提升数据中心内存密度的研究

易正烨[1)]

[1)](华中科技大学 武汉国家光电研究中心, 武汉市 430074)

摘 要 如今越来越多的服务陆续上云，并且服务规模的越来越大，而与此同时内存价格居高不下，数据中心如何通过成本较低的手段，快速地在有效地资源中尽可能多地部署用户服务是一个关键问题即如何提升内存密度。内存冗余数据删除手段是就广泛使用且效果显著的提升内存密度方式，但与此同时内存冗余删除带来了不可忽略的计算开销和安全性、隔离性风险，我们需要安全高效的内存冗余删除机制来提升数据中心内存密度。本综述分析了基于数据冷热的内存密度优化和基于内存能力指针的数据跨区共享，分析两类方法是如何解决内存密度提升中计算开销和内存开销间的权衡。

关键词 数据中心；内存密度；内存冗余删除；内存虚拟化；内存共享；分区隔离；

The Survey of Increasing Memory Density in Data Centers

Zhengye Yi[1)]

[1)](Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, 430074, Wuhan)

Abstract Nowadays, more and more services are going to the cloud one after another, and the scale of services is getting bigger and bigger, and at the same time, the price of memory is still high, how can data centers deploy as many user services as possible in the effective resources by means of less costly means is a key issue, that is, how to improve the memory density. Memory redundancy deletion is a widely used and effective way to improve memory density, but at the same time, memory redundancy deletion brings non-negligible computational overhead and security and isolation risks, we need a safe and efficient memory redundancy deletion mechanism to improve the memory density of data centers. This review analyzes memory density optimization based on data hot/cold and data cross-region sharing based on memory capacity pointers, and analyzes how the two types of approaches address the trade-off between computation overhead and memory overhead in memory density improvement.

Key words Datacenter；Memory Density；Memory Dedupliacation；Memory Virtualization；Memory Share；Compartmentalization Isolation；

1 引言

最近几年 DRAM 技术没有出现突破性进展，导致每 GB 的 DRAM 内存单价居高不下，这让云服务提供商需要更高的成本来通过扩展数据中心内存规模，以满足日益增长的业务需求。如何在数据中心有限的内存中尽可能多地部署用户应用和服务，也就是如何提高数据中心内存密度(Memory Density)变得日趋严峻。可以看出，内存密度是关键的成本控制因素之一，提高内存密度可以显著降低云服务商的总拥有成本。

提升数据中心内存密度的方法可以从两个视角入手：(1)从内存本身(操作系统层面)来说，可以通过使用内存冗余删除(Memory Deduplication)机制实现内存空间的节省；(2)从内存外部(系统结构层面)来说，可以通过利用新型计算设备、新型器件来缓解内存压力，提高内存服务部署密度[2]。本综述将结合最近三年的最新研究进展，分析从内存本身即操作系统层面来提升数据中心内存密度方法。

内存冗余数据删除在现代计算机操作系统中应用广泛，其中典型的实现就是 Kernel same-page Merge(KSM)算法，KSM 通过定期扫描物理内存以找到相同的页面，并通过将单个物理副本映射到多个虚拟位置来实现节省物理内存空间。但是传 KSM 算法存在几个严重的问题，(1)KSM 的定期扫描策略严重占用了 CPU 资源，对系统中协同运行的其他应用和服务的正常运行造成了风险，存在较高的尾延迟；(2)随着近些年来随着 CPU 性能提升，为了提高 TLB 的命中率，操作系统页面越来越大(例如，x86 架构提供对 2MB 和 1GB 内存页面的支持，ARM 架构支持 1MB、16MB 内存页面)，这使得 CPU 的计算和匹配开销更大，并且内存中出现完全相同页面的概率会随着页面的增大而越来越小，以页面为粒度的内存重复数据很难达到缩减内存冗余数据的目的，得不偿失 KSM 算法几乎失效；(3)此外，由于 KSM 通过将单个物理页面映射到多个虚拟位置来实现重复页面删除，并且使用写时复制(copy on write,COW)机制在共享页面需要执行写操作时触发复制操作，虽然一定程度上提高了去重效率，但是该机制容易遭受侧信道攻击，安全性不佳[1、3、4、6]。

数据中心作为集中托管计算资源和存储资源、处理业务的基础设施，为了保证服务之间的隔离性、集群安全性、服务可迁移与可扩展性，多使用了基于虚拟机的或基于容器的虚拟化技术，在这些特性的加持下为数据中心内存密度的提升提出了新的要求。由于虚拟机和容器使用了不同的策略实现虚拟化，所以在数据去重和数据共享上存在较大的差异：(1)容器位于物理机器和主机操作系统之上，容器不虚拟底层计算机硬件，仅只是虚拟化操作系统，所以在单台机器上运行的多个容器实例之间共享宿主机的操作系统，操作系统为容器实例提供共享内存用户空间抽象和映射不同进程间相同二进制对象的加载器，并且共享操作系统可以在加载时通过用户级信息对对象内容去重，没有额外的运行时开销；(2)而每个虚拟机都有自己的客户态操作系统，通过监管程序(Hypervisor)使用指令集架构层面暴露的少数几个虚拟化接口，对底层操作系统虚拟化，监管程序在保证强隔离性和安全性的同时在运行时进行内存冗余数据删除操作。典型的虚拟机监管程序有 VMware ESX、Linux KVM 等，他们的内存重复数据删除机制都是基于 KSM 算法。

无论是容器还是虚拟机他们的与内存重删的机制都存在明显不足，容器虽然能通过操作系统加载器在进程加载时识别重复对象消除了运行时开销，但是由于容器的一系列操作都是基于共享操作系统，操作系统涉及复杂接口众多攻击面大，有较大的潜在安全风险，这对于数据中心这种数据和计算密集型基础设施是无法容忍的；而虚拟机使用的监管程只有少数几个接口，具备较小的计算可信基(trusted computing base)具备更高的安全性，并且虚拟机的虚拟机制也决定了其相较于容器隔离性更强，但是其高度依赖内存扫描，计算开销和尾延迟较高，而且页面级别的去重粒度效果不佳。

基于上述几点，我们可以总结出在数据中心中涉及提升内存密度机制的几个关键技术要求：(1)最小可信基下的强隔离性：(2)较低性能开销与尾延迟；(3)细粒度的内存去重和共享粒度。后文中将梳理近几年在数据中心中提升内存密度领域的研究进展，详细分析他们是如何克服现有技术架构不足，实现上述三点关键要求的。

2 基于数据冷热的内存密度优化

前文中提到同一主机机器上的不同虚拟机通常运行类似的操作系统(OS)或应用程序。在不同的虚拟机之间很可能存在大量冗余数据，但随着现代操作系统的内存页面越来越大，传统的基于内存页面冗余消除算法几乎失效。

为了实现更有效的重复数据删除，当前的操作系统利用了激进的重复数据删除方法(ADA)，它积极地将大页面(如 2MB 页面)拆分为较小的基本页面(例如，4KB 页面)，然后在基本页面之间执行重复数据删除。但是，在拆分之后，TLB 中条目覆盖的内存空间可以显着减少。尽管 ADA 节省了更多的内存空间，但访问拆分的大页面会显着增加最后一层 TLB 未命中率和页表遍历的数量，从而降低内存访问性能。此外，当前操作系统没有很好地支持拆分大页面的重建。在不断运行的系统中，拆分页面越来越多，导致内存访问性能不断下降。随着云数据中心的流行，ADA 逐渐无法适应新环境。

为了最大化删除数据中心内存中的冗余数据，并同时尽可能高的保持数据中心中托管物理服务器上内存访问性能，可以根据内存中数据访问的冷热模式来优化内存冗余删除算法。

2.1 基于页面冷热的页面拆分优化

Yao 等人提出了 SmartSD：定期扫描页面，并

拆分具有高重复率的冷大页面，通过重复数据删除节省内存空间，同时，当拆分的大页面变得热以提高内存访问性能时，重建拆分的大页面[7]。由于定期扫描内存的需要可能会产生额外的 CPU 开销。为了解决这个问题，SmartMD 还支持监控 TLB 未命中的成本即每个 VM 和主机系统的页表遍历百分比，这些信息用于动态调整页扫描周期，以平衡虚拟机的性能和内存节省。它还使用按需策略来重建大页面，从而减少 SmartMD 带来的 CPU 开销。

SmartMD 的核心是如何有效地监控页面的重复率和访问频率，以及如何动态在大型页面和基本页面之间进行转换，以实现高重复数据删除率和高内存访问性能，其实现可以概括为以下几个方面：(1)使用计数布隆过滤器引入监视器中，以便它可以以非常低的开销准确地跟踪页面的状态；(2)使用自适应转换方案，该方案选择性地将大页面拆分为基本页面，并根据访问频率和重复率选择性地重建大页面；(3) 基于 TLB 缺失率设计了灵活的调整策略，动态调整页面扫描周期和页面重建时间。

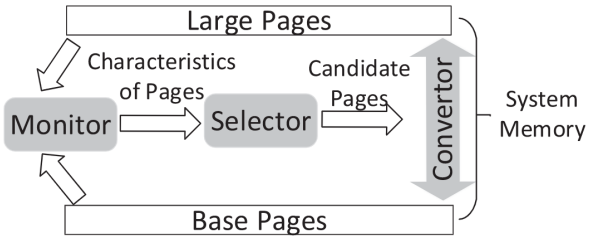

图 1 SmartMD 系统架构

图 1 为 SmartMD 的系统架构，可以分为 Monitor、Selector 和 Converter 三个模块。在 Monitor 中，SmartMD 定期扫描所有大页面和基本页面以记录页面的访问频率、重复子页面的数量和大页面的重复率；Selector 则将使用此信息来选择候选大页面进行分割或候选基础页面进行重建；最后，Converter 根据当前系统内存利用率、数据访问频率和大页面重复率动态选择执行大页面和基本页面之间的转换。SmartMD 基 Linux 3.4 实现，并使用 QEMU 进行实验来管理 KVM，在搭载两颗物理 CPU 的两个物理 NUMA 节点上验证 SmartMD 性能。数据验证集使用了 Graph500、SPECjbb2005 等，系统开销测试实验结果显示 SmartMD 的性能开销远低于 KSM，与略高与不使用任何内存冗余删除机制性能开销，而在内存重删率测试实验中，如图 2 所示，SmartMD 能有效节省内存开销，虽然随着测试时间持续推进 SmartMD 和 KSM 重删率基本一致，但是 SmartMD 能在服务启动初期就开始有效工作。从而可以看出，SmartMD 能在较低性能开销前提下，快速识别内存重复页面并删除。

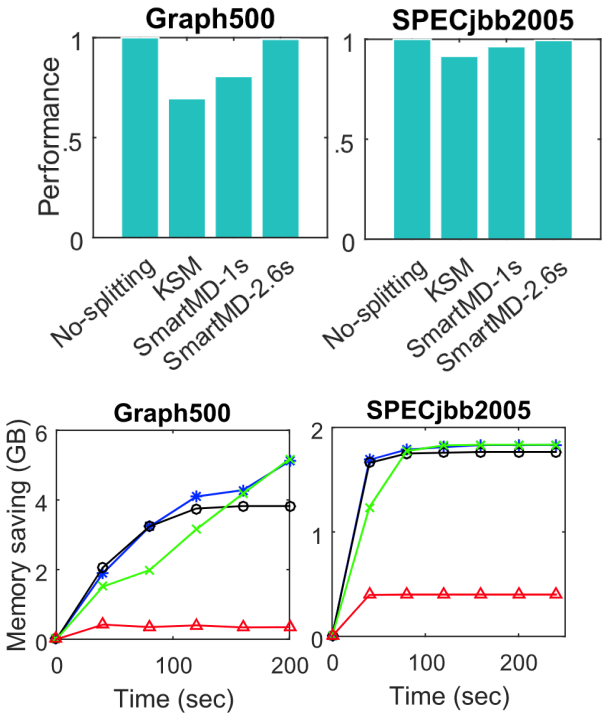

图 2 SmartMD 性能测试

2.2 基于容器冷热的数据块重用

近期无服务器(severless)计算模式炙手可热，因为使应用程序开发人员能够通过将供应、管理和缩放资源的负担转移到云提供商上来专注于应用程序逻辑，以及通过细粒度计费提供更高的成本效益，开发人员仅为功能实际运行时间付费。随着要求更高的应用程序和工作负载迁移到无服务器计算模式的数据中心中，提供商面临的关键问题是如何满足严格的性能要求，同时确保资源效率。

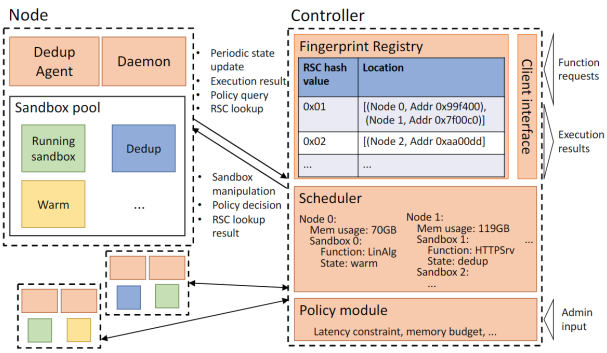

图 3 Medes 系统架构

无服务器模式性能在很大程度上取决于函数实例可以开始作用于最终用户请求的速度，目前无服务器平台通过在容器的冷热两个状态之间来管理的，然而该方法在性能和效率之间引入了高度约束，需要使用大量资源才能实现良好的性能，这使得运营商很难通过控制资源使用来调整数据中心性能。

Divyanshu 提出了一种灵活性更高的管理机制 Medes，在与当今平台相同的效率下实现了更好的性能[6]。Medes 通过引入一种新的容器状态即消除重复状态 Dedup 来改善当前的机制，该状态的内存占用和启动性能介于冷态和热态之间。Dedup 状态是建立在将可重用容器数据块 RSC 之上，RSC 可被其他容器所使用。Medes 实验表明(1)相同功能的容器在其内存由有高达 85%的重复块；(2)不同功能的容器中数据块的重复率在 80-90%。从本质上讲，RSC 的概念是通过在容器中删除这些冗余内存块来改善无服务器平台的内存性能。

在 Dedup 态中，容器内存中所有冗余内存块都被删除，只有在内存中保留唯一父辈。具体来说：Medes 对基础容器重删后转换为 Dedup 态，该基础容器中只存储其他容器中不存在的 RSC，而那些备份删除的数据块存在于其他远程基础容器中；启动/恢复一个新服务时，Medes 使用到存储的本地 RSC 与通过网络从远程基础沙盒读取的 RSC。重复数据消除方法可确保 Dedup 状态的容器内存占用比热的容器小得多，并且 Dedup 状态容器的启动比冷态容器启动快得多。通过重复数据消除更多的沙盒，总体内存使用量比只有热沙盒的平台要小，使得数据中心可以承载更多的服务。此外，还可以利用这些节省的内存来保留更多的沙盒，从而提高性能。

图 3 展示了 Medes 的整体架构它，由一个控制器和多个功能节点组成，通过数据中心网络互连。

控制器有四个主要组件：(1)客户端接口:通过该接口提交功能请求并检索结果；(2)调度器:跟踪系统状态以产生新的容器或分配现有容器来服务传入请求，并决定是否将已完成的容器转换为热态或 Dedup 状态；(3)指纹注册表：一个包含 RSC 的哈希值及其在集群中用于重复数据消除的相应位置的哈希表；(4)策略模块：存储延迟、存储器约束之类的策略参数。

每个节点由守护进程和重复数据消除代理组成：(1)守护进程根据控制器的指令操作本地容器，并更新节点状态；(2)重复数据消除代理，按照控制器指令对本地容器执行重复数据消除，当请求本地容器时将它们从 dedup 态恢复到热态。

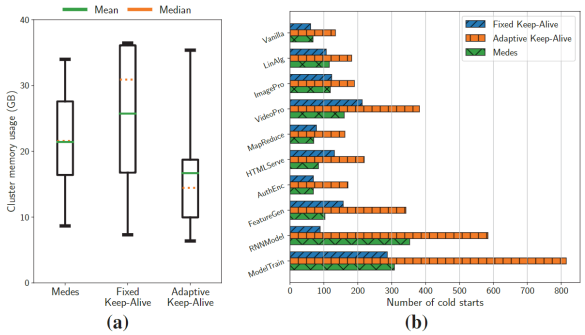

图 4 Medes 性能测试

图 4a 显示，与固定的保活策略相比，Medes 在满足相同的延迟目标前提下内存占用减少了 11.4%；而虽然自适应保活策略的内存占用较小，因为其短暂的保温期会导致内存使用量减少，但是增加冷启动次数，图 4b 所示自适应保活策略的冷启动次数比 Medes 多了 50%。我们进一步观察到，与固定保活策略相比，Medes 可以将冷启动次数提高 1.58 倍，这导致端到端尾延迟提高 1.9 倍，这是因为 Medes 实现的灵活策略允许它对内存占用较大的函数的容器进行重复数据消除，为其他较小的函数腾出更多空间来保持沙盒的热态，从而使这两个函数都达到各自的延迟目标。例如，通过积极地对 RNN 模型的容器进行重复数据消除而不保持热态的 RNN 模型的热态容器，Medes 以冷启动为代价减少了内存使用达到了延迟目标，由此节省的内存也可以用来为其他功能的沙盒保温。总的来说，与固定的保活策略相比，Medes 可以在更小的内存占用中满足延迟目标。

3 基于内存能力指针的跨区数据共享

内存能力(Memory Capability)指针是由硬件提供的一种保护和共享机制。与传统的指针相比，内存能力指针通过扩展到 256 位提供了对应内存空间的访问权限信息、可访问地址空间信息、指针有效性等元数据信息，具备更强的鲁棒性、扩展性和隔离性，此外，内存能力指针可以在页表转换机制中使用。基于内存能力指针的机制，可以在单个内存地址空间中划分出多个相互独立的内存隔离区，也可在多个隔离的区域之间使用单个页表管理。

CHERI(Capability Hardware Enhanced RISC Instructions)架构提供了对内存能力指针的具体实现，通过一系列的新指令、新寄存器来支持内存能力指针设计。通过 CHERI 架构替代应用中的所有指针可以实现对每次内存访问的空间和权限控制，这也就是存能力(pure-cap)模式，Pure-cap 模式可以保证内存安全性和细粒度的内存共享；此外，CHERI 也支持混合模式，在混合模式中允许使用传统指令集架构中的指令，在这种模式下所有控制流和内存操作都会执行程序计数器和默认数据能力的检查，以限制非内存能力指令执行的代码和数据访问。内存能力指针为软件层面的分区隔离提供了一条新的思路，并通过其特性消除基于页面的内存共享的计算开销。

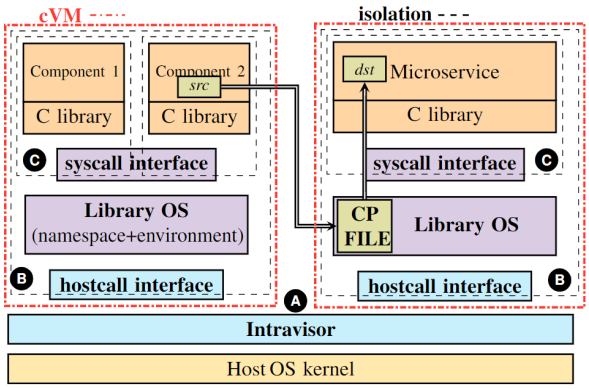

图 5 cVM 系统架构

3.1 分区隔离

Sartakov 等人结合内存能力指针特性和 CHERI 架构的最新研究，提出了 CAP-VMs(cVM)用于云环境中的基于内存能力的隔离和共享机制，cVM 是应用程序组件的虚拟化和分区抽象[4]。这些组件通常可以共存和交换数据，cVM 将它们与使用 CHERI 功能的低开销数据交换的支持隔离。cVM 的设计具有以下特点：(1)旁路通信：cVM 间是相互信任的，其通信绕过了主机操作系统内核以提高性能，cVM 使用按需访问用于通信的内存区域能力而不影响相邻的内存。(2)低开销隔离：cVM 使用内存能力指针来降低进程和模块的隔离开销，例如 cVM 可以隔离共享库，并对调用接口进行最小的更改。(3)强兼容性：cVM 使用 CHERI 的混合模式，只需要更改使用新的通信 API 就可在应用程序中使用内存能力特性。

图 5 展示了 cVM 的整体架构，每个 cVM(图中 A 所示)都是一个应用程序组件，可以分为三个部分(1)程序的二进文件与二进制库、(2)标准 C 库文件和(3)库操作系统。

cVM 架构中添加了两个新的隔离区边界，通过内存能力强制执行：(1)监管程序边界(图中 B 所示)，将 Intravisor 与所有 cVM 分开以及 cVM 彼此之间隔离开。Intravisor 负责 cVM 的生命周期和隔离，允许它们之间的安全通信，并提供其他无法在非特权库操作系统中实现的原语(例如，存储和网络 I/O、时间、线程和同步)，它可以访问所有 cVM 的内存。(2)程序边界(图中 C 所示)，它将程序与库操作系统分开，该库操作系统为所有操作系统原语提供命名空间。因此，单个库操作系统实例可以托管多个相互隔离的程序及其自己的代码和数据(图 1 中最左边的 cVM)。这些隔离边界由 CHERI 功能强制执行；分区内容不能访问超出其边界的内存，除非通过下面描述的受控接口。最后，使用 CPU 环和基于 MMU 的隔离，与主机操作系统有经典的分离。

cVM 两个评估平台：(1)基于单核 FPGA 的 CHERI 实现；(2)没有 CHERI 支持的多核 SiFive RISC-V 实现。使用 NGINX、Redis、Python 三类工作负载评估了 cVM 的性能，在具有多个应用程序组件的典型部署中，与容器相比，cVM 可以在降低延迟和增加吞吐量的同时实现隔离。这种性能提升是由于减少了内存副本的数量，使用快速调用 cVM 中隐藏 TCB 的内存能力。此外，与容器相比，cVM 具有更小的 TCB。

3.2 对象重用

Sartakov 等人基于 CHERI 架构提出了一种新型的带有内存能力增强的软件分区隔离机制和二进制对象共享机制 ORC[5]。在图 6 和图 7 的例子中，app1 与 app2 在不同的虚拟机上使用了同样的二进制对象 database、libC 和 kernel，图 6 中为一般系统中为了保存虚拟机间隔离性与安全性的虚拟机部署方式，在这种模式下虚拟机监控程序 Hypervisior 需要使用页面级别的内存去重方法来提升内存密度；而图 7 展示了 ORC 方法提升内存密度的原理，app1 和 app2 运行在两个独立的隔离区中，两个隔离区部署在同一个页表上，通过内存能力指针实现对共享内存空间的非重叠内存访问，实现细粒度的二进制内存对象共享，其中页表和内存能力指针由可信计算基 Intravisor 托管。

尽管分区 1 和分区 2 共享了相同的二进制对象，但是还是保证了两个分区的强隔离性：分区 1 和分区 2 都有自己的操作系统实例，并且被限制只

能访问不重叠的内存地址范围。分区间通过内存能力指针跨隔离区共享对象，对二进制对象的内容进行访问。如果可能，ORC 加载程序向 Intravisor 程序请求已由另一隔离区加载的对象的内存能力指针。否则，隔离区会加载对象本身，并将其注册到 Intravisor 中，允许其他隔离区重复使用该对象。

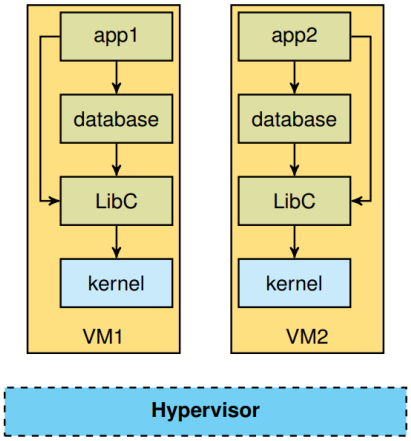

图 6 Hypevisor 工作原理

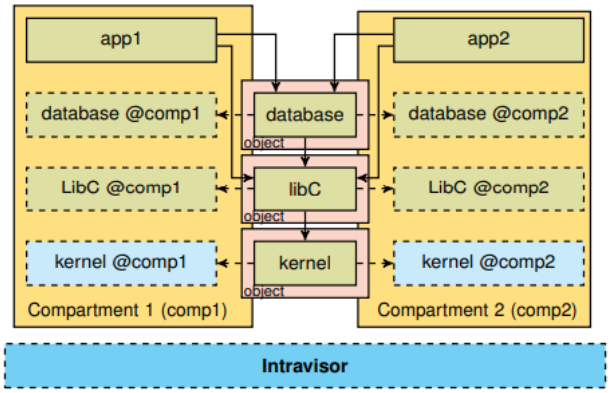

图 7 Intravisor 工作原理

ORC 具体的工作流程如下：(1)首先 ORC 需要对待装载程序进行编译，找到潜在的可共享对象，它会将具备全局可写对象的可写状态移动到隔离区自己的本地内存空间中，而待装载程序的代码、常量则编译为共享对象；(2)然后 ORC 会为共享对象分配内存空间来加载该对象，并为执行该对象做好准备；(3)加载完成后，隔离区调用 Intravisor 的 ORC_register API 向监管程序注册共享对象，监管程序会把共享对象保存在自己托管的专用隔离区中，并计算注册对象的哈希值与其绑定，用于后续的使用，在之后的运行过程中，其他隔离区都可以使用 Intravisor 中已经完成注册的共享对象；(4)当共享对象在监管程序 Intravisor 中完成注册时，其他隔离区可使用 ORC_request()API 来请求获取共享对象，发起请求的隔离区会将自己本地的对象内容哈希值一同传递给监管程序，监管程序与请求对象在自己托管的共享隔离区中同名对象的哈希值进行匹配，如果对象内容哈希值完全一致，则会将该对象的内存能力指针传递给发起请求的隔离区，如果不相同则是具备不同的可写状态，需使用装载程序在隔离区本地内存中为该对象分配内存空间来加载这个变量，完成后整个应用程序就可以执行。

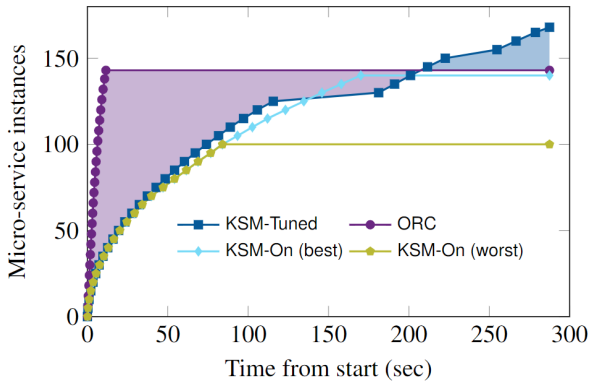

图 8 ORC 性能测试

ORC 在实现中基于搭载 Armv8-A CPU 的 Morello 开发板，该开发版支持 CHERI 架构，测试工作负载选用了实时视频转码微服务，使用 Ffmped 执行转码。如图 8 所示，在与 KSM 算法的对比实验中我们可以看到，ORC 中的实例创建和重复数据删除上比 KSM 的效率更高，尽管最终 KSM-Tuned 内存密度好于 ORC，但是达峰时间太长，不利于云中实时任务的处理，ORC 能在短时间内快速部署更多的服务实例，以实现云中服务的快速响应和实时处理，绝壁更好的系统吞吐量。后续实验也显示在云中使用 KSM 机制将 p99 延迟增加了两倍以上，而在云中使用 ORC 的 p99 延迟只增加了 12%-17%，有效解决了提升内存密度带来的尾延迟问题。

4 总结

综上所述，目前针对数据中心中内存密度提升可以分为两个大的方向，即(1)基于传统的内存重复数据删除方法进行优化和(2)基于 CHERI 新型指令集架构使用内存能力进行优化。这两者各有优缺点，第一类方法通过数据访问模式、数据访问频率等信息来有效平衡计算资源与内存资源，第二类方

法通过使用内存能力的访问控制机制来实现对消除额外运行时计算开销并节省内存资源，这两者都能达到上佳的内存密度提升效果，但是第一类方法本质上没有跳脱出内存页面扫描的开销，在长远的未来随着服务规模的越来越大其性能有待进一步考证，而第二类方法需要特定的硬件支持，难以在当今数据中心中有效快速铺开。随着新型高速互联协议(如 CXL)出现、闪存成本降低、新型存储介质(如 NVM)的发展，如何提升数据中心内存密度会有新的解决方案。

参 考 文 献

[1]S. Fan and Z. Hua, ‘ISA-Grid: Architecture of Fine-grained Privilege Control for Instructions and Registers’.

[2]H. Ji et al., ‘STYX: Exploiting SmartNIC Capability to Reduce Datacenter Memory Tax’.

[3] S. Narayan et al., ‘Going beyond the Limits of SFI: Flexible and Secure Hardware-Assisted In-Process Isolation with HFI’.

[4] V. Sartakov, D. Eyers, T. Shinagawa, and P. Pietzuch, ‘CAP-VMs: Capability-Based Isolation and Sharing in the Cloud’.

[5] V. Sartakov, L. Vilanova, M. Geden, D. Eyers, T. Shinagawa, and P. Pietzuch, ‘ORC: Increasing Cloud Memory Density via Object Reuse with Capabilities’.

[6] D. Saxena, T. Ji, A. Singhvi, J. Khalid, and A. Akella, ‘Memory deduplication for serverless computing with Medes’, in Proceedings of the Seventeenth European Conference on Computer Systems, 2022.

[7] L. Yao, Y. Li, F. Guo, S. Wu, Y. Xu, and J. C. S. Lui, ‘Towards High Performance and Efficient Memory Deduplication via Mixed Pages’, IEEE Transactions on Computers, pp. 926–940, Apr. 2023.

附录 1. 论文分析问答记录

提问：论文ORC中提到的微服务不同容器间的数据重删，docker已经通过不同层次的镜像实现了，为什么这篇文章还要做这个呢？

回答：

(1) docker镜像的不同层次文件系统是对镜像而言的，它是用来减小镜像导出大小，而ORC是对容器实例而言；

(2) 在容器实例层面上来说，docker的去重粒度是数据块层级，涉及大量的哈希计算和相似块对比，而ORC的去重粒度是对象层级，相较于docker自身去重机制粒度更小；并且docker的去重是在运行时进行，周期性扫描内存页面开销较大，而ORC仅在用户空间加载执行程序时进行，没有运行时开销，资源占用更低。

(3) 此外docker去重机制是由宿主机操作系统托管，文章说到操作系统攻击面太大，安全性不佳，而ORC通过独立的监管程序实现，接口少攻击面更小更加安全。