基于新挑战下的创新性内存管理策略综述

张慧琴[1)]

[1)](华中科技大学计算机科学与技术学院 武汉市 中国 430074)

摘 要 本综述涵盖了五种创新性的内存分配策略，包括基于马尔科夫链的 Buddy 动态分配优化、XPage 大页面内存管理、NBBS 多核非阻塞伙伴系统、Spring Buddy 高并发云计算内存管理以及 LLFree。这些策略在不同层面上对传统的 Buddy 系统进行了创新与发展。其中，基于马尔科夫链的 Buddy 动态分配优化通过预测内存需求，提高了内存分配的效率；XPage 大页面内存管理通过引入 OAT、FM 和 HPM 的设计，实现了对巨大页面内存的高效管理；NBBS 多核非阻塞伙伴系统在伙伴系统基础上引入了非阻塞特性，适用于多核并发环境；Spring Buddy 高并发云计算内存管理通过弹簧核心层和弹簧懒惰层的协同作用，实现了对系统并发性和资源聚合能力的动态调整；LLFree 通过分层机制等策略，为多核系统提供了高效、可扩展的内存管理解决方案。展望未来，我们综述的这五种内存分配策略，为应对内存管理新技术挑战下的策略方法引入了新的思想和技术，具有很大的发展意义。

关键词 伙伴系统；马尔科夫；巨大页面管理；非阻塞伙伴系统；自适应弹性内存管理；可扩展可选的持久性页帧分配

A Review of Innovative Memory Management Strategies Based on New Challenges

HuiQin Zhang[1)]

[1)](Huazhong University of Science and Technology School of Computer Science and Technology，Wuhan China 430000)

Abstract This review covers five innovative memory allocation strategies, including Markov chain-based Buddy dynamic allocation optimization, XPage large page memory management, NBBS multicore non-blocking buddy system, Spring Buddy high-concurrency cloud memory management, and LLFree.These strategies innovate and evolve the traditional Buddy system at different levels. Among them, Buddy dynamic allocation optimization based on Markov chain improves the efficiency of memory allocation by predicting the memory demand; XPage large page memory management achieves efficient management of huge page memory by introducing the design of OAT, FM, and HPM; and NBBS multicore non-blocking Buddy system introduces non-blocking features on the basis of the Buddy system, which is suitable for multicore concurrent environment; Spring Buddy high concurrency cloud computing memory management realizes dynamic adjustment of system concurrency and resource aggregation capability through the synergy of spring core layer and spring lazy layer; LLFree provides efficient and scalable memory management solutions for multi-core systems through layering mechanism and other strategies. Looking ahead, these five memory allocation strategies we have synthesized introduce new ideas and techniques that are of great developmental significance for strategy approaches to address the challenges of new technologies in memory management..

Key words Partner systems; Markov; Huge page management; Non-blocking partner systems; Adaptive elastic memory management; Scalable optional persistent page frame allocation

1 引言

随着计算机体系结构的迅猛发展和应用场景的日益复杂，内存管理作为计算机系统中至关重要的一环，一直以来都备受学术和工业界关注。内存分配策略作为内存管理的核心之一，直接影响着系统的性能、效率和稳定性。然而，传统的 Buddy 系统在面对多核、高并发等复杂应用场景时显露出一些不足，如内存碎片化、性能瓶颈等。为了解决这些挑战，近年来涌现出一系列创新性的内存分配策略，本综述将重点关注其中五种具有代表性的策略进行介绍与分析，包括基于马尔科夫链的 Buddy 动态分配优化、XPage 大页面内存管理、NBBS 多核非阻塞伙伴系统、Spring Buddy 高并发云计算内存管理以及 LLFree，对这些策略进行对比分析。

本综述的目的在于深入剖析这些新兴内存分配策略的设计原理、实现机制以及性能特点，旨在为学术界和工业界提供对于内存管理领域的全面了解。通过对这些策略的对比分析，我们将探讨它们的创新之处，并为系统设计者提供更多选择的建议。通过本文的详细讨论，我们期望为内存管理领域的研究者提供深入的洞察，并为未来内存管理领域的发展方向提供一些建议。在这个快速发展的计算机科学领域，对于内存管理策略的深入研究将为系统性能的提升和创新性应用的推动提供有力支持。

2 研究背景与挑战

2.1 物理内存分配系统

我们仅研究 Linux 的物理内存分配形式，Linux 内核将物理内存管理对象分为节点、区和页框三个层次。早期的内核仅支持单处理器系统，而现在的 Linux 内核则可以在包括多处理器系统在内的各类体系结构的计算机上运行。为了适应 NUMA 体系结构中处理器拥有各自本地内存节点(即分布式内存)的情况，内核采用节点描述符存储内存节点的相关信息。每个物理内存节点对应一个节点描述符，包含相应内存节点的标识符、起始页框号、页框数等字段。

依据内存节点寻址特点及用途的不同可将每个内存节点进一步划分为若干个内存区。内核为每个内存区分配一个区描述符，其中包含内存区的名称、起始页框号、页框数等字段。物理内存是一个线性地址空间，所有内存节点都是统一编址的。页框是物理内存管理的基本单位，如果以页框作为元素，整个物理内存就可以视为一个页框数组，而物理内存管理的主要工作就是从这个数组中分配和回收页框。内核为每个页框分配一个页描述符，用于记录对应页框的信息。节点描述符、区描述符和页描述符构成物理内存管理的基本数据结构，这些结构被相互关联并组织在一起。

2.2 Linux伙伴系统

伙伴算法是一种动态存储分配算法 ，用于实现操作系统内核空间和用户空间的分配和回收操作。最早的伙伴算法是二分伙伴算法，之后，先后提出斐波那契伙伴算法、加权伙伴算法、泛化伙伴算法等诸多变体。实现伙伴算法的内存管理模块便称为伙伴系统，它是固定分区和可变分区的一个合理折中方案。在现代操作系统内核中单纯的伙伴系统并没有得到广泛应用，而分页机制则成为内存管理的 主流技术。尽管如此，Linux 内核成功地将分页机制与伙伴系统系统结合起来，分页机制将逻辑地址空间映射到物理地址空间，伙伴系统负责在物理地址空间中分配和回收页框。为了追求时间效率，Linux 内核选择实现了二分伙伴算法，该算法的优点在于伙伴地址的计算更加简便、高效。

2.2.1 设计思想

伙伴系统从指定的内存区中分配页框，使用完毕后页框被回收到其所属的内存区。为了满足不同尺寸的需要，内存区中的页框被组合成一些内存块，每个内存块包含2^k个连续的页框，其中 k 称为内存块的阶，阶为 k 的内存块称为 k 阶内存块。数的上限，默认为 11，即阶的取值范围是 0~10，因此内存块的尺寸最小为 4K，最大为 4M。对于一个内存块，它的第一个页框称为首页，其余页框称为尾页。一个内存块的第一个页框的描述符称为首页描述符。内存区的第一个页框在区内的相对索引称为该内存块 的首页索引，一个 k 阶内存块的首页索引必须被2^k整除。

2.2.2 分配算法

当内核请求分配内存时，伙伴系统执行分配算法以满足其需求。分配算法的基本思想是寻找能够满足内核需求的最小空闲内存块，如果该内存块的阶大于内核请求的阶，则将其逐步划分为一系列低

阶内存块,直到划分出恰好满足需求的一个内存块,并分配给内核使用。其余低阶内存块按其所属的阶被依次插入相应的空闲链表,用于满足今后的内存请求。

2.2.3 回收算法

伙伴系统回收算法的基本思想是首先确定待回收内存块的伙伴,如果伙伴是空闲的,则将二者合并为一个高阶空闲内存块。重复上述合并过程,直至伙伴不再空闲或合并形成的空闲内存块达到最高阶。在上述过程中,每次合并前都要将伙伴从其所在的空闲链表中移除。合并完成后,最终形成的高阶空闲内存块被插入相应空闲链表的表头。[1]

2.3 技术挑战

在传统的单核单 CPU 内存管理范式中，目前的内存分配方式已经表现出良好的效果。然而，随着新一代硬件趋势，包括多核处理器和非易失性内存（NVRAM）的普及，传统的帧分配器设计面临着全新的挑战。

在巨大页碎片问题上，传统帧分配器可能导致内存利用的低效，也无法满足对持久性的新需求。因此，当前的页帧分配器存在多核性能问题、状态分散、全局锁的争夺、巨大页碎片和无法适应新型硬件趋势等缺陷。

在多核性能方面，传统的 Linux 帧分配器显现出较差的扩展性。分散的分配器状态也导致了性能的下降，引发了对全局锁的频繁竞争，在适应具有崩溃容忍性的 NVRAM 方面也存在着很大的问题。

在高速云计算场景下，每台服务器通常承载多个服务进程，这加剧了系统的并发压力。结果，页面分配和释放期间的内管理过程成为一个严重的瓶颈。传统的伙伴系统和反向伙伴系统等方法虽然可以用来来提高内存管理的性能，但是它们不能适应云计算的高并发环境。

这些问题迫使我们对传统 Buddy 内存分配方式的优化、新型页帧分配器的设计进行深入研究，以满足当今不断发展的硬件环境和需求。

3 研究现状与优化

针对出现的这些新挑战、新难题，纵观近三年的相关论文，在本综述中对如下五个内存管理策略进行详细介绍。

3.1 基于马尔科夫的Buddy动态分配优化

在操作系统的内存分配中，如果自由内存块的链表不包含满足当前需求的内存块，则系统需要申请更大的自由内存块链表或启用内存回收机制，这消耗时间并影响内存分配效率，尤其是在大内存分配中更是如此。该论文作者提出了一种基于马尔科夫链的动态内存分配算法，通过分析经典的伙伴算法和马尔科夫链预测原理，实现了内存分配的动态预测，以提高内存分配的效率。算法通过马尔科夫链的状态转移矩阵动态更新，实现了对系统内存动态分配的预测。

关于马尔科夫链的具体实现细节在本综述论文中不再详细介绍。

3.1.1 马尔科夫动态内存预测

为了更准确地预测下一个内存块的大小，使用状态转移矩阵来记录内存块大小之间的转移概率，通过历史分配数据动态更新状态转移矩阵。

具体而言，在系统启动时，初始化状态转移矩阵和预测链表等数据结构。在内存分配时，根据当前内存分配状态，选择使用传统伙伴算法或预测链表中的内存块。在内存释放时，判断释放的内存块是否与预测相符，如果相符则放入预测链表。

对于马尔科夫状态转移的操作如下：

状态转移矩阵更新：在高阶内存分配状态下，根据历史分配数据动态更新状态转移矩阵。

预测链表管理：根据内存分配和释放结果，维护预测链表中的内存块。[2]

3.1.2 小结

总的来说，该文章提出的基于马尔科夫链的动态内存分配算法，在实验中取得了良好的效果，是将 Buddy 与机器学习算法有效结合的一种策略。

3.2 Xpage大页面内存管理

这项研究提出了一种针对 linux 大页面内存的一种全新的内存分配方式，旨在通过巨大页面的直接使用，避免或减少内存碎片问题，并在不分割巨大页面的情况下支持动态的巨大页面内存管理，从而显著降低大数据应用性能的退化。相较于 linux 中标准的大页面内存管理方式 THP、Ingens 等具有更大的优势。

3.2.1 XPage 设计核心

如图 1，XPage 由三个核心组件组成：对象地址表（OAT）、碎片监视器（FM）和大页管理器（HPM），它们与三个子管理器（缓存管理器 TM、非完全运行

管理器 NRM 和完全运行管理器 FRM）进行交互。

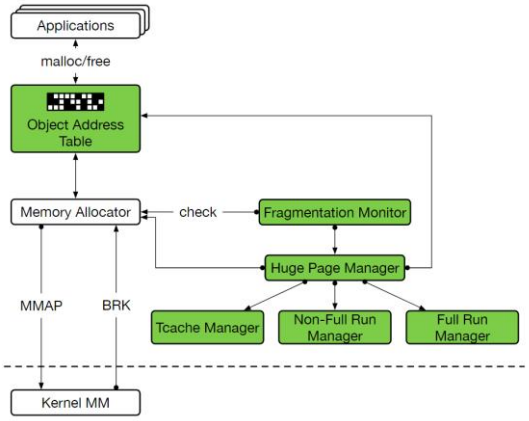

图 1 Xpage 核心组件

OAT 实现了内存分配器在应用程序释放内存之前自由和透明地移动已分配的内存块的机制。它采用间接内存指针，返回给应用程序一个指向 OAT 中条目的间接指针，通过该指针可以解析到实际存储对象数据的真实地址。

FM 记录已使用的内存区域数和已分配的内存块数，通过计算未使用内存区域的大小与所有内存块的大小的比值，来评估内部碎片情况。当内部碎片超过设定的阈值时，FM 与 HPM 协作，触发内存压缩。

HPM 负责拦截所有内存分配和释放请求。内存空间以层次结构进行管理，其中一个巨大页面（2MB）作为内存块的基本单元，被分割成多个运行（run）。每个运行包含一个或多个连续的页面，用于保存特定大小的对象。为了提高运行的管理效率，XPage 创建了两个配对堆（PH），分别用于跟踪完全利用的运行和部分利用的运行，以最小化未充分利用的内存块数量。

不同于在 linux 内核空间运行的 THP 和 Ingens，XPage 在设计时，实现了内存分配器与应用程序（malloc/free）和内核（brk/mmap）进行交互，对内存布局有全面的了解，可以更好的处理内部碎片问题，而这些问题是 THP 和 Ingens 无法做到的。此外，Xpage 还尽可能的确保了整个分配器的公平性和安全性，降低了内核线程的管理开销。

3.2.2 内存分配与释放过程

XPage 的内存分配过程中，HPM 拦截 malloc()调用，调用 arena malloc 获取请求大小加上 8 字节的空闲内存区域，并在 OAT 中查找空闲条目，将该条目的地址存储在请求内存区域的前 8 字节中，最后返回 OAT 中的地址给应用程序。

内存释放时，HPM 拦截 free()调用，调用 arena demalloc 释放指定指针 p 的内存区域，首先获取存储在头部前 8 字节的 OAT 条目地址，释放 OAT 中的条目，并释放整个头部和数据区域至运行。若当前运行为空，释放该运行；若当前内存块为空，释放该内存块。

3.2.3 内存压缩

当内部碎片超过阈值时，FM 与 HPM 协作，触发内存压缩。压缩过程分为 tcache、non-fullrun 和 fullrun 级别。Tcache 压缩通过创建在最低内存空间中创建空运行，将 Tcache 槽指针迁移到新创建的内存区域，之后释放未使用的运行。Non-fullrun 压缩通过导出所有非满运行的地址，运行两个独立的算法，从 compaction 数组的两端开始搜索可以用于迁移的自由内存区域，直到两算法相遇，完成非满运行的压缩。Fullrun 压缩旨在将所有数据从高地址打包到低地址，通过导出所有满运行的地址，创建在最低内存空间中创建的空运行，从压缩数组的末尾向左扫描，如果指针地址高于新创建的运行的地址，则迁移运行指向的数据，否则表示所有运行都已充分打包，压缩完成。[3]

3.2.4 小结

总体而言，XPage 通过引入 OAT、FM 和 HPM 的设计，实现了巨大页面内存管理的动态、灵活和高效性。实验结果验证了 XPage 在动态大数据工作负载下最小化内存碎片的能力。这一内存管理框架为处理大规模数据应用中的内存管理问题提供了一种创新的解决方案。

3.3 NBBS多核非阻塞伙伴系统

该研究是一种基于伙伴系统的内存管理方案，旨在于提供对内存的高效管理，允许在多线程环境中进行并发的内存分配和释放操作。它在这个 Buddy 的基础上引入了非阻塞特性，使其适用于多核并发环境。

3.3.1 NBBS 设计核心

NBBS(Non-Blocking Buddy System)使用一棵完全二叉树来组织内存块，其中每个节点代表一个内存块。树的深度由系统预定义，通常为一个较小的值。每个节点包含一个位图，用于表示节点自身的状态以及其子树的状态。

论文中提出了一种基于 Read-Modify-Write（RMW）指令的非阻塞方法，避免了传统方法的自旋锁协调机制。NBBS 通过 RMW 指令在分配和释放操

作的关键路径上检测并发冲突，实现了非阻塞协调，从而提高了并发访问的效率。

3.3.2　NBBS 内存分配算法

NBBS 内存分配过程中，首先确定目标级别，然后搜索可用节点，得到可用节点后尝试对可用节点进行获取，获取后再进行缓存更新。

具体而言，会根据内存请求的大小，使用二进制规则确定目标级别(即包含足够内存的节点所在的级别)，在目标级别的节点中进行搜索；首先尝试使用线程的“软件缓存”中的缓存引用，如果缓存引用失败，执行线性搜索以找到一个可用的节点；使用 RMW 指令，尝试获取找到的节点，通过原子操作来尝试获取找到的节点，如果成功，则更新节点及其祖先节点的状态（标记为占用和部分占用），如果节点或其祖先节点已满，则尝试分配失败，重新搜索可用节点；如果分配成功，更新“软件缓存”中的缓存信息，并指向同一级别的下一个节点。在缓存更新时，会涉及到 RMW 指令的使用，以确保多个线程同时更新缓存时不会发生冲突。

3.3.3　NBBS 内存释放算法

内存释放过程中，首先标记祖先节点，根据内存地址找到对应的节点索引，然后调用 FREENODE 过程，将节点的祖先节点标记为正在合并状态；再进行释放节点操作，将节点标记为空闲状态，表示该内存块现在可用于新的分配；最后更新合并状态，更新所有之前标记为正在合并的节点，以通知其子树已经释放，可以再次用于内存请求。

3.3.4　NBBS 优化策略

软件缓存：为每个线程维护一个指向每个级别的最后释放节点的指针，通过线程的“软件缓存”来提高节点的搜索效率。

4 级封装：将多个节点的状态封装到一个 64 位字中，通过使用 CAS 操作确保原子性。这减少了关键路径上的 RMW 操作数量，提高了性能。

3.3.5　小结

总的来说，NBBS 通过引入非阻塞特性和一些优化策略，使得在多线程环境中能够以高效和并发的方式管理内存。在传统伙伴系统的基础上，通过位图表示节点状态，使用 CAS 指令确保状态更新的原子性，同时通过软件缓存和 4 级封装优化提高内存分配和释放的性能。

3.4　Spring Buddy高并发云计算内存管理

该文章旨在于缓解在高并发云计算环境下存在系统并发压力，改善内存分配和释放时的并发性，并避免不必要的碎片。创新性地使用 Spring Core Layer 来提供并发响应和资源聚合能力，使用系统的并发压力，使用 Spring Lazy Layer 对进程行为进行预测，进一步减轻对系统资源的争用。

3.4.1　Spring Buddy 设计核心

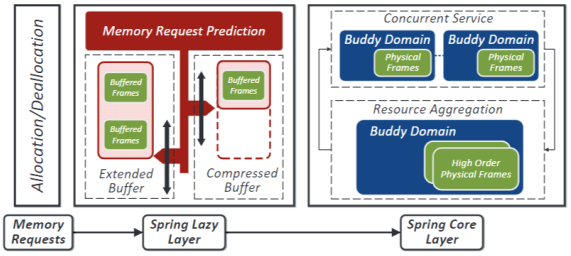

图 2　Spring Buddy 技术核心

如图 2 展示了 Spring Buddy 的整个技术概览，包括两个主要层次，即弹簧核心层和弹簧懒惰层。

弹簧核心层：Spring Buddy 核心层实现了系统对全局并发压力的动态适应。在高并发场景下，弹簧核心层会扩展以满足来自不同 CPU 核心的请求，而在低并发场景下则会进行压缩，以适当地聚合资源并减少碎片。

弹簧懒惰层：Spring Buddy 使用动态调整的懒惰机制进行请求缓冲。弹簧懒惰层根据对潜在内存请求的预测动态调整每个 CPU 核心的缓冲区大小。因此，它提高了缓冲区命中率，同时减少了资源的浪费。

这两个层次的协同作用使得 Spring Buddy 在不同的并发情景下都能够有效地管理内存，提高资源利用率，降低碎片化，从而优化了内存管理性能。

3.4.2　弹簧核心层

弹簧核心层管理全局的内存资源池，并通过弹簧域（buddy domains）的动态划分来处理并发请求，以提高系统的并发性。

在内存分配与释放过程中，在系统通常情况下，经历确定目标级、搜索可用节点、获取节点、缓存更新、内存释放、更新合并状态。

弹簧扩展：当系统面临高并发压力时，弹簧核心层会动态扩展，将内存划分为多个弹簧域，以实现高并发内存管理。

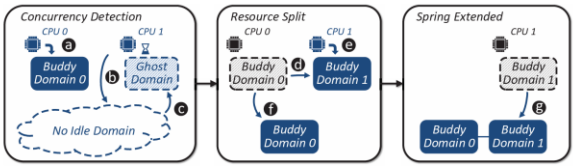

图 3 Spring 扩展

具体而言，如图 3，当检测系统中的高并发请求时，初始只有一个弹簧域，被 CPU 0 使用。当 CPU 1 发起内存请求时，发现没有可用的弹簧域，因此创建一个虚拟的 “ghost”弹簧域，并强制 CPU 1 等待。同时，CPU 0 完成请求后，将部分页面传递给 ghost 弹簧域，激活 ghost 弹簧域成为新的弹簧域，服务 CPU 1。当 CPU 1 完成请求后，将弹簧域 1 放回全局弹簧，此时弹簧的数量增加，实现了动态扩展的过程。

这种动态扩展的机制可以根据系统的需求自适应地调整，有效地提高了内存管理的性能。

弹簧压缩：当系统进入低并发请求阶段时，为了避免碎片化，弹簧核心层会将多个弹簧域合并为一个，扩大资源池的管理边界，实现碎片的整合。

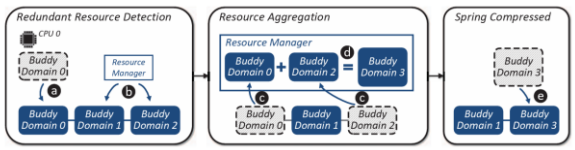

图 4 Spring 压缩

具体而言，如图 4，当一个弹簧域 0 完成服务并被放回到全局弹簧时，触发探测操作。系统检测到具有相同类型且空闲时间较长的两个冗余弹簧域，选择两个适当的冗余弹簧域 0 和 3 进行合并，资源管理器选择这两个弹簧域，并执行合并操作。在合并的过程中，从两个独立的、小碎片化的弹簧域中创建一个更大的弹簧域 3，其中包含更多高阶的空闲页面帧。合并后的弹簧域被视为一个新的弹簧域，其边界被重新调整。此时，弹簧核心层进行了实质性压缩，多个小的碎片被整合成一个大的碎片，减少了内存碎片化。合并后的新弹簧域被放回到全局弹簧，使其成为系统管理的一部分，同时保持整体的并发性。

这种动态压缩机制有助于系统在不同并发负载下自适应地管理内存资源，提高了内存管理的效率。

3.4.3 弹簧懒惰层

弹簧懒惰层根据对潜在内存请求的预测动态调整每个 CPU 核心的缓冲区大小。

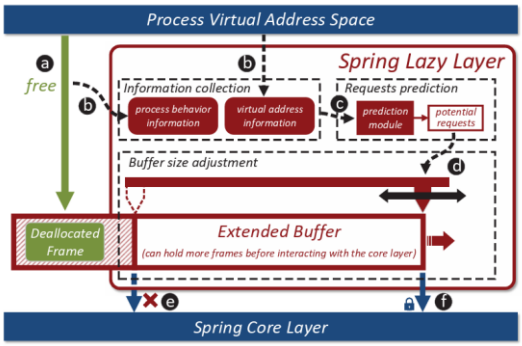

图 5 Spring Lazy Layer 核心

具体而言，如图 5，弹簧懒惰层的实现过程涉及内存请求模式分析、懒惰缓冲区管理、内存请求预测和懒惰扩展四个过程。首先，弹簧懒惰层根据进程的内存请求模式，识别连续内存请求的模式，从而决定是否进行懒惰缓冲。当处理内存释放请求时，将页面帧放入懒惰缓冲区，并根据懒惰缓冲区的限制动态调整缓冲区大小。然后，收集有关进程内存请求行为和虚拟地址空间映射的信息，预测未来可能发生的内存请求。最后，根据这一预测，调整缓冲区边界以容纳更多页面。在懒惰缓冲区预测到一批请求即将结束时，将缓冲区中的页面释放到全局资源池，这样可以避免在缓冲期间对全局资源的争用

通过以上步骤，弹簧懒惰层可以根据进程的内存请求模式和当前系统状态，智能地管理懒惰缓冲区，提前预测并适时扩展，以提高内存管理的效率和性能。这种动态调整的机制使得系统能够更好地适应不同负载和请求模式。[4]

3.4.4 小结

总体而言，Spring Buddy 通过弹簧核心层和弹簧懒惰层的协同作用，实现了对系统并发性和资源聚合能力的动态调整。在实现的过程中，弹簧核心层负责全局资源的分割和合并，而弹簧懒惰层则通过对内存请求模式的预测，提高了对内存请求的聚合效率。该机制为云计算环境中的内存管理提供了一种高效且灵活的解决方案，特别适用于多服务进程并发运行的场景。

3.5 LLFree

LLFREE，是一种比较先进的内存分配器，该论文通过深入研究 Linux 伙伴系统分配器存在的主要缺陷，致力于在多核系统中提供高效、可扩展的内存管理解决方案。其主要设计原则包括对硬件的充分尊重、避免共享和谨慎处理冗余。采用了分层机制，引入了“Reserve-on-Free”启发式方法、基

于分配和释放的局部性假设等一系列措施，优化了对 CPU 首选树的选择策略。

3.5.1　LLFree 设计原则

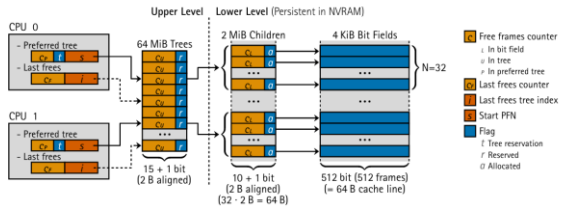

图 6 LLFree 核心

如图 6，在整个 LLFREE 的实现过程中，采用了分层机制，分为低层（Low-Level）和高层（Upper Level）：

底层使用表项和位字段进行管理负责处理普通页框和大页框的分配与释放，实现了对普通页框和大页框的高效管理，同时解决了崩溃一致性问题。高层通过构建性的分配策略实现了可扩展性，将物理内存管理为一个树状结构，采用树根来管理子级，以避免大页框的碎片化，并解决虚拟共享问题。

3.5.2　LLFree 低层设计

低层内存分配器的主要职责是处理普通页框（base frames）和大页框（huge frames）的分配和释放操作，同时限制崩溃一致性问题。

为了实现这一目标，底层内存分配器采用了一个表项（table entry）和一个包含 512 位的位字段（bit field）来管理每个大页框。

对于普通页的分配，首先通过原子递减计数器 cL 来保证操作的原子性，其中 cL 的值代表有多少个空闲基本框架。然后，通过特殊的处理器指令，在位字段中搜索一个值为 0 的位，将其分配给用户。

对于大页框的分配，通过将计数器 cL 从 512 更改为 0，表示整个大页框都已经被分配，并设置分配标志 α 来记录必要的簿记。在这种情况下，位字段全部为零。标记位 α 主要是为了满足系统崩溃时，恢复数据而记录一些关键信息，以便在系统崩溃后能够恢复到正确的状态

在该实现策略下，在大多数情况，低层分配器仅需访问两个缓存行（表项、位字段），就可以实现对基本框架的分配和释放。对于大页框，仅需访问一个缓存行（表项）即可。

此外，LLFREE 有在 NVRAM 上提供崩溃一致性的能力，以确保在系统崩溃后，分配器的状态可以被正确地恢复。实现主要是将分配器的底层信息存储在持久内存中，再使用一个额外的存储区域，保存包含一个魔术标识符、内存区域的大小以及一个启动/关闭标记。这用于在系统断电后检测系统是否需要进行恢复。

3.5.3　LLFree 高层设计

LLFree 高层主要是负责解决虚假共享和大页框的碎片化的问题，以提升系统的可扩展性，致力于减少对锁的竞争。

具体实现上，将物理内存构建为一个称为“树”的块数组，形成页表树形结构。每个树根指向一个下层表，下层表又有多个子级，每个子级又指向若干个位字段，用于管理基本帧。。

为了避免虚假共享，建立了 CPU 本地预留机制，为每个 CPU 分配一个固定的首选树。具体分配时，首先在首选树中选取空闲帧分配，只有在首选树中没有剩余的空闲帧时，再选取新的预留树。

为了避免大页框的碎片化，通过保留算法，将页表树分为三类：allocated（表示几乎所有框架都被占用），free（表示几乎所有框架都是空闲的），partial（介于两者之间）。在本地预留树空闲空间不够的时候，首先在当前 CPU 首选树的邻近范围内搜索部分或空闲树，然后按顺序搜索整个树数组，最后在没有找到部分树时，重复搜索以寻找空闲树或有足够空闲页面的已分配树。[5]

3.5.4　小结

总的来说，LLFREE 在解决传统内存分配器存在问题的同时，也为多核系统提供了一种高效、可扩展的内存管理解决方案，具有重要的理论和实际意义。未来的工作还可以在进一步优化性能、拓展应用领域等方面展开。

4　对比结论分析

通过这本综述提到的这些论文的研究与对比，可以发现：

Xpage：提出了一种基于页面的内存管理策略，引入了自适应哈希索引技术，以优化内存管理。这种创新有助于减少内存碎片化，特别适用于大规模数据处理。然而，在并发性和动态适应性方面相对传统 Buddy 分配器可能稍显不足。

NBBS：着重优化了多线程环境下的内存管理，通过引入锁粒度的概念，提高了内存分配和释放操作的并发性。相较于传统 Buddy 分配器，NBBS 在多线程密集型应用场景下具备一定优势，具有一定的创新性。

Spring Buddy：引入了弹簧核心层和弹簧懒惰层，以应对高并发云计算环境。其动态扩展和压缩机制，以及懒惰层的内存请求预测调整，使其在处理并发和资源聚合上表现出色。在高并发云计算环境下具有显著创新，但在某些特殊场景可能产生额外的内存开销。

LLFree：提供了多核系统上的高效可扩展内存管理解决方案。通过分层机制、避免虚假共享和处理大页框碎片化，以及提供崩溃一致性，使其在多核系统下具备良好性能。不过，对硬件的要求相对较高，需要权衡硬件支持和性能优势。

马尔科夫动态内存预测：提出了一种基于状态转移的内存管理策略，利用马尔科夫决策过程对内存分配进行建模。该方法通过分析进程的内存请求模式，动态地调整内存管理策略，以提高内存分配的效率。这种创新的思路使得系统能够更好地适应不同负载和请求模式，但在实际应用中可能需要更多的计算资源用于状态转移和决策过程，可能对系统性能产生一定的影响。

总体而言，在应对如今面对的新技术挑战，这些内存分配策略都具有很大的发展潜能，马尔科夫内存管理在动态适应性方面表现出色；XPage 有效解决了内存碎片化问题；NBBS 为多核环境提供了非阻塞特性；Spring Buddy 增强了高并发云计算场景，LLFree 在多核系统中提供了高效和可扩展的解决方案。这些策略对内存管理创新发展具有很大的指导意义。

5 总结与展望

综合而言，本文讨论的多样化内存管理策略展示了不断演进以解决各种计算场景挑战的努力。这些创新为更高效、更适应性的内存管理解决方案铺平了道路。展望未来，整合机器学习和人工智能可能进一步增强这些策略，使系统能够根据不断变化的工作负载动态学习和优化内存管理。对硬件架构和持久内存技术的探索也为未来在这一领域的进一步发展提供了希望。内存管理领域的持续研究和发展凸显了在计算领域不断演变的背景下确保系统性能和资源利用的关键作用。

参 考 文 献

[1] Xue Feng. Linux Kernel Partner Systems Analysis. Computer software and computer applications,2018,27(01)
(薛峰. Linux内核伙伴系统分析.计算机软件与计算机应用,2018,27(01))

[2] Peng Liang; Lijun Chen.Dynamic memory allocation prediction algorithm based on Markov chain. In 2022 14th International Conference on Measuring Technology and Mechatronics Automation.pages 2157-1473,2022.

[3] Wenqi Cao; Ling LiuS; Gong Su; Arun Iyengar. Efficient Huge Page Management with Xpage. 2021 IEEE International Conference on Big Data (Big Data).IEEE,2021.

[4] Romolo Marotta; Mauro Ianni; Alessandro Pellegrini etc.A Non-Blocking Buddy System for Multi-Core Machines. IEEE Transactions on Computers.pages 599-621,2021.

[5] Yihui Lu; Weidong Liu; Chentao Wu etc. Spring Buddy: A Self-Adaptive Elastic Memory Management Scheme for Efficient Concurrent Allocation/Deallocation in Cloud Computing Systems. 2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS).IEEE,2021.

[6] Lars Wrenger, Florian Rommel, and Alexander Halbuer etc.LLFree: Scalable and Optionally-Persistent Page-Frame Allocation. USENIX ATC '23 Open Access Sponsored by King Abdullah University of Science and Technology (KAUST).2023.

附录.

问题：在多核处理器系统中，内存管理存在一些主要的挑战和弊端
回答：在当前多核处理器环境下，内存管理领域面临着一系列挑战。

其中一项关键问题是内存一致性问题。在多核系统中，各个核心拥有独立的高速缓存，对共享内存的修改可能导致缓存不一致，进而引入复杂的缓存一致性协议。为应对这一问题，一些创新性的内存分配策略，如Spring Buddy，通过动态适应和资源聚合的方式提高系统的并发性，从而有效解决内存一致性挑战。

还有就是多核环境下的并发冲突与锁竞争，对系统整体性能产生负面影响。为最小化并发冲突，提高系统的并发性能，一些内存分配策略，例如LLFree、NBBS等，其中LLFree采用了分层机制来高效管理普通页框和大页框，有效缓解了锁竞争问题。

此外，内存带宽瓶颈也成为多核系统性能的制约因素。一些策略，如LLFree，在内存分配过程中引入了NVRAM，以提供崩溃一致性，从而优化了内存访问效率，减轻了内存带宽压力。

大页框碎片化问题对内存管理提出了新的挑战。通过底层和高层的设计，一些内存分配策略，如LLFree、Xpage等，成功解决了大页框碎片化问题，提供了高效可扩展的内存管理解决方案。

动态负载均衡是多核系统中一个复杂的问题。一些内存分配策略，包括Spring Buddy，通过协同作用的不同层次，实现了对系统并发性和资源聚合能力的动态调整，为动态负载均衡提供了高效且灵活的解决方案。

在未来的研究方向上，可以进一步优化这些策略与问题，使其更好地适应不同应用场景和系统配置，为多核处理器内存管理领域的发展提供更多可能性。