新型存储硬件下分层内存系统中的缓存效率

柳子淇[1)]

[1)](华中科技大学 计算机科学与技术学院,武汉市 中国 430097)

摘 要 随着计算机系统在海量数据、深度学习和复杂模拟面前追求更高性能，对更大更高效内存的需求也在不断增加。随着非易失性存储器（NVM）技术的进步、新型接口逻辑（ZNS SSD）的引入以及新兴通信协议（CXL）的出现，传统的存储器层次结构也在不断演变。分层内存系统使用多个存储级别来平衡性能、成本和功耗。然而,利用硬件实现的分层系统效率低下，这与硬件层面缺乏对应用内存访问模式的信息有关。不过，最近的研究表明，可以通过操作系统层面的优化来解决这一问题。本文也提及了如何使用新硬件进行内存扩展，例如将ZNS SSD 用作交换分区，将CXL-SSD 用作扩展内存。在最新研究部分，重点介绍了Johnny Cache[1]，它在操作系统层面优化了页面分配策略，以减轻分层内存系统硬件实施的低效率。通过实施静态和动态页面分配策略，Johnny Cache 在分层DRAM+PMEM 系统上实现了比Linux 和HeMem 高达5 倍的性能提升。这篇综述分析了分层内存系统硬件实现方面的挑战，并探讨了操作系统层面的内存系统设计与优化。

关键词 内存系统, 页面分配, 页面映射策略, 操作系统, 新型存储硬件

Cache Efficiency in Tiered Memory Systems with Novel Storage Hardware

Ziqi Liu [1)]

[1)](1)(Huazhong University of Science and Technology, Wuhan, China, 430074)

Abstract As computer systems strive for higher performance in the face of big data, deep learning, and complex simulations, the need for larger and more efficient memory is on the rise. The traditional memory hierarchy is evolving with advancements in non-volatile memory (NVM) technology, the introduction of new interface logic (ZNS SSD), and emerging communication protocols (CXL). Tiered Memory Systems use multiple storage levels to balance performance, cost, and power consumption. Inefficiencies in hardware-implemented systems have been linked to a lack of a holistic view of application memory access requirements at the hardware level. However, recent research suggests that this can be addressed through optimizations at the operating system level. The paper explores the use of new hardware for memory expansion, such as ZNS SSDs as swap partitions and CXL-SSDs as extended memory. In the state-of-the-art research section, the focus is on Johnny Cache, which optimizes page allocation policies at the operating system level to mitigate inefficiencies in hardware implementations of Tiered Memory Systems. Implementing static and dynamic page allocation policies, Johnny Cache achieves a performance improvement of up to 5 times compared to Linux and HeMem on a tiered DRAM+PMEM system. This review analyzes challenges in hardware implementations of tiered memory systems and explores memory systems at the operating system level.

Keywords Memory Systems, Page Allocation, Page Mapping Strategies, Operating Systems, New Storage Hardware

1 引言

如今，计算机系统的性能和数据处理需求不断攀升。随着大数据、深度学习和复杂模拟等应用的广泛应用，计算机系统需要更大、更高效的内存来应对不断增长的数据量和计算复杂性。随着非易失性内存（NVM）技术的进步，新型接口逻辑（ZNS SSD），和新兴通信协议（CXL）的出现，传统的内存层次结构正经历着深刻的变革。这些新技术往往能提供更高的存储密度和更低的访问延迟以及一些新的硬件特性，为分层内存系统带来了新的挑战和机遇。

2 背景与动机

2.1 分层内存系统

分层内存系统（Tiered Main Memory Systems）是一种存储体系结构，采用多个层次的存储设备，每个层次都具有不同的性能、容量和成本特征。每一层次都具有不同的性能和容量特征。这种设计旨在平衡性能、成本和功耗之间的权衡，以提高整个系统的性能和效率。在分层内存系统中，数据通常根据其访问频率和紧急性被分配到不同的层次。例如，经常被访问的数据可能会被存储在更快但容量较小的缓存中，而较少访问的数据则可能存储在主内存或辅助存储器中，因此大部分时间系统可以从更快速的存储器层次中获取所需的数据。其设计思想是在内存系统中引入多个层次的策略，以更好地适应不同类型的数据访问需求。这种架构的设计使得计算机系统在运行时能够更智能地管理数据的存储和访问，从而提高整体性能。

分层内存系统往往将DRAM与较慢但更丰富的存储层（SSD、PMEM、CXL内存扩展模块等）结合在一起。现有的大多数系统依赖于监视对数据的访问的软件守护进程，例如HeMem[2]其目标是：使得经常被访问的数据被迁移到DRAM，而不经常被访问的数据被迁移到较慢的存储层；分层内存系统也可以纯粹基于硬件实现，使用DRAM作为位于CPU和较慢存储层之间的“L4”缓存。

2.2 缓存映射

缓存映射策略是计算机体系结构中关键的设计决策，它定义了下一级存储设备中的数据块如何映射到上一级设备的缓存中。不同的分配策略和映射方法直接影响着缓存的命中率和响应时间。随着计算机工作负载的不断演变，设计出更加智能和适应性强的缓存分配映射方案变得尤为重要。

下面列举三种基本的缓存映射策略并简要分析其优缺点，分别是直接映射、全相联映射以及组相联映射。

首先是直接映射策略（Direct Mapping），直接映射是最简单的一种缓存映射策略。在直接映射中，每个主存块只能映射到缓存中的一个特定位置。具体来说，主存块的地址通过取模运算得到在缓存中的位置。其映射关系是固定的，不会发生变化。直接映射更适用于小容量缓存；其次，是全相联映射策略（Fully Associative Mapping），其灵活性较高，在全相联映射中，主存块可以映射到缓存中的任意位置，而不需要按照固定的映射关系，其映射关系可以根据需要进行调整。这对于较大的缓存空间是有利的，可以更好地避免冲突问题。然而，全相联映射的实现可能需要更多的硬件支持和操作系统管理；最后是组相联映射策略（Set Associative Mapping），其在一定程度上结合了直接映射和全相联映射的特点，组相联映射是直接映射和全相联映射的折中方案。在组相联映射中，缓存被分为多个组，每个组包含多个缓存行。主存块通过哈希函数映射到某个组中的某个缓存行。虽然映射关系是固定的，但是可以根据需要进行组内的调整。

在大多数情况下，直接映射往往适用于小型缓存，全相联映射适用于较大缓存，而组相联映射则提供了一种在两者之间平衡的选择，可以在一定程度上减少冲突，同时保持一定的灵活性。然而，选择相对折中的组相连映射并不总是是更好的方案，选择缓存映射策略需要在考虑缓存空间大小的同时，兼顾综合硬件实现的复杂性和操作系统管理的开销，例如，本文涉及到的Linux Page Cache，采用的则是全相联映射。

2.3 硬件实现的分层内存系统的问题

硬件实现的分层内存系统的效率低于预期。例如，实验已表明在Intel傲腾DRAM+PMEM存储系统中，以守护进程形式实现的数据迁移内存系统在性能上优于Intel的“内存模式”。（在“内存模式”中，CPU将DRAM用作对PMEM的直接映射缓存）现有研究普遍认为分层系统的硬件实现效率低一方面是因为硬件缺乏对应用程序内存访问需求的整体视图，另一方面，硬件实现过程中，限制了更复杂的缓存策略的使用。然而，较新研究[1]指出：硬件缓存的上述限制并非根因性的，可以在操作系统层面解决。Johnny Cache[1]证明了在先前基于硬件的系统中观察到的性能不佳是由于Linux的页面分配策略导致的缓存冲突所造成的，并且通过对页面分配策略进行简单的改进，就可以在几乎不引入额外开销的情况下降低冲突。

2.4 其他利用新硬件拓展内存的探索

ZNSwap[3]是一项利用ZNS(Zone Namespace) SSD提升Linux内存交换性能的技术。通过利用ZNS SSD，ZNSwap相比传统基于SSD的交换机制具有显著的性能优势。这些优势包括对不同内存访问模式的稳定吞吐量和显著降低的延迟,m从而提高整体系统性能。另一种扩展内存容量的方法是使用支持CXL(Compute Express Link)技术的SSD[4]。CXL是一种高速互连技术，可以将加速器和内存设备连接到CPU。通过利用支持CXL的SSD，可以构建一致性低延迟、可扩展的扩展内存系统。这种方法允许超越传统DRAM的内存容量限制，为内存密集型工作负载提供改进的性能。总之，ZNSwap和使用支持CXL的SSD为Linux系统中的内存性能提供了有希望的解决方案。这些方法为内存密集型工作负载提供了改进的性能、降低的延迟和更大的内存容量。

3 研究现状

这部分内容主要针对Johnny Cache进行展开讨论，关注了新型存储器件构建的分层内存系统中的页面缓存冲突问题。

3.1 Johnny Cache:缓存分配时的冲突避免

本文基于一个观察，即先前提到的硬件缓存的限制并非根本性问题，可以在操作系统层面进行解决。本文展示了先前硬件为基础的系统中观察到的性能不佳是由于缓存冲突，而这是由Linux的页面分配策略引起的，通过对页面分配策略进行简单的改进，可以在几乎没有额外开销的情况下减轻缓存冲突。

Linux的页面分配未考虑页面在硬件缓存中的物理位置。因此，Linux遭受了生日悖论的影响：DRAM缓存很大，但许多页面往往映射到可用缓存位置的一部分。本文提出了以下简单的静态页面分配策略以减少冲突：每次分配一个新页面时，使其物理地址映射到当前映射页面最少的缓存槽。上述策略即为静态策略。补充地，文中还提出了一种动态策略，通过分析页面的访问频率，区分热页面和冷页面。动态策略将新页面分配给访问频率最低的缓存槽，并通过在检测到冲突时动态重新映射页面来响应工作负载变化。作者意外的发现在许多工作负载中，静态策略已经有效减少了冲突，而动态策略的开销抵消了进一步减少冲突可能带来的性能提升。上述两种策略分别记为JC-static和JC-dyn，作者在一个分层的DRAM+PMEM系统上评估了这些系统，与Linux页面分配机制和HeMem（最新的基于软件的页面迁移系统）进行了比较。在绝大多数应用程序中，JC在性能上优于Linux和HeMem，性能提升至多高达5倍。像HeMem和JC-dyn这样涉及分析的方法会受到分析和迁移开销的影响，并且在某些工作负载中无法检测到热页面。相反，像JC-static这样在分配时避免DRAM缓存中的冲突的方法是健壮且足以在大多数工作负载中实现接近最佳的性能。

总的来说，本文提出了以下贡献：

• 通过对操作系统页面分配算法进行轻微修改，观察到硬件管理的DRAM缓存可以变得高效。

• 将冲突避免作为分层内存系统页面管理的首要原则，而不是依赖数据迁移。

• 设计、实现和评估了超越最新页面迁移系统的页面放置策略。

3.1.1 设计

本文的设计基于这样一个观点：即只要缓存中的冲突很少发生，DRAM缓存就是有效的。冲突发是指当两个及以上数据项映射到相同的缓存位置，如果这些数据项依次被访问，缓存性能就会下降。以Intel MEM MODE为例进行分析。当DRAM Cache MISS时，从PMEM抓取一个W，并复制到DRAM Cache、CPU Cache。但X对应的缓存槽位被X占据了，此时：若X是干净的，开销为一次PMEM读，为最好情况如图1（a）所示；若X脏，写回X其开销为一次PMEM写+PMEM读。另外，CPU cache可能会逐出Y，若Y脏，写回DRAM，可能还会再从DRAM中逐出一个Z，又写回PMEM，此为最差情况，如图1（b）所示。

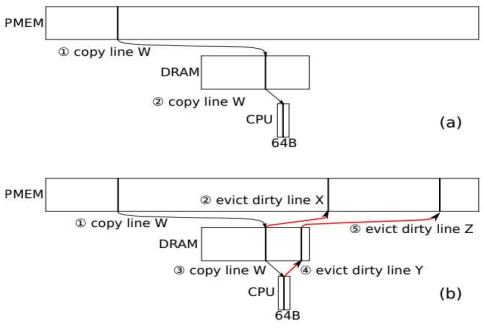


图1 缓存冲突时可能出现的情况

无论事件的确切顺序如何，冲突都是昂贵的。从PMEM中读取（在AppDirect模式下）比

从DRAM中读取慢3.2倍，写入慢4.4倍。导致两次写回PMEM的读取比从DRAM中读取慢9.7倍，导致两次写回的写入比从DRAM中写入慢7.2倍（导致两次写回PMEM不完全等同于执行两次对PMEM的写入，后者将慢8.8倍）

静态策略：静态策略将映射到相同DRAM缓存位置的分配页面数量最小化。假设一个可以存储D个页面的DRAM缓存，静态策略分配前D个页面，使它们映射到不同的缓存位置。接下来的D个页面在每次被分配时，它们仅可能与单个其他页面发生冲突，以此类推。

动态策略：动态策略对内存访问进行采样，计算每个页面和每个缓存位置的热度。当分配新页面时，内核将其映射到最冷的可用位置。冲突避免守护程序监视在相同缓存位置的热页面之间的冲突。当两个页面，映射到相同的DRAM缓存位置，都经常被访问时，其中一个页面会被映射到不同的缓存位置。

静态策略的主要优势在于不需要监视内存访问，因此没有额外开销。静态策略背后的直觉是，通过最小化在缓存中重叠的页面数量，显著减少热数据项之间冲突的可能性。实际上，在大多数工作负载中，即使在内存占用量远远超过可用DRAM的情况下，最多只有几GB的数据是热的。这种最小化重叠的思想使得热页面之间的冲突概率显著降低。例如，考虑一个分配数据量是可用DRAM大小的两倍的应用程序，其中5%是热数据。按照静态策略分配页面，可以确保每个缓存位置映射到2个页面。给定的热页面有5%的机会与另一个热页面竞争缓存，有95%的机会与一个冷页面竞争。换句话说，在上述情况下,大多数热页面与一个冷页面“配对”，因此不会经常被DRAM缓存中驱逐。

动态策略在页面分配时做出更为明智的选择，并且守护程序在分配时可能被忽略的冲突。动态策略借用了软件迁移系统中的“热度”概念，但这里的热度是用于跟踪和避免热页面之间的冲突，而不是像软件迁移策略中那样将热页面迁移到DRAM。

3.1.2 实现

在本节中，描述了页面分配策略和迁移守护程序的实现。

文章的缓存分配策略是通过内核初始化、页面初始化、页面故障处理和页面取消映射处理等钩子函数实现的，统一的逻辑由名为Johnny Cache（JC）的框架管理。文中假设了一个直接映射的1路缓存，与Intel在分层DRAM+PMEM系统的“内存模式”中的实现相似。虽然DRAM以缓存行粒度缓存数据，但内核只能以页面粒度分配和迁移数据。因此，JC维护的所有元数据都是在页面级别上。在内核启动时，查询处理器的内存控制器以找出DRAM缓存的大小，称之为缓存容量。由于缓存是直接映射的，系统中的每个页面都映射到缓存中的唯一索引，我们称之为bin。页面的bin是其页面框架号（页面的第一个字节的物理地址/页面大小）对缓存容量取模的结果。此外，缓存的每个bin都有一个热度。热度的定义取决于策略。例如，对于静态策略，它对应于映射到该bin的已分配页面的数量。

与Linux的默认页面分配策略类似，本文使用一种懒惰的页面分配机制：只有在首次访问时才物理分配页面。具体地，该页面放置策略通过hook函数关联到缺页异常处理程序来实现。该框架维护一个按热度排序的可用页面的bin列表。当发生页面故障时，从具有最低热度的bin中返回一个页面。类似地，每当释放一个页面时，都会调用内核解映射处理程序，并让当前的分配策略感知到映射已取消。

3.1.3 评估

本文仅展示部分实验评估结果，具体实验配置参见原文。文中使用一个大型数组做读取性能对比，如图2，数据集大小为480GB，其中2%为热数据（9.6GB）。(a) 线程数量少，热页面很少被检测到，HeMem 的性能不理想。(b)线程数量多，JC 比HeMem 更快达到最佳性能。 文中评估

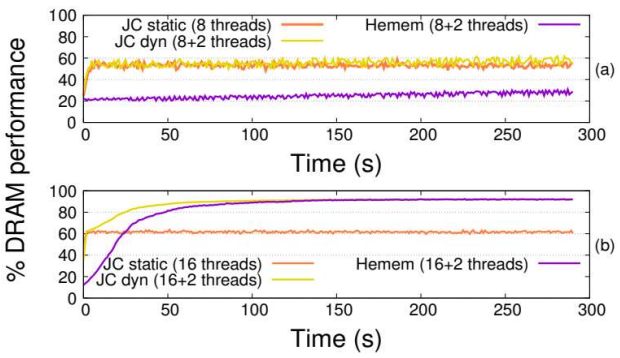
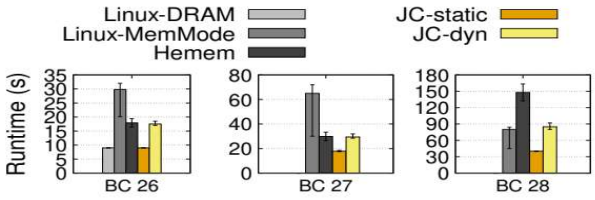


图2 单个大规模数组上的读取性能对比

了BC（Betweenness Centrality）的性能，使用与核心数相同的线程数，分别测试了默认的Linux页面分配器、HeMem和JC。图3展示了介数中心性算法迭代的平均持续时间。 JC-dyn在BC 28上的性能优于HeMem，因为它在PMEM上创建了较少的争用：JC-dyn只偶尔迁移数据（平均每次运行4GB）。它的开销主要来自监测内存访问和页面



图三 BC迭代的平均耗时

冷却。JC-static表现良好，因为它能够在分配时避免大多数冲突。JC-static分配了经常访问的数组的页面，使它们不会相互冲突，有效地减小冲突，而无需进行任何迁移。热数据一次性分配可能看起来有些“幸运”，导致JC-static将所有热页面放入不同的缓存箱中，但事实上这种模式在高性能计算应用中非常常见。

实验结果表明，硬件缓存能够更快地达到稳态性能，并在处理散布的小热项时通常远远优于软件迁移。在对称多处理（SMP）应用程序Betweenness Centrality（BC）的性能评估中，JC-static在多个配置下优于HeMem和JC-dyn，并在一些情况下与手动分配所有数据到DRAM的性能相匹配。硬件缓存的性能优势归因于其在分配时能够避免大多数冲突，有效减少了等待时间，而软件迁移的监测线程与BC的执行产生干扰，导致性能下降。评估结果指出，在有低CPU开销需求的情况下，硬件缓存相对于软件迁移具有明显的优势。

4 总结与展望

在新型存储硬件的背景下，分层内存系统的设计成为提高整个计算机系统效率的关键因素。总结来看，分层内存系统将内存系统深化层次设计，引入更多更高速和更大容量的存储器，并采用更智能的层次划分和数据管理算法。在数据迁移技术路线中，智能缓存管理将更注重借助轻量级的机器学习技术，可能会更准确地预测数据访问模式，并灵活地调整缓存策略，提高命中率和整体系统性能。此外，分层内存系统将更深入地针对特定应用领域进行优化，已有一些在键值数据库中的工作[5]，未来或许可以在人工智能领域设计专门针对深度学习模型的存储层次结构，以提高训练和推理的效率。

分层内存系统将成为推动计算系统性能提升的核心技术。它可能整合更多创新的存储介质，优化能效以实现绿色计算，并更好地支持异构计算和跨设备协同，带来更高效、更智能的内存管理。

参 考 文 献

[1] Baptiste Lepers , Willy Zwaenepoel. Johnny Cache: the End of DRAM Cache Conflicts (in Tiered Main Memory Systems). OSDI, 2023, 519-534

[2] Amanda Raybuck, Tim Stamler, Wei Zhang, Mattan Erez, Simon Peter. Hemem: Scalable tiered memory management for big data applications and real nvm. SOSP, 2021, 392-407

[3] Shai Bergman, Niklas Cassel, Matias Bjørling, Mark Silberstein. ZNSwap: un-Block your Swap. ATC, 2022, 12:1-12:25

[4] Shao-Peng Yang, Minjae Kim, Sanghyun Nam, Juhyung Park, Jin-Yong Choi, et al. Overcoming the Memory Wall with CXL-Enabled SSDs. ATC, 2023, 601-617

[5] Kefei Wang, Feng Chen. Catalyst: Optimizing Cache Management for Large In-memory Key-value Systems. VLDB, 2023, 4339-4352

[6] Jonathan Corbet. AutoNUMA: the other approach to NUMA scheduling. ”https://lwn.net/Articles/488709/”, 2019

[7] Intel. How Does the DRAM Caching Work in Memory Mode Using Intel® Optane™ Persistent Memory ”https://www.intel.com/content/www/us/en/support/articles/000055901/ memory-and-storage/intel-optane-persistentmemory.html”, 2021.

[8] Many contributors. Samsung Electronics Introduces Industry’s First 512GB CXL Memory Module. ”https://news.samsung.com/global/samsungelectronics-introduces-industrys-first512gb-cxl-memory-module”, 2022.

附录1: 汇报记录

问题**1**：在**Linux Page Cache**中采用什么映射策略？

Linux Page Cache采用全相联映射。有三种常见的基本映射方式：直接映射，全相联映射，组相连映射。论文原文中提到了生日悖论，这个例子能够表达作者的设计理念但并不完全贴切，生日悖论描述的碰撞概率是直接映射下的碰撞概率。而Linux Page Cache采用全相联映射，并增加了一系列的设计以强化Linux内存系统性能，例如伙伴策略等等，实际的内存系统并没有遇到生日悖论所阐述的严重碰撞问题。

问题**2**：分层（**Tiered**）和层次化（**Hierarchical**）的区别？

这两个术语在不同的上下文中可能有一些重叠，但它们通常用来表示不同的概念。

1. 分层（Tiered）：- ”分层”通常表示将系统或组织划分为不同的层次，每个层次有不同的角色、功能或性质。- 在分层内存系统（Tiered Main Memory Systems）中，如前面所述，不同层次的内存（如缓存、主内存、辅助存储器）形成了分层结构，每一层次都有不同的性能和容量特征，以满足不同类型的数据访问需求。

2. 层次化（Hierarchical）：- ”层次化”通常表示在一个垂直结构中，将元素按照不同的层次或级别组织起来，形成一种层次结构。- 在计算机科学中，”层次化”可以指代许多结构，例如树状结构、网络层次结构等。在树状结构中，根据父子关系将元素组织成层次结构，具有明显的上下级关系。总的来说，”分层”更侧重于将整个系统或组织划分为不同的层次，每个层次有不同的功能和特性。而”层次化”更侧重于描述一个垂直结构中元素之间的层次关系，有明确的上下级关系，强调层次结构的组织方式。