么预取器则认定新页面属于这个页面流;最后,预取器会将新页面的虚页号依次存储在其流训练表对应条目中的虚页号数组中。当虚页号数组已满时,将计算这个流的步长(即虚页号数组中相邻虚页号之间的距离),并找到出现次数最多的步长。这样,预取器确定用于预取的虚页号是页面流的结束页 $+i\times$步长,其中 i 是预取偏移量。为了防止系统和网络延迟对预取的影响,本文设置 $i=100$ 以确保页面在应用程序访问之前及时到达。在图 7 中,新页面{2,8}到达,确定其属于流训练表中的第一个条目,计算步长等于 2,那么预取的页面是 $8+2\times100=208$。

4 具体实现

本研究在 x86 服务器上构建了软硬件协同框架和 2 个使用样例的原型系统。本研究在用户空间中实现了大部分软件功能,只在内核空间中留下一小部分用于与 VMS 交互。这种方法很灵活,因为它可以实现快速的开发迭代、易于编程和在线系统升级[20-21]。本研究构造了一些系统调用和共享内存,使软硬件协同框架可用于用户空间代码。对于这 2 个使用样例,本研究在用户空间中构建了基于流的预取器和 LRU 页面列表。它们通过新的系统调用与内核进行通信。

4.1 硬件测试平台

大多数 MC 都是供应商锁定的,并与 CPU 和 LLC 打包在一起成单个模具,因此不能修改 MC 来实现热点页面提取表。本研究在 x86 服务器上构建了一个仿真验证原型,如图 8 所示。它有几个主要组件。为了模拟记录单元中的跟踪部分,本研究将一个名为 HMTT[10-11] 的基于硬件的内存跟踪工具附加到 MC 和 DRAM 芯片之间的链接上。HMTT 工具可以通过在DDRx内存总线上侦听通过MC的DDR

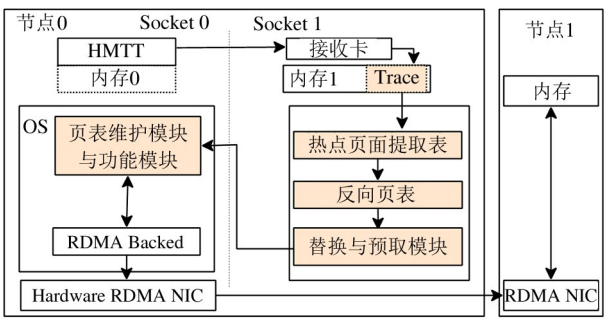

图 8 测试平台结构图

数据包来实时捕获内存访问信息。本研究将 HMTT 插入 DIMM 插槽,并将 DRAM 安装到 HMTT 板上。HMTT 可以配置为监视一定范围的物理内存空间。

HMTT 将在一个节点收集的内存跟踪通过 PCIe 转发到另一个接收节点,而后者又将访存信息保存在其本地固态硬盘(solid state disk, SSD)上。为了在同一节点捕捉和使用访存信息,本研究修改了 HMTT 配置。如图 8 所示,HMTT 位于 Socket 0 的 DIMM 和内存 0 之间。它可以获得对内存 0 的实时内存访问。通过将操作系统配置为仅在内存 0 上运行,HMTT 可以监控所有正在运行的应用程序的内存访问。内存控制器中的热点页面提取表仅输出热页,因此它消耗的内存带宽很少,并且可以写入与应用程序相同的 DRAM,而 HMTT 则输出所有内存访问信息。为了避免在同一个内存上的内存带宽干扰,本研究配置 HMTT 以将收集在内存 0 中的访存信息写入内存 1,通过 PCIe 将访存信息发送到硬件的接收卡,该接收卡负责将它们写入内存 1 中的保留区域。

本研究的原型系统在功能上等同于本研究的设计,并且足以用于验证目的。然而,该原型存在以下限制:(1)HMTT 受限于其现场可编程门阵列(field progrmmable gate array, FPGA)频率,因此在捕获吞吐量上存在上限;(2)它需要大量的工程工作(例如,将 HMTT 与单独的 PCIe 设备连接)并产生更高的构建成本;(3)它增加了延迟,因此影响了访存的及时性。如果本研究的设计集成到芯片或 MC 中,例如使用开源硬件平台[22],这些问题就不存在了。本研究的硬件实际部署是非侵入式的,应该可以在各种 CPU 平台上运行。这一探索留给未来的工作。

4.2 运行软件

本研究的设计原则是在用户空间中实现大部分功能并最小化内核更改。这使其能够快速迭代并无缝升级软硬件协同框架。

内核模块。只有一部分反向页表是在内核空间中实现的。具体来说,在操作系统启动时,为反向页表保留了一个连续的物理内存区域。该表是图 6 描述的反向页表条目的平面数组,按物理地址索引。当软硬件协同框架首次启动时,其将扫描所有现有页表以构建初始反向页表。同时为 VMS 中的一些函数(例如 pte _ clear、set _ pet)插入轻量级回调函数,以保持反向页表更新。为了进行高效通信,本研究还实现了一些用于低吞吐量控制任务的系统调用和用于高吞吐量数据共享的共享内存。

用户态模块。本研究在用户空间中实现了过滤模块和反向页表。本研究用一个线程模拟热点页面表,同时用一个固定线程来轮询循环缓冲区的读写指针。只要缓冲区不为空,线程就会从读取热点页面信息。在当前的实现中,线程将直接将访存信息发送到热点页面表中。

替换模块。替换模块使用 C 语言来实现,它通过共享内存接受来自软硬件协同框架的输入。本研究还为其实现了一个新的系统调用,系统调用接受来自用户空间的替换请求,然后将页面替换到远端内存。然后,它会返回替换成功的页面号。为了提高可扩展性,本研究不等待 I/O 传输完成,而是在发出换出请求后立即返回。一旦 I/O 传输完成,内核就会相应地更新页面状态。

预取模块。与替换模块类似,预取模块也是用 C 语言实现的,并通过共享内存与软硬件协同框架通信。由于远端内存系统具有波动的工作负载,预取器可以通过调整预取线程的数量来自动扩展其计算能力,每个线程都接受来自软硬件协同框架的输入并通过新的系统调用独立地与 Fastswap 通信。

本研究改进了内核的预取接口,因为它受限于同步的数据路径和粗粒度锁。因此,本研究将交换接口分为 2 个步骤。第 1 步是寻找空余内存并将预取请求发送到基于的 RDMA 交换系统,例如 Fastswap 中的 Frontswap。第 2 步立即将控制权返回给调用线程,无需等待请求完成。一旦请求完成,RDMA NIC 发送一个中断,中断处理程序(构建在工作队列子系统之上)将完成页表相关操作。此外,本研究用细粒度的页面锁替换了粗粒度的锁,这大大提高了并行度。本研究对内核的所有更改都不会干扰 Fastswap。在 Fastswap 之上运行的应用程序无需任何更改即可轻松享受软硬件协同框架带来的优势。本文将与 Leap[5] 或 Infiniswap[4] 等其他分类存储系统的集成留给未来的工作。

5 测 试

在本节中,首先评估 2 个构建在软硬件协同框架的使用奖励:使用实际应用程序进行页面替换和预取。然后,对软硬件协同框架本身进行了测试。

测试台和工作负载。如图 8 所示,测试台有 2 个通过 RDMA 连接的服务器,一个是运行软硬件协同框架的本地服务器,另一个用作远程内存。这 2 台服务器都有一个 14 核 Xeon E5-2680 CPU、35 MB,20 路 LLC、64 GB DRAM 和一个 56 Gbps Mellanox ConnectX-4 NIC。表 2 显示了本文实验中使用的 14 个大规模内存应用程序。

表 2 测试应用

应用名称	使用内存/GB	使用核数	内存带宽/(GB/s)
GraphX (BFS,CC,LP,PR)	40	8	3 ~ 6
Spark-Bayes	37	8	3 ~ 6
Spark-K-means	40	8	2 ~ 4
OMP-K-means	3.2	4	2 ~ 3
High Performance Linpack	1.2	2	3 ~ 6
NPB (CG,FT,LU,MG,IS)	1 ~ 7	2	3 ~ 6
QuickSort	4	1	1 ~ 2

5.1 页面替换

对于这 2 个用例,本文首先评估替换框架。本文使用 Linux 默认替换机制及其 kswapd 作为对比项。本文使用由软硬件协同框架的完整访存信息支持的 LRU 列表、流访问模式优先替换以及 2 种优化同时使用来评估替换框架。使用 Fastswap[3] 作为基础交换系统,因此页面被替换到远程内存。使用应用程序完成时间加速比和访问远端内存次数减少百分比 2 个指标进行比较。访问远端内存次数减少百分比定义为与 Linux 默认替换机制相比减少的百分比,其值越高越好。

图 9 和图 10 报告了使用软硬件协同的替换框架的 8 个实际应用程序的加速比和读远端次数减少比例。其中,美国国家航天局并行基准测试(NAS parallel benchmarks, NPB)-多重网格(multigrid, MG)、NPB-快速傅立叶变换(fast Fourier transform, FT)和 Spark Graphx 的访问中大多为随机访问,因此,具有 LRU 优化的替换方法比流替换方法更好。相比之下,NPB-整数排序(integer sort, IS)的访问模式以流模式为主,因此流替换方法让其性能提升最高(1.3 倍)。其原因是通过优先对流页面的访问,尽可能地保留了随机页面到本地内存,降低了对远端内存的读取次数(图 10)。由于 K-means 的访问与 Stream 类似,因此替换框架对其都不起作用。

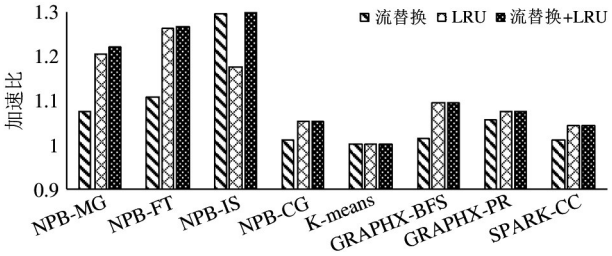

图 9 不同替换策略下的加速比

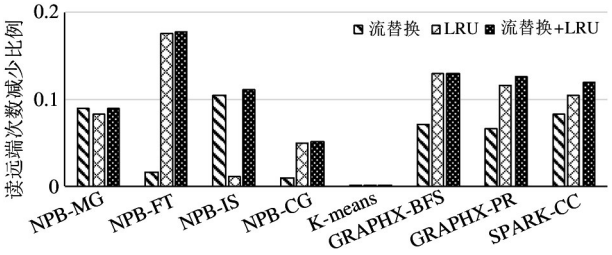

图 10 不同替换策略下读远端次数减少比例

5.2 页面预取

本文将软硬件协同的预取框架与 Fastswap 集成,并使用各种实际应用程序进行测试,将其与 Fastswap 与 Leap 进行比较。然而,由于数据路径实现速度慢,Leap 的性能比其他 2 个差得多。因此,为了更好地说明,本文省略了 Leap 的结果。本文使用 2 个指标来衡量预取性能:准确率是预取的页面命中次数与发出的预取页面数的比;覆盖率是预取的页面命中次数与没有预取时访问远端内存数量的比。本文还使用应用程序完成时间与其在 Fastswap 的完成时间的比来统计新框架对应用的加速比。对于所有测试,本文将本地内存设置为应用程序占用空间的 50%。

图 11 显示了使用软硬件协同预取框架对应用

程序完成时间的加速比。对于 QuickSort 和 K-means-OMP,与没有远端内存的本地场景相比,即使应用一半的内存在远端,软硬件协同框架也不会导致应用性能下降。这是因为软硬件协同预取框架可以准确预测应用访问模式并将页面异步预取到本地内存中,完全消除缺页。使用软硬件协同预取框架对所有应用的平均加速比为 32.2%,其相比于 Fastswap 最多加速 48.2%,最少为 11.4%。

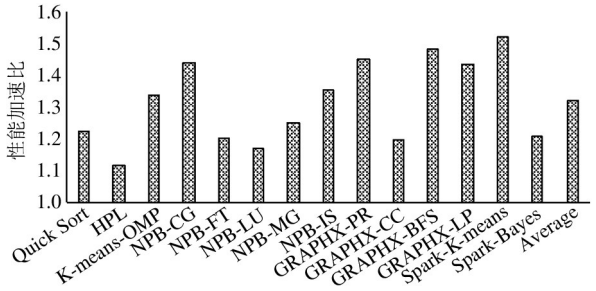

图 11　应用在软硬件集合预取框架下的加速比

同时,本文进一步比较了它们的准确性和覆盖率。如图 12 所示,软硬件协同预取框架的平均精度超过 97%,这意味着只发出少数错误预取,并且本地内存没有受到污染。然而,Fastswap 的平均准确率仅为 71.1%。图 13 显示了软硬件协同预取框架的预取覆盖范围分为 2 部分:流命中和 swap 命中。

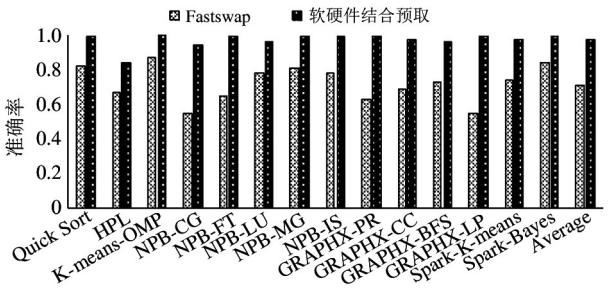

图 12　软硬件集合的预取器的准确率

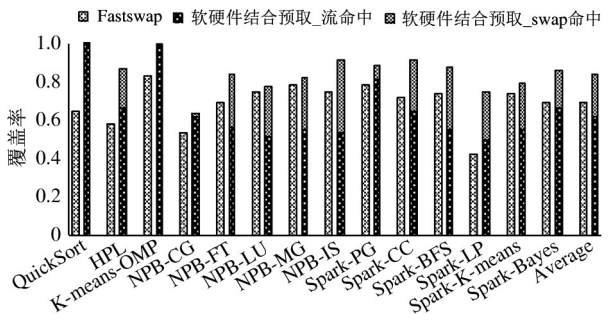

图 13　软硬件集合的预取器的覆盖率

swap 命中是底层远程系统在出现缺页时发出的预取页面数。本质上,软硬件协同预取框架是对运行在同一服务器上的远程内存系统的补充。流命中是流形成后发出的预取页数,不会导致缺页产生。实验表明 QuickSort 预取覆盖率最高,K-means 应用程序完成时间与其内存都在本地内存时非常接近。

5.3　替换与预取结合测试

图 14 展示了当替换框架与预取框架结合在一起时对应用的影响。对于除 K-means 之外的所有应用程序,预取与替换框架结合时的性能都优于其他设置。通过共同使用同一张流训练表,预取框架减少了流页面的缺页次数,而替换框架减少了随机页面的缺页次数。由于 K-means 的随机页面较少,预取与替换框架结合时相比单独的预取框架几乎没有提升。

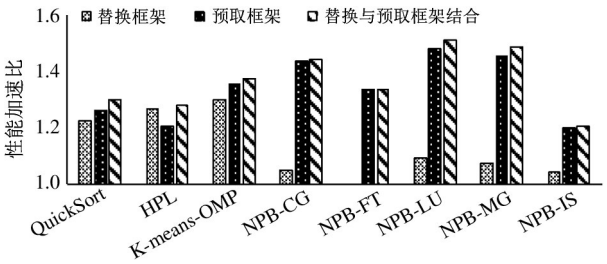

图 14　当预取与替换框架结合时,应用的加速比

5.4　硬件设计可行性

本文在 Verilog 中实现了热点页面表,以验证所提出硬件设计的可行性以及它们的内存带宽消耗。本文利用 CACTI[23] 使用 22 nm 技术节点估计面积和静态能量消耗。热点页面提取表需要的面积为 0.000 252 mm^2,它的静态功耗是 0.0959 mW。

6　结　论

本文介绍了一种软硬件协同框架,通过在内存控制器中增加热点页面表,将热点页面信息不断写入内存中的缓冲区,软件通过对缓冲区的不断监控,获取应用实时访存的物理地址,从而弥合操作系统和应用程序内存访问行为之间的语义鸿沟。同时,本文建立了一种高效的翻译机制,将物理地址高效地翻译为虚拟地址与进程号。通过对访存信息的学

习,本文构建了高精度的异步预取框架与替换框架,降低了应用关键数据路径的开销,提升了远端内存系统的性能。值得一提的是,本文利用内存跟踪工具构建了一个原型仿真系统,证明了预取与替换框架所带来的好处。本工作可能为改进各种基于虚拟内存的系统打开大门。

参考文献

[1] LAGAR-CAVILLA A, AHN J, SOUHLAL S, et al. Software-defined far memory in warehouse-scale computers [C]// Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems. Providence: ASPLOS, 2019: 317-330.

[2] WEINER J, AGARWAL N, SCHATZBERG D, et al. TMO: transparent memory offloading in datacenters [C]// Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. Lausanne: Association for Computing Machinery, 2022:609-621.

[3] AMARO E, BRANNER-AUGMON C, LUO Z, et al. Can far memory improve job throughput? [C]// Proceedings of the 15th European Conference on Computer Systems. Crete: Association for Computing Machinery, 2020:1-16.

[4] GU J, LEE Y, ZHANG Y, et al. Efficient memory disaggregation with infiniswap[C]// The 14th USENIX Symposium on Networked Systems Design and Implementation. Boston: USENIX, 2017: 649-667.

[5] AL MARUF H, CHOWDHURY M. Effectively prefetching remote memory with leap[C]//2020 USENIX Annual Technical Conference. Boston: USENIX, 2020:843-857.

[6] EISENMAN A, GARDNER D, ABDELRAHMAN I, et al. Reducing DRAM footprint with NVM in Facebook[C]// Proceedings of the 13th EuroSys Conference. Porto: EuroSys, 2018:1-13.

[7] LI H, BERGER D S, NOVAKOVIC S, et al. First-generation memory disaggregation for cloud platforms [EB/OL]. (2022-03-01)[2022-11-08]. https://arxiv.org/pdf/2203.00241v1.pdf.

[8] NAVIN SHENOY. A milestone in moving data[EB/OL]. (2019-03-11)[2022-11-08]. https://newsroom.intel.com/editorials/milestone-moving-data/.

[9] WANG C, QIAO Y, MA H, et al. Canvas: isolated and adaptive swapping for multi-applications on remote memory [J]. (2022-10-12)[2022-11-08]. https://arxiv.org/pdf/2203.09615.pdf.

[10] BAO Y, CHEN M, RUAN Y, et al. HMTT: a platform independent full-system memory trace monitoring system [C]//Proceedings of the 2008 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. Annapolis: Association for Computing Machinery, 2008:229-240.

[11] HUANG Y, CHEN L, CUI Z, et al. HMTT: a hybrid hardware/software tracing system for bridging the DRAM access trace's semantic gap[J]. ACM Transactions on Architecture and Code Optimization, 2014,11(1):1-25.

[12] TSALAPATIS A, GERANGELOS S, PSOMADAKIS S, et al. Utmem: towards memory elasticity in cloud workloads [C]// International Conference on High Performance Computing. Orléans: ISC, 2018:173-183.

[13] AGUILERA M K, AMIT N, CALCIU I, et al. Remote regions: a simple abstraction for remote memory[C]// 2018 USENIX Annual Technical Conference. Boston: USENIX, 2018:775-787.

[14] RUAN Z, SCHWARZKOPF M, AGUILERA M K, et al. AIFM: high-performance, application-integrated far memory[C]// The 14th USENIX Symposium on Operating Systems Design and Implementation. Banff: USENIX, 2020:315-332.

[15] WANG C, MA H, LIU S, et al. Semeru: a memory-disaggregated managed runtime [C]// The 14th USENIX Symposium on Operating Systems Design and Implementation. Banff: USENIX, 2020:261-280.

[16] CALCIU I, IMRAN M T, PUDDU I, et al. Rethinking software runtimes for disaggregated memory[C]// Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. Virtual: Association for Computing Machinery, 2021:79-92.

[17] GOUK D, LEE S, KWON M, et al. Direct Access, high-performance memory disaggregation with DirectCXL[C]// 2022 USENIX Annual Technical Conference. Carlsbad: USENIX, 2022:287-294.

[18] WANG L, ZHANG X, LU T, et al. Asynchronous memory access unit for general purpose processors [J].

(2021-12-26)[2022-11-08]. https://arxiv.org/pdf/2112.13306.pdf.

[19] PAN H, LIU Y, LU T, et al. LSP: collective cross-page prefetching for NVM[C]//2021 Design, Automationand Test in Europe Conference and Exhibition. Grenoble: DATE, 2021:501-506.

[20] HUMPHRIES J T, NATU N, CHAUGULE A, et al. ghost: fast & flexible user-space delegation of linux scheduling[C]//Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. Virtual: Association for Computing Machinery, 2021:588-604.

[21] MARTY M, DE KRUIJF M, ADRIAENS J, et al. Snap: a microkernel approach to host networking[C]//Proceedings of the 27th ACM Symposium on Operating Systems Principles. Huntsville: Association for Computing Machinery, 2019:399-413.

[22] XU Y, YU Z, TANG D, et al. Towards developing high performance RISC-V processors using agile methodology [C]//2022 55th IEEE/ACM International Symposium on Microarchitecture. Chicago: IEEE, 2022:1178-1199.

[23] SHIVAKUMAR P, JOUPPI N P. CACTI 3.0: an integrated cache timing, power, and area model, Technical Report 2001/2[R]. Palo Alto: Western Research Lab Research Report, 2001.

Hardware-software co-designed remote memory system

LI Haifeng[*,**], LIU Ke[*], CHEN Mingyu[*,**]

([*]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**]University of Chinese Academy of Sciences, Beijing 100049)

Abstract

Currently, page prefetching and page replacement in remote memory systems are limited due to the semantic gap between the operating system and the application access behavior. This paper proposes a hardware and software co-designed remote memory system to bridge the gap. A hot page detection in the memory controller is added to transfer real-time memory access information to the operation system. At the same time, through the learning of memory access information, an asynchronous prefetching framework and replacement framework is built to reduce the cost of application critical data paths and improve the performance of the remote memory system. The memory tracking tool is used to emulate the memory access unit in the memory controller and build a prototype system. Experiment results show that the prefetching framework can achieve more than 90% accuracy and coverage. Compared with the Fastswap, the performance is improved by 59%. Compared with the default replacement framework of the kernel, the replacement framework improves application performance by 30%.

Key words: remote memory system, hardware-software co-designed, prefetching framework, replacement framework