上海交通大学

计算机系统原理课程论文

硬件资源分离

作者:
杨健邦

指导老师:
夏虞斌

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy

in the

Research Group Name
Department or School Name

January 10, 2019

Contents

Chapter 1

内存分离

内存资源在数据中心中相对昂贵而且稀少的资源，如何利用好内存资源一直都是学术界以及工业界的一个研究重点。其中，一个主要研究方向是内存分离（memory disaggregation）。将内存资源从一体化服务器（monolithic server）中解耦出来，服务器上的应用能同时访问“本地”的和“远端”的内存，看上去就像是访问一块内存一样，这样的技术被称为内存分离（memory disaggregation）。通过内存分离，应用程序能使用到更多的内存资源，也减少了内存利用率不高而导致的资源、能源浪费的情况。本章节将主要通过介绍INFINISWAP系统，来介绍内存分离的一个具体实现并通过这个具体实现来理解。

在本章的开始，我们先介绍内存分离技术的研究背景，从而了解数据中心对内存资源的需求和内存分离技术出现的缘由。再分类地说明内存分离的研究现状，介绍内存分离不同的研究方向以及其对应的技术。接着，我们着重介绍再内存分离技术领域比较先进的INFINISWAP系统。最后我们对内存分离技术做一个总结，并提出一些我们的问题和思考。

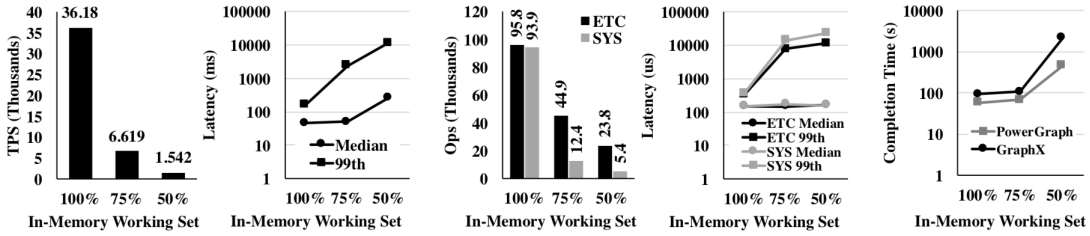
1.1 背景介绍

1.1.1 问题的由来

内存不足的现象是自内存设备出现以来就已经存在的问题。在过去，解决内存不足问题的目标仅仅是说让程序在内存不足的时候也能正确运行，因此产生了虚拟内存（virtual memory）、换页（paging）等技术来缓解内存不足的状况，本质是通过硬盘来暂存内存数据，这样的方法虽然解决了内存不足的问题，但性能却很差。

现在，虽然单机上的内存的容量已经增大了很多，但内存不足的情况依旧存在。因为应用程序对于内存的需求也越来越大了，而换页的性能很差，并不能满足应用的需求。除了内存不足的问题，新的问题也逐渐涌现。并不是所有的应用程序都会都将本机的所有内存全部使用，因此出现内存过剩的资源浪费问题。

内存不足与内存过剩是一对矛盾，却又会同时出现，这展现出的是内存使用的不均衡性。其根本原因是传统的一体化服务器的架构给内存的部署、分配和使用所带来的限制。一方面，单机上内存资源在服务器部署之后就已经基本上是确定不变的了，服务器部署运行之后再去拓展内存资源是一件麻烦而且成本大的事情。另一



(a) TPC-C on VoltDB (b) Facebook workloads [26] on Memcached (c) Analytics

FIGURE 1.1: 在不同内存容量下应用程序的性能。

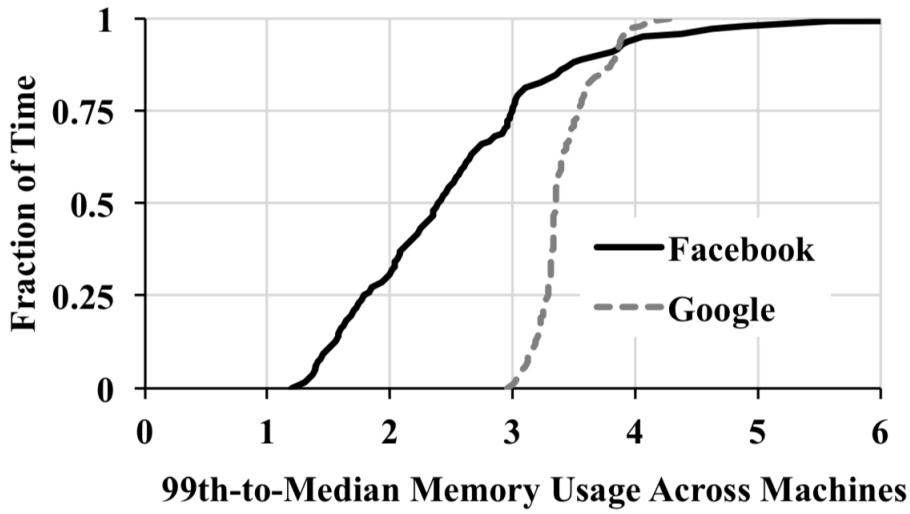


FIGURE 1.2: Facebook和Google的两个机器集群中存在的内存使用不均衡现象。

方面，单台服务器所能支持的最大内存是有限，这必然不能满足应用程序日益增长的内存需要。

1.1.2 问题的研究意义

首先，为了说明内存不足而导致换页的不利影响，有相关研究人员做了一些测试。他们选择了三种不同的内存应用程序进行测试：（i）运行在VoltDB内存数据库上的TPC-C基准测试程序；（ii）运行在Memcached键值仓储中的模拟facebook工作负载的程序；（iii）运行在PowerGraph的TunkRank算法，使用的数据集是Twitter。

为了避免由外部因素造成的性能影响，测试只关注应用程序在单机下的性能（图1.1）。结果显示由于内存不足所导致的换页确实对应用程序产生重大的、非线性的、急速下降的性能影响。另外，换页会导致非常明显的尾延迟现象。以上的现象都表明，内存不足是一个非常重要的问题，进行内存分离的研究是十分有必要的。

其次，不同机器中的内存使用也存在着不均衡的现象，这将导致资源浪费。有相关的研究人员统计了Google和Facebook的两个实际运行的集群中机器内存使用情况

的一些数据（图 1.2）。通过记录和计算10秒内前99%的机器的平均内存使用率与所有机器平均使用率的比值，来表示内存使用的不均衡性。结果显示，集群中有超过一半的内存因资源利用不均衡而导致其未被使用到的。这样的资源利用不均衡也验证我们的说法。

1.2 当前内存分离技术的分类和对比

很长一段时间以来，数据中心都在使用着一体化服务器的架构，使得大多数内存分离技术都是基于这种架构去设计并实现的。如分布式共享内存（distributed share memory）技术和远程换页（remote paging）技术。那么，这些方法。

1.3 基于RDMA高速网络的内存分离技术：INFINISWAP

INFINISWAP是针对RDMA高速网络设计的一个远程换页系统，通过将集群中每台机器的交换分区划分成大块去管理，适时地去收集机器中未使用的内存，并把这些内存暴露给应用程序使用。整个过程对于应用程序来说是透明不可见的，因而应用程序不需要做任何修改。在接下来的小节中，我们将系统设计和性能测试来对INFINISWAP系统进行介绍。

1.3.1 设计目标

INFINISWAP的主要设计目标是高效地将集群中所有机器的内存都暴露给应用程序去使用，不需要对应用程序或者操作系统做任何修改。系统必须具备可扩展性、容错性和透明性，同时做好隔离，使用远程机器上的内存不能打扰到远程机器上应用的性能。

1.3.2 架构概述

INFINISWAP包含了两个主要的组件——INFINISWAP块设备和INFINISWAP守护进程。每台机器上都包含着这两个组件，因此每台机器的角色相同，从而实现了一个去中心化的系统（图 1.3）。

INFINISWAP块设备

INFINISWAP块设备向虚拟内存管理器提供常规的块设备IO接口，让虚拟内存管理器将这个设备当成是一个交换分区。当出现换页的时候，INFINISWAP块设备便可以透明地通过RDMA操作向其它机器中读取内存或写入内存。

INFINISWAP将每台机器上的内存地址空间划分成固定大小的大块（slab），大块是INFINISWAP内存映射和负载均衡的基本单位，它由很多内存页所组成。以大块为单位进行内存映射处理的原因尽量减少远端机器上CPU的参与，减少对远端机器上应用程序的打扰，同时提升远程换页的速度。当大块映射完毕之后，INFINISWAP块

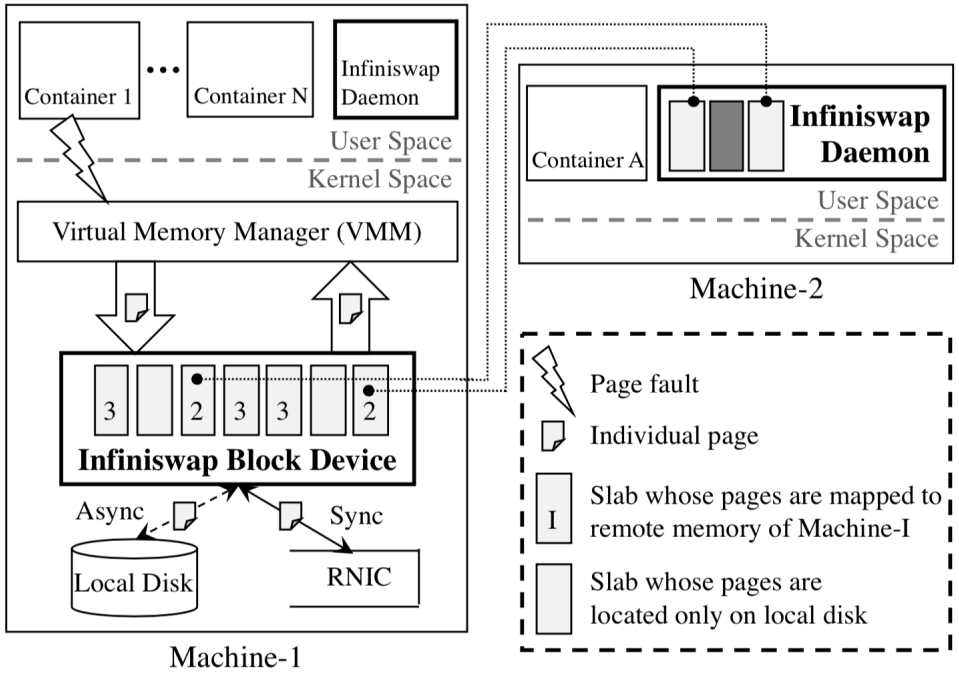


FIGURE 1.3: INFINISWAP系统架构。

设备便可以通过RDMA读写操作来进行换页，换页的基本单位仍然是内存页，而不是大块。

INFINISWAP守护进程

INFINISWAP守护进程是运行在用户态的一个程序，仅仅参与控制层面（control plane）的活动。它仅仅负责处理其它机器发过来的大块映射请求以及预先分配相应的内存空间。真正的对数据层面（data plane）的操作则是通过RDMA请求然后由网卡去执行，并不会打扰目标机器的CPU。

1.3.3 系统设计

透明性和隔离性设计

透明性 INFINISWAP提供与传统块设备一样的接口和语义，应用程序不需要做任何修改便可以使用到更多的内存，它也不知道自己使用的是本地内存还是远程内存。这样的透明性使得INFINISWAP系统具有很好的兼容性与通用性，能在现有的数据中心快速通入使用，不需要增加新的硬件。

隔离性 INFINISWAP使用RDMA高速网络来传输内存页数据，不需要打扰远端CPU的执行。对远程CPU的打扰主要在于大块的映射，但大块的大小是比内存页要大很多的，这样的一个设计也为了减少对远端机器CPU的打扰次数。

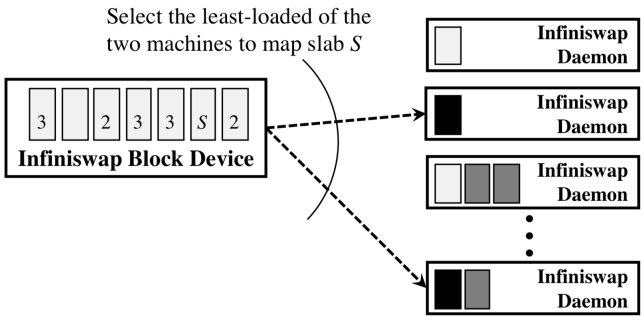


FIGURE 1.4: INFINISWAP块映射策略。

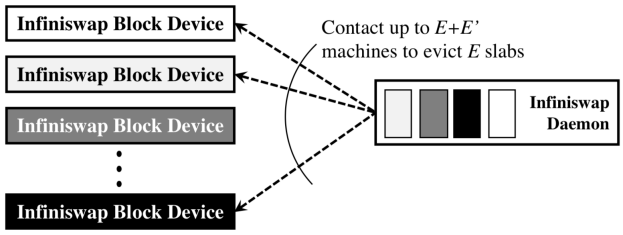


FIGURE 1.5: INFINISWAP块淘汰策略。

容错性设计

INFINISWAP对应用程序是透明的，应用程序认为远端机器上面的内存也是存在于本地的。因此，INFINISWAP需要提供与应用程序只使用本地内存一样的语义，即当远端机器出现错误时，不能影响到本地应用的运行。

容错的方案是在进行RDMA远程换页的同时，异步地将内存也写入本地硬盘中做为备份。由于写备份是异步的，并不在关键路径上，因此也不会对性能产生很大的影响。

实现这样的容错方案要处理一些边界问题，最重要的边界问题是处理*read-after-write*的情况。即当内存页已经写到远端的内存但还没有写入本地硬盘中，在这时候远端机器发生了崩溃。如果在这同时应用程序又要读取这块内存页，块设备便需要从硬盘的写队列中读取内存页，然后返回给应用程序。

可拓展性设计

INFINISWAP并没有一个中心化的设计，避免由中央的协同器成为系统的瓶颈而导致系统不具有可拓展性。去中心化的设计虽然能够使系统具备一定的可拓展性，但缺少中央协同器来收集全局的信息，这会带来很多问题。

其中关键的问题，如何制定好的大块映射策略来使得集群中每台机器的内存使用情况都趋于均衡。 INFINISWAP使用了*power-of-choices*的技术来作为块映射以及块淘汰的策略。比如*power-of-two*的块映射策略，会先随机地选择两台远程机器，比较

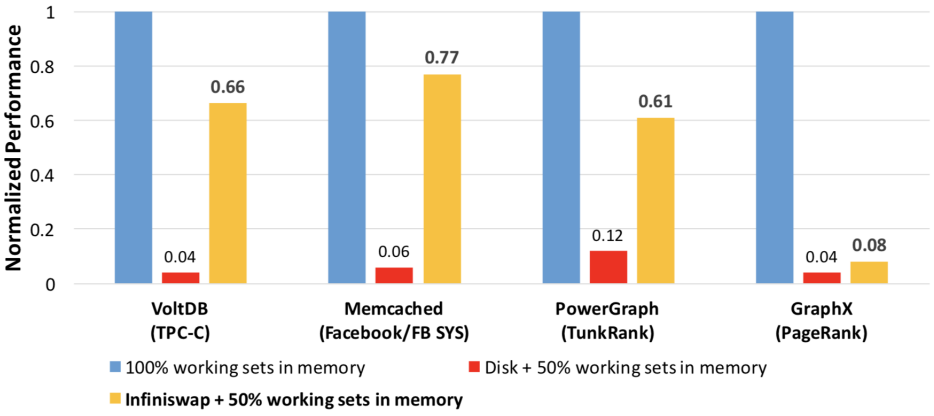
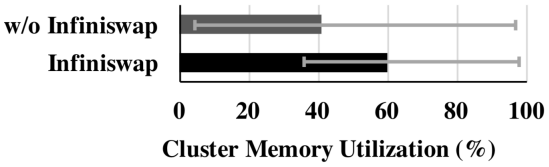
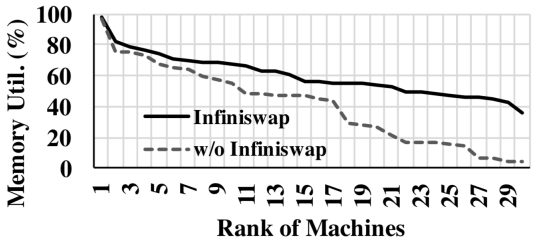


FIGURE 1.6: 应用程序的性能



(a) Cluster memory utilization



(b) Memory utilization of individual machines

FIGURE 1.7: 集群的内存使用情况

两台机器的内存使用情况，然后选择内存使用较少的一台机器作为目标机器来进行块映射。类似地，块淘汰策略也是使用了*power-of-choices*的方法。

1.3.4 系统测评

测试使用32台机器的集群，机器之间用56Gbps的Infiniswap网卡连接。每台机器有2个NUMA节点和64GB的内存，每个NUMA节点有8个物理CPU，一共32个vCPU。

应用程序的性能 用于测试的应用程序有VoltDB、Memcached、PowerGraph和GraphX。测试分别测了单机内存能满足应用程序100%的内存需求、能满足应用程序50%的内存需求和满足应用程序50%的内存需求并使用INFINISWAP系统三种情况下应用程序的性能（图 1.6）。最终的测试结果表明，在单机内存只能满足应用程序50%的内存需求时，使用INFINISWAP能将应用程序的性能提升2到16倍。

集群的内存使用情况 测试创建了90个容器运行在集群上，每个容器使用不同的内存需求配置。测试的结果显示集群的平均内存利用率从40.8%提升到了60%，提升了1.47倍。而内存的不均衡现象也有所缓解（图 1.7）。

1.4 本章小结

Chapter 2

LegoOS

2.1 研究背景

现阶段，数据中心的部署、执行、故障单元等都是一整台的物理服务器（monolithic server），这台服务器包含了运行一个程序所需要的全部硬件资源（CPU、内存、磁盘等）。但是，这种架构使得硬件资源很难得到充分的利用。例如，不同的应用程序对于资源占用的侧重不同：一些计算密集型的程序对CPU要求很高，但是不需要占用很多内存；而一些数据存储服务程序则要求磁盘和网络具有较大的吞吐量，而CPU性能则其次。单台完整的服务器无法完美适配不同的程序对于资源的使用。另外，即使服务器只有一个硬件组件故障，也会使整体不可用。

硬件资源分解（hardware resource disaggregation）是一种新的技术，把不同的硬件设备组织为独立的组件，组件之间依靠网络进行通信而不是集中在一台服务器中。这样可以最大化硬件资源的利用率，弹性装配，还能够隔离硬件之间的故障。

2.2 相关研究和挑战

- 硬件发展：首先，网络的速度越来越快，在逐步接近内存总线的数量级，这使得基于网络的硬件通信在性能上成为可能。其次，硬件设备的功能越发丰富，因而可以直接在硬件上实现控制器、处理网络请求等，无需软件的辅助。
- 内存分离：基于RDMA技术（远程直接内存访问），目前已有大量远程内存利用的研究。例如，在一个集群中，内存负载大的服务器可以把一些页面交换到内存负载轻的服务器上而不是本地磁盘，从而获得更高的效率。[1]
- 存储分离：有关分布式存储的研究十分丰富，而且已经大规模的部署在了云服务商的机器上。
- 多内核操作系统：一些操作系统在每个CPU核心上运行一个小的kernel，并使用消息机制通信。这样，运行在任何物理核心上的kernel代码都可以快速替换。[2]

以上研究已经分别实现了内存、存储、核心等的分离，但OS主体仍然运行在monolithic server上，OS仍然需要管理不同的硬件资源（CPU，内存，和存储等）。LegoOS是

全新设计的体系结构。与传统的基于monolithic server的方案相比，它能够真正完全分离硬件资源，每个组件（component）只管理属于自己的硬件资源（CPU，内存，或存储中的一个）。例如，管理CPU的组件将没有本地的内存和存储，而是通过网络与内存组件和存储组件通信。这样做可以实现内存和CPU等资源的完全利用，而且方便随时添加和移除硬件组件，甚至是完全异构的硬件；进一步的，硬件故障的影响范围被局限在了每个独立的硬件组件而不是整台服务器。这些优点是monolithic server的方案所无法做到的。

当然，这带来了许多挑战：在组件之间的网络通信速度达不到总线的速度时，如何保证提供良好的性能；如何协调与管理所有独立的组件；如何确保一个组件的failure不影响到其他组件；如何兼容现有应用；等。

2.3 架构和设计

LegoOS在设计上针对三种硬件组件：处理器、内存、存储，分别称为pComponent、mComponent、sComponent。在接口方面，LegoOS暴露给用户的是一组vNode。每个vNode有自己独立的IP，可以运行多个pComponent、mComponent、sComponent。LegoOS确保每个vNode的资源相互之间完全隔离。LegoOS实现了大多数的Linux system call接口，使得大多数程序可以未修改的在LegoOS的一组vNode上运行。

2.3.1 处理器管理

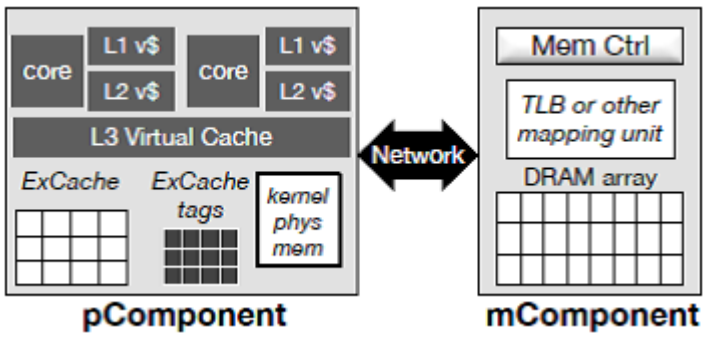


FIGURE 2.1: LegoOS pComponent and mComponent Archiecture

pComponent专门管理处理器内核。它使用了针对数据中心程序的一个简单的调度模型。每个pComponent为内核线程保留少量的核心，然后分配其他的核心给用户线程。并且，尽量减少线程调度和内核抢占以提高性能（例如，不使用中断而是用轮询处理网络请求，由于网络延迟在LegoOS中很低）。

pComponent完全不需要地址映射，mmu和TLB等完全由mComponent维护。在本地，pComponent只持有少量的内存缓存，称为ExCache，位于处理器的Last-Level Cache（LLC）之下。每个ExCache line有一个虚拟地址的tag和两个标记位（P，R/W），由软件设置、硬件检查。当ExCache miss时，LegoOS通过网络从对应的mComponent中

取回数据填入ExCache line中。ExCache miss的处理甚至可以完全由硬件实现以获得最高的效率。

2.3.2 内存管理

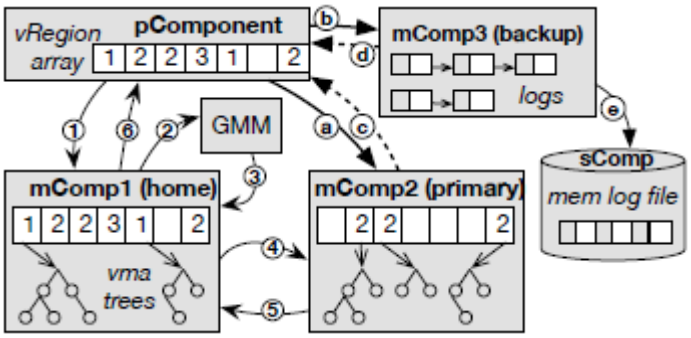


FIGURE 2.2: LegoOS Distributed Memory Management

mComponent管理虚拟内存和物理内存，包括它们的分配、回收和映射。这里采用了两级的内存管理。在high level，把虚拟地址空间粗粒度的划分为了固定大小的vRegions，每个已被分配的vRegion都被一个mComponent管理。在low level，mComponent记录了每个vRegion上分配给用户的vma树。

当用户程序申请虚拟内存空间时，pComponent把相关的请求转发给对应的mComponent。mComponent会查询自己管理vRegions，寻找一个合适的vRegion来为用户分配内存。如果可用虚拟内存空间不足，还会向全局的内存资源管理器GMM申请一个新的vRegion并把请求转发到另一个mComponent。pComponent也会缓存用户程序的vRegion array。

考虑到内存故障的可能性与影响更大，LegoOS对mComponent提供了可靠性保证。当pComponent刷新ExCache时，消息被同时发往primary mComponnet和对应的backup mComponent。其中primary mComponent维护内存数据和元数据，backup mComponent在后台把内存操作写入由sComponent管理的append-only log中，这样可以在失败时通过log恢复内存内容。

2.3.3 存储管理

LegoOS在sComponent上实现存储功能。类似于NFS，存储服务器采用无状态的设计，并通过vNode抽象后暴露给用户。用户可以通过标准POSIX api操作vNode上的挂载点，从而访问存储系统。

由于sComponent的内部内存有限，LegoOS在mComponent上放置存储缓冲区。当用户通过系统调用访问文件时，抽象层把请求的文件完整路径、偏移量和大小转发给mComponent，mComponent查找缓冲区，并在需要时从sComponent获取缺失的数据以及将文件数据sync到sComponent中。

2.4 本章小结

LegoOS是第一个为了硬件资源分离而设计的操作系统，它将一个操作系统管理不同硬件的功能分解开。相比于monolithic server，LegoOS可以达到更有效的资源利用率、更高的故障隔离性以及更好的硬件资源弹性。测试和评估表明，经过优化的LegoOS有着很低的网络延迟和较好的内存吞吐量，以及更长的平均无故障时间（MTTF）。而且，随着硬件功能的进一步丰富，一些componnent监视器的逻辑可以由硬件完成，从而进一步提高性能。

Bibliography

Gu, Juncheng et al. “Efficient Memory Disaggregation with Infiniswap.” In: *NSDI*. 2017, pp. 649–667.

Zellweger, Gerd et al. “Decoupling Cores, Kernels, and Operating Systems.” In: *OSDI*. Vol. 14. 2014, pp. 17–31.