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、唐楚哲、黄政、谢镇涛

指导老师:
夏虞斌

2019 年 1 月 13 日

目录

Chapter 1

总述

硬件资源，包括计算机处理器计算资源、内存容量、存储设备容量与 IO、网络带宽等，是处理各类计算任务的核心。随着时代的进步，工业界的需求对硬件资源不断提出新的要求，驱动着硬件技术的发展与进步。其中，硬件资源分离作为本文讨论的主题，是一种新兴的提高硬件资源利用率的研究方向与技术类别。本章将从计算的发展开始介绍，列举当前大规模计算下的挑战，并阐述为何硬件资源分离成为了当前备受关注的解决方案。

1.1 计算的发展

1.1.1 计算能力的增加

在过去几十年，计算机计算能力不断提升。从最早的电子管、晶体管，到现在普遍使用的多核处理器，计算翻倍的摩尔定律几乎从未失效。现在，英特尔最新的第九代酷睿处理器，处理器频率最高能达到 5GHz [7]。除此之外，计算机内存技术、存储技术、网络技术等也都在不断地发展。内存除了容量与速度不断增大之外，还出现了许多新的特性，比如非易失性（non-volatile)[23]。存储技术的发展也与内存相似，随着新的介质提出与工艺改进，在容量与速度上存储技术稳步提高。与此同时，在不同介质下，存储设备还会有细微的性能特性，比如随机与顺序访问的性能差异。其他支撑技术也在近年来得到不断发展。

1.1.2 计算需求的发展

与计算能力增加同时进行的，还有计算需求也在不断的变多、变高。最初计算机仅用于科学计算，且数据量在现在看来是非常小的。随着计算机走向民用与新世纪互联网的普及，各式各样的计算任务被提出，多种计算需求同时出现。现在，在不同的场景下，计算任务有着截然不同的计算需求，并且可以被分入多种类别。有关注低延时的计算任务，也有关注可靠性的计算任务，有需要读取大量数据的计算任务，也有将大部分时间用在计算上的计算任务。

1.1.3 计算方式的变化

从单机计算，到小型分布式系统，再到支撑与计算平台的数据中心，计算的方式发生了巨大的转换。当计算需求提高，计算任务的弹性增加，往将的计算任务用在单独特用的计算机的方式不再流行。通过建立数据中心，运行着多个分布式计算系统，将各式各样计算任务收集在一起，协调调度并执行，计算资源得到更有效地利用，而单一使用者空闲导致资源限制的问题得到缓解。

1.2 大规模计算下的资源挑战

为了处理大规模的计算，现有的系统往往面临多多方面的挑战，包括：

- 最大化利用计算资源；
- 克服节点间带宽的限制；
- 低成本运维海量硬件；
- 在不可靠硬件基础上提供高可靠的计算能力和存储能力；
- 提供高可用服务等。

其中，资源利用率问题是我们在本文关注的重点。

数据中心存放管理这大量的计算资源，并且可以动态地分配给不同的资源消费者，及各类计算任务或者进一步的资源管理者 [24]。早期常用的方法是直接通过搭建在多台机器上的分布式系统，直接将任务，显式地分配给某台机器进行执行。为了便于管理与调度，现在以虚拟机与容器为单位的资源管理变得流行。无论是什么方式，都存在着各种任务对资源的需求不一致，难以统一协调，最终导致资源分配不足或者资源浪费的问题。在商用计算平台上，因为出服务的稳定性要求，资源超供导致的闲置问题更为突出。如何在保证服务的质量前提下，如何更高地提高资源利用率成为了当先热门的商业痛点与研究方向。

1.3 硬件资源分离

硬件资源分离是一类关注在硬件层面，尝试为了提高资源利用率的方法。其核心思想是，不将硬件资源与本地计算机强绑定在一起。允许通过特定的机制，将不同机器的某项资源同时服务于某项计算任务，从而提高整体的资源利用率，加速计算的完成。

在本文中，我们将从关键的两类资源——内存与存储的资源分类入手，介绍针对单一资源的分离是如何完成的。随后，我们将从系统层面的资源分离，即如何将多种资源在统一的框架下进行分离使用进行介绍。每部分内容都会配合相关的最新研究成果作为案例分析。

Chapter 2

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。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，并通过这个案例来理解内存分离的作用与意义。

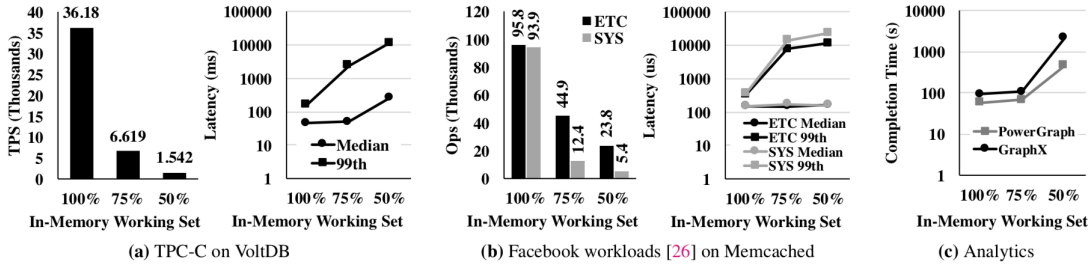
在本章的开始，我们先介绍内存分离技术的研究背景，从而了解数据中心对内存资源的需求以及内存分离技术出现的缘由。再介绍内存分离的研究现状，介绍内存分离不同的研究方向以及其对应的技术。接着，我们介绍内存分离技术领域比较先进的 INFINISWAP 系统 [6]。最后我们对内存分离技术做一个总结，并提出一些我们的问题和思考。

2.1 背景介绍

2.1.1 问题来源

内存不足 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，而换页需要读写硬盘，性能很差，并不能满足应用的需求。内存不足的问题导致的是应用程序的性能下降。

内存使用不均衡 除了内存不足的问题之外，在数据中心集群内部，内存使用还有不均衡的问题。对于单台服务器来说，它将会运行各式各样的应用程序，而且会同时运行多个不同的应用程序。每个应用程序对内存的使用需求并不一致，但是每台服务器的可用的内存容量却是固定的。因此，必然会出现某些服务器在某段时间内，内存并



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

图 2.1: 在不同内存容量下应用程序的性能 [6]。

不会完全被使用，甚至利用率很低。结合上面上面的内存不足问题，在集群中，就可能出现这样的现象：一部分机器的内存不足，另一部分机器的内存过剩，这就是内存使用不均衡的现象。内存使用不均衡的问题导致的是资源浪费。

无论是内存不足的问题还是内存使用不均衡的问题，它的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或减少内存资源是一件非常麻烦的事情。另一方面，单台服务器所能支持的最大内存有限，而且机器之间不能直接使用对方的内存，这必然不能满足应用程序日益增长的内存需要。

2.1.2 问题展示

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 [6]。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 2.1）。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 2.2）。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了我们的说法。

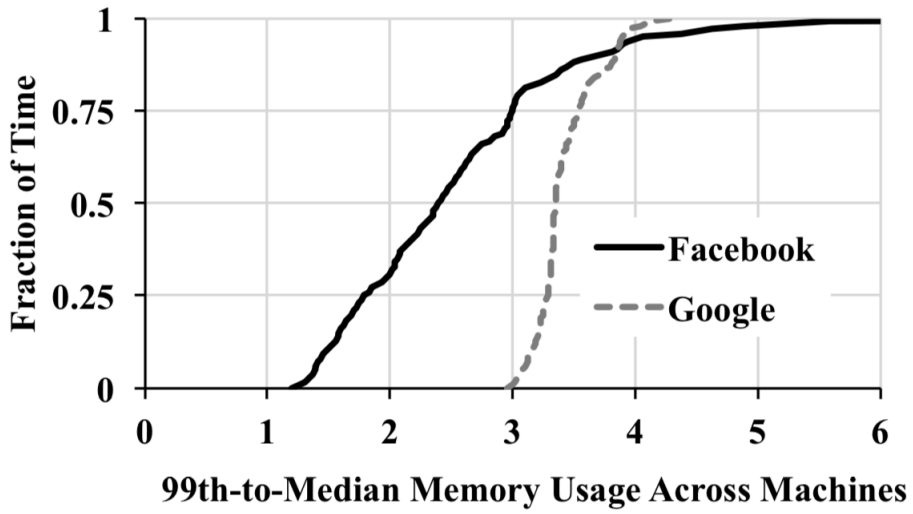


图 2.2: Facebook 和 Google 的两个机器集群中存在的内存使用不均衡现象 [6]。

2.2 当前内存分离技术的研究现状

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。这些方法考虑的是易用性、可行性，不需要对现有服务器架构设计进行太大的修改。

内存分离技术有多种的划分方法。LegoOS[17] 的研究人员按照软硬件的划分将内存分离技术分成两大类，一类是硬件辅助的内存分离技术，指的是设计一些专用的小硬件来辅助和加速内存分离；另一类是纯软件实现的内存分离技术，只需要现有的硬件设备来实现内存分离。而 INFINISWAP 的研究人员则是按照实现方法来划分，将内存分离技术主要分为远程内存换页技术（Remote Memory Paging）和分布式共享内存（Distributed Shared Memory）两大类。

我们认为 INFINISWAP 的划分思路更加贴合内存分离的主题，我们在此基础上做一些补充，将内存分离技术可以分为如下几类：

2.2.1 应用层面的分布式内存管理

将应用部署到多台机器上，每台机器上的应用主要去管理和使用本地的内存，它们之间通过网络获取和共享各自的内存数据，这是应用层面的分布式内存管理。其与其它内存分离技术的最大区别是，应用程序是知晓自己是处在一个分布式集群环境中的，开发人员要根据应用程序的具体情况设计内存的管理以及机器之间数据交换的方法和协议。通常，内存密集型的一些分布式应用通常都有比较好的内存管理和交换方法，比如内存数据库、内存图计算等等，相关的系统有：DrTM[22] 和 Wukong[18]。

应用层面的分布式内存管理解决了单台服务器内存不足的问题，而且它灵活度高，开发人员能够根据应用程序的具体情况去设计高性能的内存管理方法和数据交换方法。但是，这种方法也有缺点，分布式应用编程开发和调试很具有难度，而且它也并没有解决由应用程序之间内存需求不一致而导致的内存使用不均衡问题。

2.2.2 远程内存换页技术

远程内存换页技术指的是，通过网络将本地的内存页换入换出到远端机器的内存中，而不是传统那样换到本地的硬盘 [2, 14, 4]。在过去，远程内存换页技术却受限于低缓的网络传输和过多的 CPU 的额外开销。而现在，由于高速网络如 RDMA 技术的出现，远程内存换页技术的性能有了很大的提升，目前，在这一领域最先进的工作和成果是 INFINISWAP[6]。

远程换页技术对应用程序透明，单机的应用程序可以不需要进行修改便可以使用远程换页技术，因而通用性很好。除了上面所提到的网络传输和 CPU 开销问题，通常的远程内存换页技术还需要中央协同器来进行页淘汰和负载均衡等工作。

2.2.3 分布式共享内存

分布式共享内存系统将集群所有机器的内存在逻辑上组织成一个全局的地址空间，供应用程序去使用，集群内部任何一台机器上的应用程序都可以对这一地址空间进行读写 [1, 12, 16]。过去的分布式共享内存技术因维护一致性而产生过多的通信开销。现在分布式共享内存技术也使用了一些新的硬件，比如 PGAS 和 RDMA，虽然减少了维护一致性的开销，但需要开发人员重写应用程序来使用它的接口。

分布式共享内存同样是对应用程序透明，通用性很好。但它需要维护一致性，而且代价比较大，因而可拓展性可能会受到一定的限制。

2.3 案例：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。

2.3.1 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。

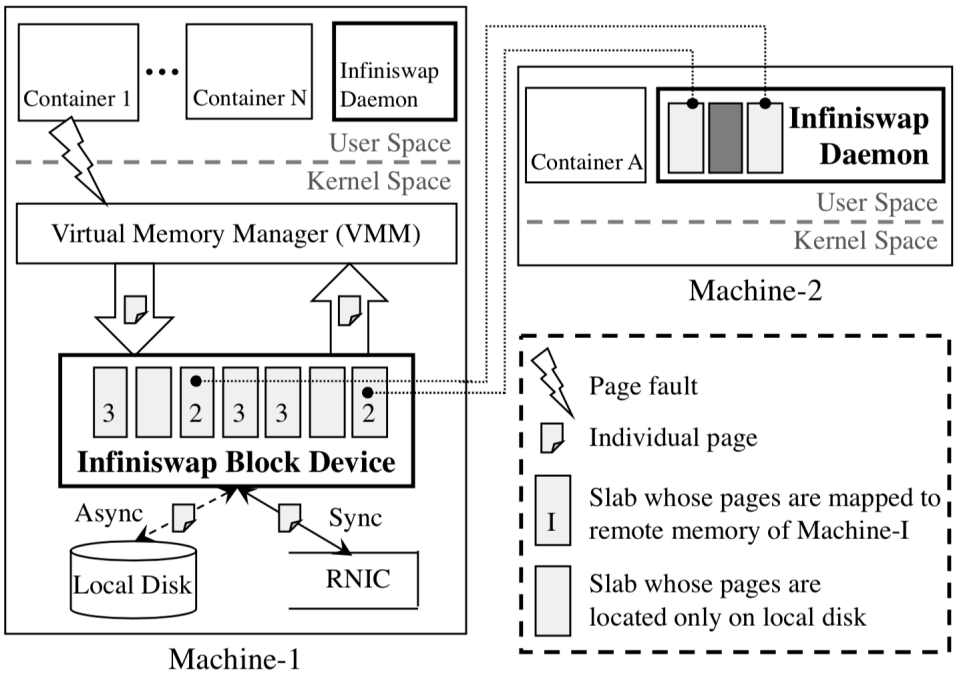


图 2.3: INFINISWAP 系统架构。[6]

2.3.2 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 2.3）。

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

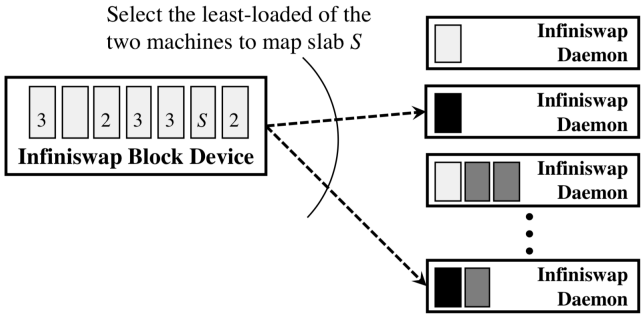


图 2.4: INFINISWAP 块映射策略 [6]。

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。

2.3.3 设计细节与实现

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。

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。

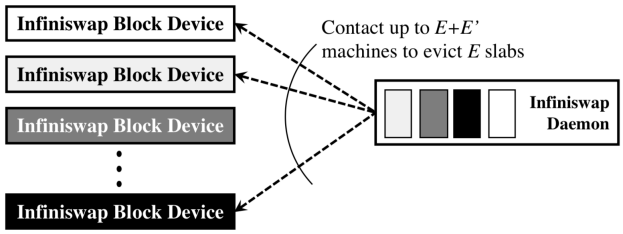


图 2.5: INFINISWAP 块淘汰策略 [6]。

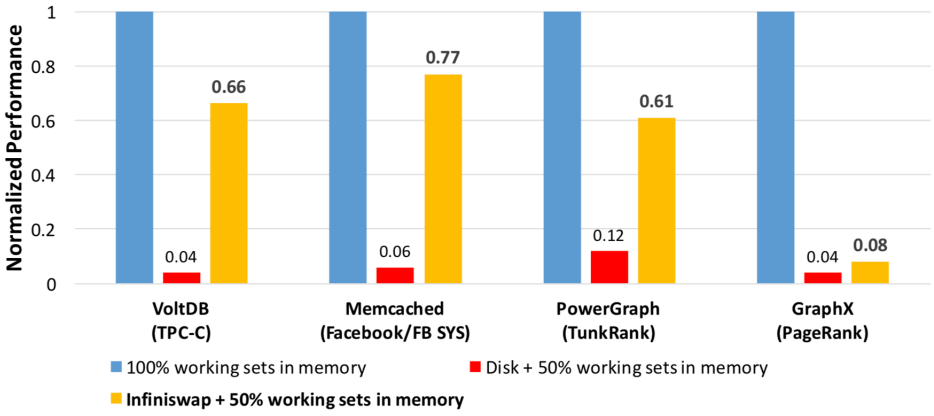


图 2.6: 应用程序的性能 [6]。

可拓展性设计

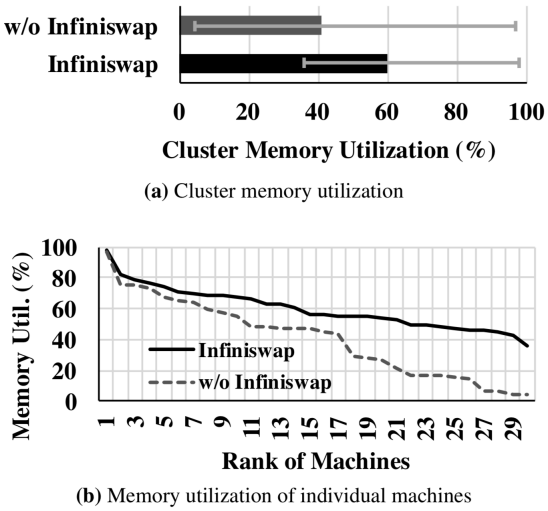
INFINISWAP 并没有一个中心化的设计，避免由中央的协同器成为系统的瓶颈而导致系统不具有可拓展性。去中心化的设计虽然能够使系统具备一定的可拓展性，但缺少中央协同器来收集全局的信息，这会带来很多问题。

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。

2.3.4 系统评测

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



(a) Cluster memory utilization

(b) Memory utilization of individual machines

图 2.7: 集群的内存使用情况 [6]。

种情况下应用程序的性能（图 2.6）。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 2.7）。

2.3.5 案例小结

案例介绍了在内存分离领域最先进的技术 INFINISWAP。它通过 RDMA 技术解决了传统的远程内存换页技术所存在的网络通信慢和过多的 CPU 额外开销的问题。去除了中央协同的模式，从而实现了比较好的可拓展性，还增加了容错处理。它在一定的程度上解决了数据中心当前存在的内存不足以及内存使用不均衡问题。

2.4 本章小结

本章我们探讨了内存分离这一主题，介绍了内存分离的背景、研究现状和技术案例，说明了当前数据中心存在的内存不足以及内存使用不均衡的问题，而内存分离则是解决这些问题的主流方法。

硬件的发展正不断推动着内存分离技术的发展。实现内存分离有很多种技术方案，但无论哪种方案，都需要依靠网络来进行数据传输。传统的内存分离技术最大的限制是网络传输速度的限制。但现在，越来越快的一些网络技术如 RDMA 的出现，使得内存分离真正变得更加切实可行。

我们认为下一步的研究方向是从架构层面去思考如何去做内存分离。内存不足以及内存使用不均衡的根本原因是一体化服务器的架构，基于这样的架构去实现的

内存分离只是缓解问题，并不能从根本上去解决问题。在后面的章节中，我们将介绍LegoOS，看看如何从根本上去解决内存的相关问题。

Chapter 3

存储分离

存储设备是用于存储信息的设备，也被称为外存。相比于内存，它容量大、廉价，但是读写速度慢。与内存相似，单台服务器上面能接入的存储设备数量也是有限的。

然而，无论是容量还是读写速度，应用程序对硬盘资源的需求还是十分巨大的，如何用一种有效的组织方式来管理和使用硬盘资源是数据中心里面的焦点问题。存储分离（storage disaggregation）也是其中一个重要的研究方向。与内存分离的定义相似，存储分离技术让应用程序同时能够访问本地服务器和远端服务器的存储设备，将存储资源从计算节点解耦。

在本章的开始，我们先介绍存储技术的发展以及存储技术所遇到的新挑战。再介绍针对存储技术新挑战相关的一些解决方案，引出存储分离技术。接着，我们以闪存分离 [9] 为例，详细介绍存储分离技术。最后我们对存储分离技术做一个总结，提出我们的思考。

3.1 背景介绍

3.1.1 存储设备的发展

存储设备首先必须能够持久化数据，此外，它有两个重要的参考指标——容量以及读写速度。因此，存储设备出现了两条发展路线，一是向着容量更大的存储设备进行发展，二是向着更快的读写速度的存储设备发展。

存储设备大致经历了打孔纸带、电子计数管、磁带、磁盘、固态硬盘、非易失存储器的发展过程，每一次存储介质的革新都会使得存储设备读写速度的飞跃性提升。英特尔的存储设备技术处于行业领先的地位，它的非易失存储技术 3D XPoint 是目前最先进的存储技术之一。以 3D XPoint 研发出来的傲腾系列商用产品，顺序读写速度已经超过了 2200MB/s，延迟小于 10us[8]。

另一方面，工业界也在不断地去优化存储技术，来提升现有的存储设备的容量。Nimbus 公司在 2018 年研发出来的固态硬盘产品 ExaDrive DC100，容量已经能达到 100TB[3]。

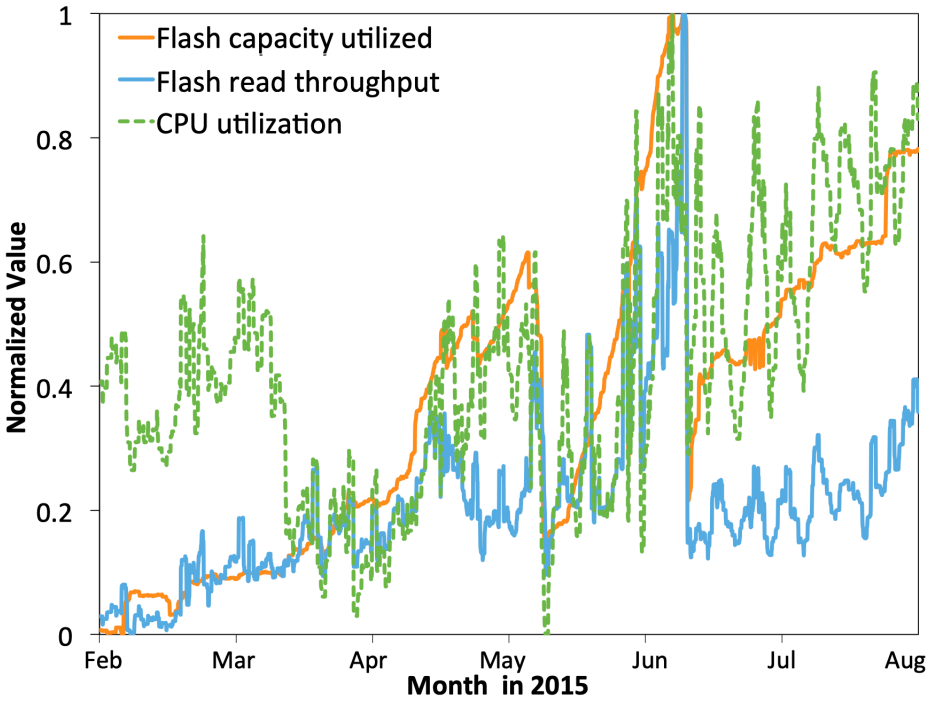


图 3.1: Fackebook 的一个机器集群的资源利用情况 [9]。

3.1.2 存储技术的新挑战

与内存的情况类似，在数据中心里面，存储也存在着单机资源不足和集群资源使用不均衡的问题。但对于存储设备来说，除了要考虑容量资源，还需要考虑 IO 资源。

容量 随着互联网的深度发展，巨大的信息流背后产生着海量的数据。对于互联网企业来说，数百 TB 的数据已经很常见了，单台服务器的存储设备并不能满足需求了，因此我们需要使用到多台机器的存储设备。在 2003 年的时候，Google 内部最大的一个集群利用了数千台机器的数千块硬盘，提供数百 TB 的存储空间，同时向数百台客户机提供服务 [5]。

IO 资源 IO 的速度相比于 CPU 速度和内存速度非常的慢，系统的瓶颈也很容易出现在 IO 上，从这个角度来说，IO 资源在数据中心也是非常珍贵的资源。数据中心中存储设备的 IO 资源稀缺而且使用不均衡。

研究人员记录了 Facebook 内部一个真实运行的集群的存储设备和 CPU 的使用情况（图 3.1）。数据显示，IO 资源的使用出现了很大的不均衡，时高时低，这会导致严重的资源浪费。

3.2 相关工作

3.2.1 网络存储

网络存储是一种特殊的、专用的数据存储服务器。它有直连式存储（DAS）、网络附加存储（NAS）和存储区域网（SAN）三种架构形式。通常，网络存储服务器包含了一个小型的服务器和很多存储设备，它可以直接接入网络，不需要应用服务器的干预，用户可以通过网络直接读取存储服务器上的数据。

直连式存储指的是将存储设备通过 SCSI 接口连接到一台服务器使用。网络附加存储是一种带有小型服务器的存储设备，这个服务器上装配有文件系统，这样的存储设备可以直接接入到普通的网络。存储区域网是指专门为存储搭建的一个独立的网络，专门同于存储通信。

网络存储将数据集中管理，将存储负载从应用服务器上解耦出来，降低了应用服务器的运营成本，也在一定程度上解决了存储容量不足的问题。但这种技术也存在着各种各样的问题。直连式存储仍然存在着存储资源不能在服务器间动态分配的问题。网络附加存储使用普通的网络，其性能受网络影响大。存储区域网的缺点是造价昂贵、成本过高。

3.2.2 分布式文件系统

分布式文件系统通过网络将多台机器的存储设备组织起来，通过文件系统的形式来管理和使用这些存储设备，并向用户提供同一的操作接口。著名的分布式文件系统有 Google 文件系统 [5]、Hadoop[19] 文件系统等等。

分布式文件系统解决了单台服务器的存储容量不足的问题，并向用户提供了方便的操作接口，在底层甚至还提供了容错处理以及一致性保证。但也存在着延迟高、读写性能差的问题。而且它对数据的管理力度过大，很多分布式文件系统的使用场景都是大文件，对于小文件的支持不是很好。并不是所有应用程序都会去使用分布式文件系统，分布式文件系统不能去管理所有已使用和未使用的存储空间，从这个角度来说，它并没有从根本上解决资源使用不均衡的问题。

3.2.3 存储资源分离

存储资源分离指的是将存储资源从计算节点分离出来，应用程序可以任意地访问本地和远端的存储资源。像 Petal[10]、Parallax[21]、Blizzard[13] 这类系统，通过实现了一个虚拟的、分布式的磁盘块仓库将多台机器的存储设备整合，来供文件系统使用，整个过程对应用程序透明，同时还有着很好的性能以及提供容错处理。

存储资源分离技术能提供更细粒度的存储资源管理，对应用程序透明，更具有通用性。既能解决单台服务器上存储资源不足的问题，也可以解决多台服务器上存储资源使用不足的问题。

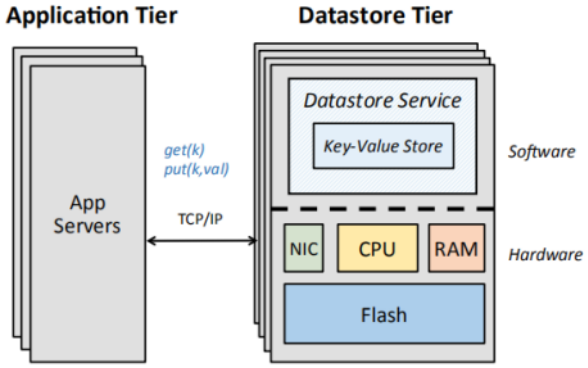


图 3.2: 直连式闪存架构。

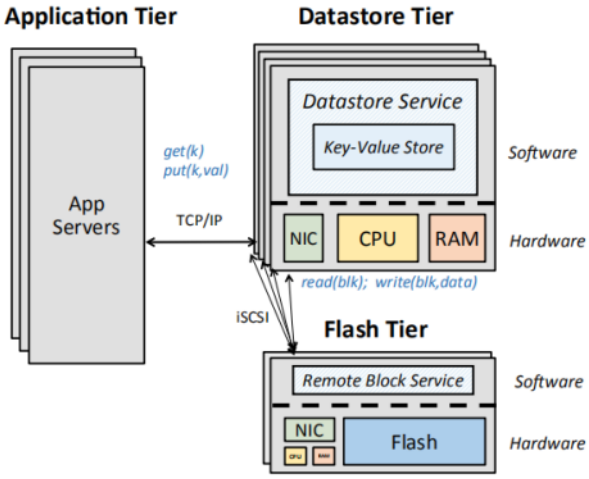


图 3.3: 分离式闪存架构。

3.3 案例分析：闪存资源分离

3.3.1 架构概述

图 3.2展示了传统的闪存使用架构。每台服务器连着直接连着一块闪存设备，而应用程序只能访问到这块直连的闪存设备。

图 3.3是分离式闪存的架构。闪存设备在物理上和数据存储层 (DataStore Tier) 中直接解耦，分成数据存储层和闪存层。在闪存层增加一个设备服务器，用于接收和处理对闪存设备的读写请求。闪存层还有一个协调管理器 (coordination manager)，负责选择合理的存储资源，以达到提高闪存资源利用率的目的。因此在这种架构中，出现了数据存储服务器和闪存设备服务器两种类型的服务器。数据存储服务器管理和使用 CPU 和内存，并跟闪存设备服务器通信，发送对闪存设备的读写请求。通过这种方法，闪存设备和计算节点解耦了，无论闪存服务器是位于本地还是远端，数据存储服务器能够访问任意地去访问。

3.3.2 设计与实现

对于本地的闪存设备，数据存储层依然是通过 SCSI 来访问。对于远端的闪存设备，数据存储层通过 iSCSI(internet small computer system interface) 来访问，因为 iSCSI 能够将对闪存设备的读写操作请求通过网络包的形式发送。下面我们来介绍对远端闪存设备读操作和写操作的详细流程。

读操作流程

iSCSI 的 initiator 接收到来自闪存层的包后，将使用 DMA(Direct Memory Access) 技术将包的数据从 NIC 模块转移至内核内存。内核解析获取的包，拆离出数据内容，将之放在 iSCSI 的 PDU。iSCSI 层将 PDU 数据复制到 SCSI 缓存。此时应用就可以从 SCSI 缓存中直接获取数据。

写操作流程

iSCSI 层的 initiator 能控制传输数据的时间，数据存储层维护一个指向协议数据单元 (Protocol Data Unit, PDU) 的指针。iSCSI 层对要写的数据进行包装（加入信息头等），然后由内核将 PDU 传输给闪存层。其中包括的 NIC 和 SCSI 的缓存读写则取决与 TCP/IP 栈的实现。

3.3.3 性能评估

延迟分析

数据存储层使用 RocksDB 键值存储，利用 mutilate 负载生成器 [11] 生成大量用户进程，进行大量的查询操作，营造期望的每秒查询率 (Queries Per Second, QPS)。图 3.4展示了在无负载系统下，本地闪存与远程闪存读取数据的用户延迟的累积分布函数 (Cumulative Distribution Function)。远程闪存的用户延迟显然高于本地闪存，在曲线的 95% 处，远程闪存读取延迟约比本地闪存高 260 微秒。由于延迟 SLA 在 5 到 10 毫秒左右，远远高于微秒数量级，所以远程闪存的延迟代价在可接受范围内。

吞吐量分析

为测试本地闪存和远程闪存的吞吐量指标，控制应用层的每秒请求率 (QPS)，并描绘点（QPS，延迟），图 3.5展示其吞吐量性能。由图分析，在相同延迟情况下，远程闪存吞吐量约为本地闪存的 80%。表观上，远程闪存的吞吐量受到了较严重的影响。一方面，这是存储设备性能和资源利用率等的权衡，远程闪存的理念是牺牲了部分性能，换取更好的资源利用率。另一方面，可以通过扩展数据存储层的 CPU 来补偿这些吞吐量损失。

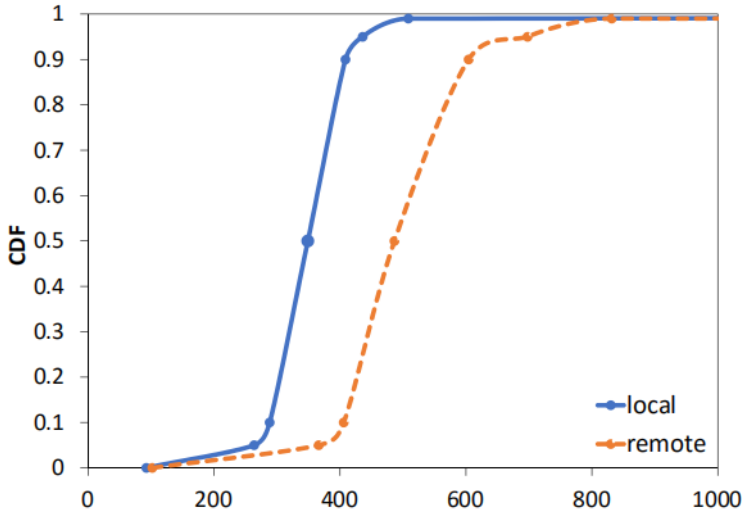


图 3.4: 读操作延迟的 CDF 曲线。

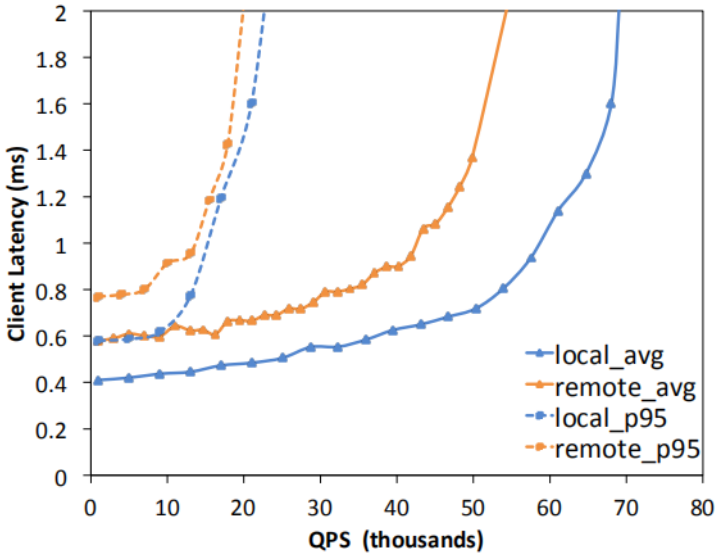


图 3.5: 单 SSDB 服务器的延迟-QPS 曲线。

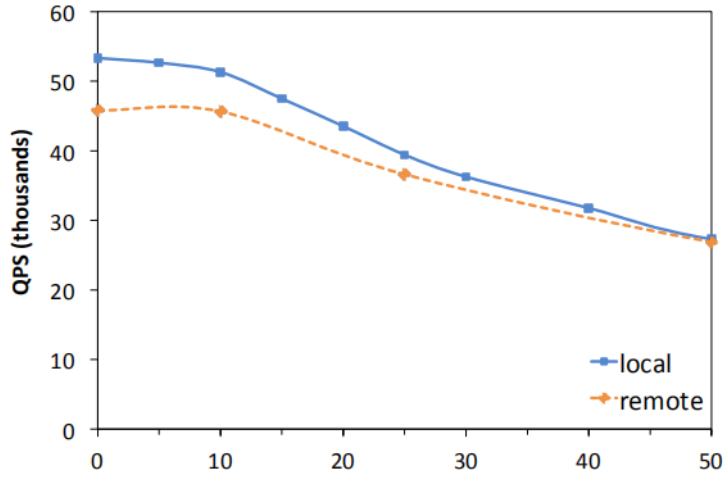


图 3.6: QPS 随 CPU 负载变化曲线

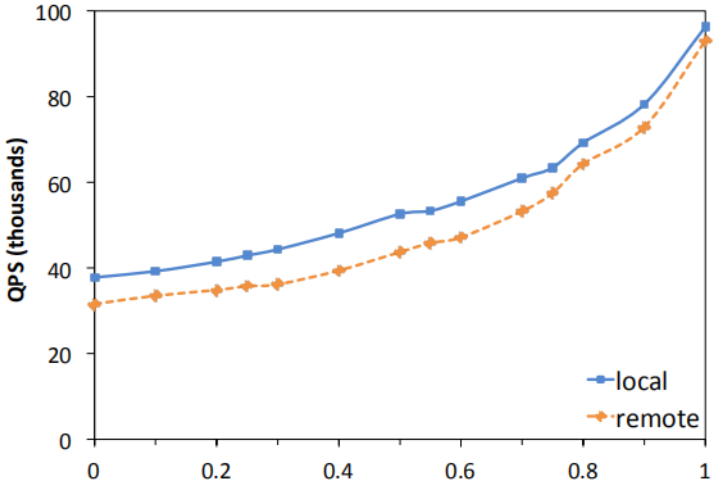


图 3.7: QPS 随读操作占比变化曲线。

敏感度分析

通过在 SSDB 中增加循环进程，创造不同 CPU 负载的环境，测试远程闪存的 QPS 对不同参数的曲线变化率（敏感度）。图 3.6展示了本地闪存和远程闪存在不同 CPU 负载下的 QPS。二者在 CPU 负载较低的环境下，本地闪存较远程闪存有更高的 QPS，意味着远程闪存的性能受到一定影响，而在 CPU 处于较高负载的环境下，本地闪存与远程闪存的 QPS 相差无几，意味着此时主要瓶颈不在 iSCSI 通讯上。同时，本地闪存的 QPS 受 CPU 负载大小的影响比远程闪存大，也就是说，远程闪存的性能表现更加稳定。

图 3.7展示了在相同请求数量下，本地闪存和远程闪存的 QPS 随写请求占总请求的比例变化曲线。二者的 QPS 都随着写请求比例增加而增加，因为写操作的返回是异步的，其效率比读操作高。二者的 QPS 差值在不同的写操作比例下差别不大，可以得出结论，读写操作所占比例对本地闪存和远程闪存性能差距影响不大。

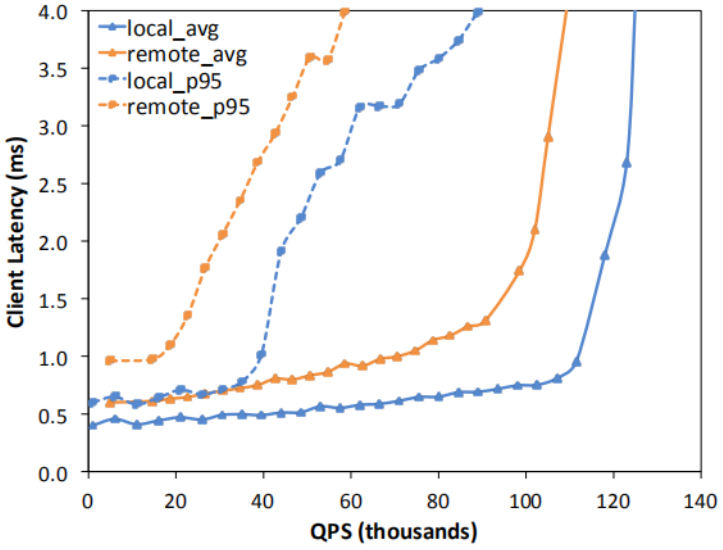


图 3.8: 2 个 SSDB 服务共享闪存下，延迟-QPS 曲线。

多服务端性能分析

实际情况中，机群中服务器主机与闪存时多对多的关系，所以分析多服务端共享同闪存集群的性能很有实际价值。图 3.8和图 3.9分别展示两个 SSDB 服务器主机和三个 SSDB 服务器主机时，本地闪存与远程闪存的延迟-QPS 曲线图。毫无疑问，本地闪存的 QPS 性能始终比远程闪存更高。值得一提的是，三个 SSDB 环境相较于两个 SSDB 环境下，远程闪存的 QPS 性能损失代价更严重。

3.3.4 存储分离适用情境分析

通过计算直连式闪存 (Direct-attached flash) 和远程闪存 (disaggregation flash) 的主要消耗 C_{direct} 和 C_{disagg}，对存储分离进行优势评估。直连式闪存的消耗包括闪存读写的消耗和数据存储端 CPU、RAM、NIC 的消耗两部分，远程闪存则包括闪存读写的消耗，数据存储层 CPU、RAM、NIC 的消耗和闪存层 CPU、RAM、NIC 的消耗。这些消耗的权重为目标数据与实际数据的比值，如目标 QPS 与实际 QPS 的比值。计算而得的消耗越大，意味着其资源利用越差，反之则越好。引入新的评估指标，消耗节约比例。消耗节约比例含义为使用远程闪存比直连式闪存节约的消耗比例。若消耗节约比例值为正，则表示该环境参数下远程闪存消耗比直连式闪存小；反之则远程闪存消耗更大。

图 3.10展示了在计算强度比例因子和存储空间比例因子分布空间下，相应的消耗节约比例。由图例的对称性可以分析而得，闪存读写的消耗与数据存储层的 CPU、RAM、NIC 消耗大致相等。在不同的情景假设下（对应图中不同的二维坐标），对资源的分配权衡也是不一样的。若闪存资源的价值比 CPU、内存低，那么服务器将倾向高闪存空间和低计算强度，对应图例的右下半边区域，该区域的消耗节约比例大于零，意味着远程闪存消耗比直连式闪存低，则此情境下，服务器主机更适合于远程闪

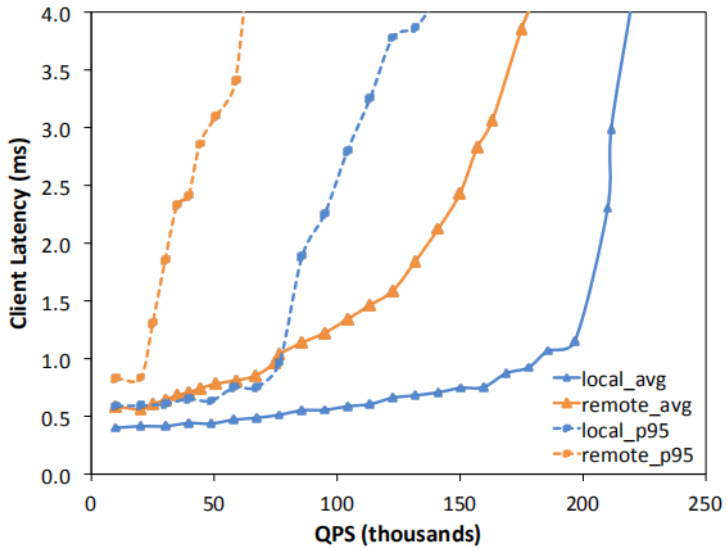


图 3.9: 3 个 SSDB 服务共享闪存下，延迟-QPS 曲线。

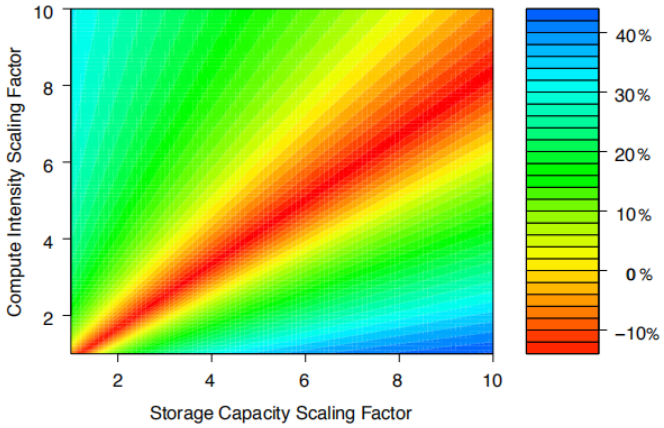


图 3.10: 消耗节约比例-计算强度比例因子-存储空间比例因子分布图

存。若闪存资源的价值比 CPU、内存高，那么服务器将倾向低闪存空间和高计算强度，对应图例的坐上半边区域，同理，该区域的消耗节约比例大于零，远程闪存消耗比直连式闪存低，服务器主机更适合于远程闪存。若闪存资源与 CPU、内存资源价值相当，对应图例的对角线区域，该区域的消耗节约比例小于零，也就是远程闪存消耗比直连式闪存高，此时服务器主机更适合直连式闪存。

3.3.5 案例小结

Klimovic 等人 [9] 对闪存分离策略做了充分的研究，分析了闪存分离带来的容量和 IO 利用率优势，也客观展示了资源分离带来的性能损失，并提出了弥补性能损失的方法。在考虑服务器主机整体架构时，不能盲目地强调闪存分离的优势，忽略其带来的性能损失和维护成本。更加客观的策略是，通过各类资源（如 CPU, 内存，闪存）的当前价值来定量计算分离式闪存带来的提升及损失孰重孰轻，进而合理取舍。

3.4 本章小结

本节我们探讨了存储资源的背景，介绍了存储设备和存储技术发展的现状，并以 Klimovic 等人 [9] 的闪存分离研究工作为切入点，分析了闪存分离的优势及不足，对其作出客观评价。

总结来说，存储资源分离能够实现高程度的存储设备解耦，使得资源管理更加高效便捷，能够大幅提高存储资源利用率，在一定程度上能避免存储资源浪费。但存储资源分离使得存储框架复杂化，带来额外开发维护成本，且利用网络作为数据传输介质，无疑使得性能也有相当的损耗。目前大多数解决方案关注点在于架构的设计，力求存储资源分配更加合理化，实际上，减小数据传输带来的性能损耗也是一大研究方向。利用良好的数据缓存等机制，从根本上减少通过网络传输数据的频率，或许能成为降低性能损耗的良策。

Chapter 4

综合分离各类资源

数据中心的规模日益扩张，对于内存的需求和存储的需求也越来越大。但是现阶段，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）。在前面的两个章节中，我们讨论了内存分离和存储分离，这些技术能够部分提高数据中心对硬件资源的利用率，并且已经在现有的商业环境中应用。然而，这些技术仍然要部署到 monolithic server 上，无法解决 monolithic serve 本身的一些缺点，如缺乏弹性、单一硬件故障导致整台 server 失败等。

如果把所有的的硬件设备都组织为独立的组件，依靠网络进行通信，就可以完全打破 monolithic server 的体系结构，真正意义上实现硬件资源的分离，并且完全解决 monolithic serve 对弹性、异构性等支持不好的缺点。本章后面将以 LegoOS 为例，介绍这一思想。

4.1 背景介绍

4.1.1 硬件的发展

近年来，网络的速度越来越快，甚至已接近内存总线速度的数量级。基于快速网络，分离的设备之间能够进行快速的通信，这使得硬件资源分离的思路成为可行。另一方面，硬件的功能和处理能力越发丰富。现代硬件可以整合大量以前由软件实现的逻辑，如控制器、网络协议栈等，这使得分离的硬件组件能够高效的进行本地管理，而不依赖 CPU 资源和复杂的软件逻辑。

4.1.2 单一硬件资源的分离

目前已有很多针对单一硬件资源分离的研究。例如，针对内存资源，有基于 RDMA 的远程换页系统 INFINISWAP[6]，可以透明的把本地计算机的内存交换到集群中另一台计算机的内存上；针对存储资源，有 Flash storage disaggregation[9]，把闪存层从数据存储层解耦出来集中管理，使得网络中的主机可以通过统一的接口读写闪存设备。

4.1.3 系统层面的分离

现有的操作系统大多以单台机器为管理单元，并假设所有硬件资源都位于本地。这种体系结构从根本上不利于硬件资源的分离。比较好的方法是从底层按照分离的思路重新设计操作系统，这样不仅简化了抽象层，还可以打破单机操作系统管理分布式硬件的语义隔离，能更好地适应分离的硬件结构。但是，重新设计一套系统架构，并且能够较好的对分布式资源进行管理，是非常困难的。不过，目前已存在一些尝试，如 Helios[15] 在 NUMA 架构的异构核心上构建了一组 satellite kernel。

4.2 相关研究

4.2.1 多机分布式计算框架

随着需要分析处理的数据量不断增大，单台机器无法完成海量数据的计算和存储。分布式计算在此背景下产生，其主要思想是把数据和计算任务分配到多台机器上，每台机器只需承担很少的任务。这其中涉及到集群管理、消息通信、任务分发等复杂的逻辑。一些分布式计算框架，如 Hadoop、Spark 等的出现简化了用户的使用，用户只需编写自己的业务逻辑，底层全部交给框架处理。

这带来了很多好处，如资源被分摊到了每台机器上，单台机器的故障不会使任务彻底中断；一个良好的调度策略能够根据每台机器的性能和负载合理分配任务，尽可能最大化利用每台机器的资源；集群中的机器可以根据需要随时添加或移除；等。但是，分布式计算集群的调度单位是一整台机器，粒度很粗，无法解决单台机器内部的问题，如单台机器各硬件负载不均衡导致资源浪费。

4.2.2 巨型虚拟机

针对多机的资源利用，还有一种分布式操作系统的方案。其原理把每台机器的硬件资源综合在一起（例如，综合所有机器实现分布式共享内存），对外则表现为一个巨型的虚拟机。结合调度，单一的任务也能够利用到不同机器的资源，提高了硬件资源的整体利用率。但是，这种设计对节点间的通信要求较高，同时调度的策略和性能也会带来很大影响。

Amoeba[20] 是一个例子，它在每台机器的每个处理器核上运行一个 microkernel，实现了一个共享的处理器池，为每个用户动态分配处理器资源。相比于多机分布式计算，Amoeba 使资源调度的粒度减小到了进程的级别。

4.2.3 弹性计算

现在，很多云服务商都提供弹性计算服务。用户可以根据自己对资源的使用需求，灵活配置和购买。云服务商在后台管理着大量机器，将物理资源（如 CPU、内存、存储、带宽等）虚拟化，并在多台机器上打通。用户对资源的使用常常是突发的，这样大多数时间物理资源是空闲的，服务商可以暂时把资源分配给其他用户。依靠资源的虚拟化和分解，实现了所有资源的按需分配，使硬件资源得到了尽可能高的利用效率。

4.3 案例分析：LegoOS

4.3.1 设计目标

多年来，monolithic server 一直是数据中心的部署和操作单元，但这种以单台服务器未中心的架构有几个重要的问题：

- 资源利用率低：由于对于同一个任务必须从同一台机器分配 CPU 和内存，而程序对资源的利用是不平衡的，导致硬件利用率整体不高。
- 硬件弹性差：一旦完成单台服务器的组装，就很难添加或删除硬件组件。
- 故障范围大：单台服务器由很多硬件组件组成，任何一个组件的损坏都会使得服务器无法工作。
- 对异构硬件支持不好：组成单台服务器的硬件之间往往紧密的耦合，一些专用的、非传统的硬件往往难以和其他硬件组件结合。

针对以上问题，提出了一种新的硬件资源分解的架构，能够使各硬件组件完全拆分开，并在此基础上构建了 LegoOS 系统。

4.3.2 整体架构

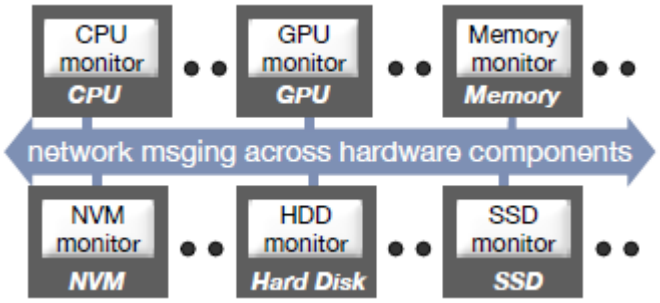


图 4.1: Splitkernel 架构

Splitkernel 模型分解了传统操作系统的功能，并将这些功能转入松散耦合的硬件组件的监视器中。每个监视器在本地运行，管理着自己的硬件组件，并仅在需要时通过网络以显式的消息传递与其他组件通信。

LegoOS 是基于 splitkernel 模型构建的一个专为硬件资源分解的操作系统。它将操作系统的功能分为了三类监视器：处理器监视器、内存监视器和存储监视器。除了处理器，每个硬件组件期望有一个硬件控制器，可以执行监视器的逻辑。LegoOS 有全局的资源管理器 GPM、GMM 和 GSM，用来粗粒度的分配处理器、内存和存储资源。细粒度的分配由各自的监视器决定。

4.3.3 具体实现

LegoOS 在设计上针对三种硬件组件：处理器、内存、存储，分别称之为 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。

处理器管理

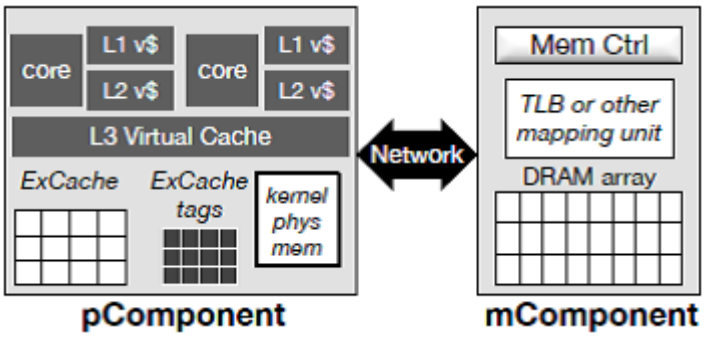


图 4.2: LegoOS 的 pComponent 和 mComponent 架构

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。当用户启动一个新进程时，由全局的 GPM 选择一个托管线程数最少的 pComponent，然后这个 pComponent 为用户线程分配 CPU 核心，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，因为网络延迟在 LegoOS 中很低）。因为全局 GPM 的存在，调度策略的着重于最小化上下文切换的开销，而不是单个线程的核心利用率。

pComponent 完全不需要地址映射，mmu 和 TLB 等全部由 mComponent 维护。为了性能，在本地 pComponent 仍持有少量的（如 4GB）内存缓存，称为 ExCache，位于处理器的 Last-Level Cache（LLC）之下，而剩余的大量内存通过网络从 mComponent 访问。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。

ExCache 冷不命中会产生很多访问 mComponent 的请求，带来了较大的性能开销。为此有一个简单的优化。注意到匿名内存的初始内容为零，因此可以直接在 ExCache 中分配这些 line，直到这个 ExCache line 被 flush，才首次发送到 mComponent 中。在设计上 LegoOS 的 pComponent 不共享内存，因此不会产生竞争。

内存管理

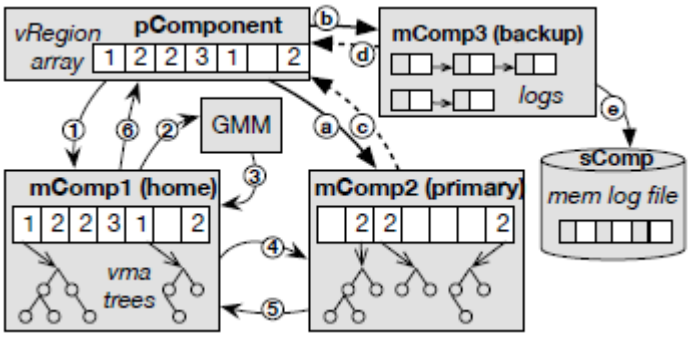


图 4.3: LegoOS 的分布式内存管理

mComponent 管理虚拟内存和物理内存，包括它们的分配、回收和映射。为了最小化网络通信，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。一个用户进程的虚拟地址空间可以跨越多个 mComponent。用户进程还有一个 home mComponent，它只是记录进程的每个 vRegion 分别由哪个 mComponnet 在管理（vRegion array），以及每个 vRegion 内的剩余未分配空间。

当用户程序申请虚拟内存空间时，pComponent 把相关的请求转发给进程的 home mComponent。home mComponent 会查询 vRegion array，寻找一个合适的 vRegion，然后把请求再转发给对应的 mComponent，由它来为用户分配 vma。如果可用虚拟内存空间不足，home mComponent 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。（实现上为了快速访问，pComponent 也缓存了一份 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：使用 primary mComponent 和 backup mComponent。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。

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。

4.3.4 测试评估

由于并没有真正的资源分解硬件，硬件组件实际由服务器模拟而来，通过限制可用的资源使其符合各个 componnet 的设定。

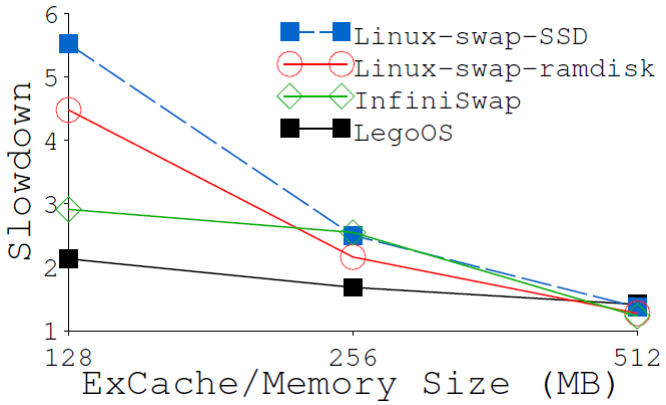


图 4.4: TensorFlow 性能

	Processor	Disk	Memory	NIC	Power	Other	Monolithic	LegoOS
MTTF (year)	204.3	33.1	289.9	538.8	100.5	27.4	5.8	6.8 - 8.7

图 4.5: MTTF 分析

- TensorFlow：在 LegoOS 上运行一个未修改的 TesorFlow 程序，其 working set 是 0.9GB，为它分配了 1 个 pComponent、1 个 mComponent、1 个 sComponent。测试的基准是一个无限内存的 Linux 系统。在有限的内存下，LegoOS 的 Slowdown 显著低于其他的 swap 方案的 OS。
- 故障分析：通过收集到的各硬件的平均无故障运行时间（MTTF），可以分别计算出 monolithic server 和 LegoOS 的 MTTF。由于 LegoOS 的组件完全分离，并且资源利用率接近 100%（传统的 monolithic server 集群的资源利用率只有大约 50%），使得 MTTF 提高了 17%-49%。

4.4 本章小结

本章主要介绍了一些综合分离各类资源的方案，如传统的多机分布式计算、巨型虚拟机，以及在云服务商大规模应用的弹性计算。这些方案从不同的技术原理、以不同的粒度实现了一定的资源分离，提高了一定的资源利用率。硬件的发展，为软件的实现提供了新的可能，使得可以跳出现有框架的限制，去尝试一些颠覆性思路。

最后讨论了 LegoOS，以一种全新的架构完全分离了各个硬件组件，从根本上解决了 monolithic server 的缺陷。

Chapter 5

总结

章节1从计算力发展与计算需求变化的角度，阐述了大规模计算对硬件资源利用率提出的新要求，介绍了硬件资源分离这一研究方向的背景。

章节2对内存资源分离技术进行了介绍。我们介绍了内存资源分离的问题现状与提高内存资源利用率的常见分离技术设计与实现，并以 INFINITESWAP 为案例，详细介绍了在高速网络下，内存资源分离的挑战、解决方案与性能评估。

章节3对存储资源分离技术进行了介绍。我们介绍了存储技术的发展，在当前面临的挑战，介绍并对比了目前常见的解决方案。以当前普遍使用的闪存介质为例，我们详细介绍了其中一种解决方案的设计挑战、设计细节与性能评估。

章节4对系统性的硬件资源分离技术进行了介绍，本节不再关注特定一种资源，而是如何在统一的框架下将各类资源进行分离，从而提高整体的利用率。我们介绍了多种实现这一目标的技术与系统，并以 LegoOS 为案例，详细介绍分析了其设计原则与实现细节。

硬件资源分离是未来趋势。目前，处理器的摩尔定律正面临失效的挑战，而随着工业物联网、区块链技术与人工智能的普及，逐渐增加并复杂化的计算需求却对计算资源提出越来越高的要求。另一方面，大规模数据中心中存在的资源利用率不理想问题与扩展性挑战却使得计算能力并不能有效地转化为完成的计算任务。为解决这一矛盾，最直接有效地方法便是将硬件资源结偶，不局限与单台机器，允许其被动态地分配给不同的计算任务，及所谓的硬件资源分离。

螺旋上升的发展。最初的一体机设计，所有资源归属于单一使用者，由其统一调度管理其资源的使用，简化了资源调度的同时，带来了巨大的扩展性挑战。单一资源进行分离，让资源不再局限在本地，解决了扩展性的问题，却使分布式系统的设计与实现变得复杂与难以管理。系统性的硬件资源分离，则试图解决但一资源分离带来的这一弊端，并同时保留资源分离对利用率的提升。我们可以看出在此发展脉络中追求，即最大化利用资源，是不变的，但在不同的背景下有着不同的实践方式。在将来，新的需求的提出，极可能再次打破现有的设计范式，提出出新的技术方向。

单纯的将硬件资源分离管理并不足够。随着硬件资源间的结偶与新硬件技术的发展，以往的假设与顾虑往往不复存在，我们需要针对性的进行分析与重新设计，才能真正地发掘出硬件资源分离的潜力。同时，为了将资源发挥到极致，在分离基础上，我们也应该进一步考虑如何设计方案，使上层软件能够更好地利用新的硬件资源模型。

我们相信并期待硬件资源分离技术被广泛使用在大型数据中心，产生重要的影响。

参考文献

[1] John B Carter, John K Bennett, and Willy Zwaenepoel. *Implementation and performance of Munin*. Vol. 25. 5. ACM, 1991.

[2] Haogang Chen et al. "A transparent remote paging model for virtual machines". In: *International Workshop on Virtualization Technology*. 2008.

[3] Nimbus Data. *Nimbus Data Launches the World's Largest Solid State Drive – 100 Terabytes – to Power Data-driven Innovation*. https://nimbusdata.com/press/nimbus-data-launches-worlds-largest-solid-state-drive-100-terabytes-power-data-driven-innovation/.

[4] Michail D Flouris and Evangelos P Markatos. "The network RamDisk: Using remote memory on heterogeneous NOWs". In: *Cluster Computing* 2.4 (1999), pp. 281–293.

[5] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google File System". In: *Proceedings of the 19th ACM Symposium on Operating Systems Principles*. Bolton Landing, NY, 2003, pp. 20–43.

[6] Juncheng Gu et al. "Efficient Memory Disaggregation with Infiniswap." In: *NSDI*. 2017, pp. 649–667.

[7] Intel. *9th Generation Intel Core Desktop Processors*. https://www3.intel.com/content/dam/www/public/us/en/documents/product-briefs/9th-gen-core-desktop-brief.pdf.

[8] Intel. *INTEL OPTANE SSD 905P SERIES*. https://www.intel.cn/content/www/cn/zh/products/memory-storage/solid-state-drives/gaming-enthusiast-ssds/optane-905p-series/905p-480gb-2-5-inch-20nm.html.

[9] Ana Klimovic et al. "Flash storage disaggregation". In: *Proceedings of the Eleventh European Conference on Computer Systems*. ACM. 2016, p. 29.

[10] Edward K Lee and Chandramohan A Thekkath. "Petal: Distributed virtual disks". In: *ACM SIGPLAN Notices*. Vol. 31. 9. ACM. 1996, pp. 84–92.

[11] Jacob Leverich. *Mutilate: high-performance memcached load generator*. 2014.

[12] Kai Li and Paul Hudak. "Memory coherence in shared virtual memory systems". In: *ACM Transactions on Computer Systems (TOCS)* 7.4 (1989), pp. 321–359.

[13] James W Mickens et al. “Blizzard: Fast, Cloud-scale Block Storage for Cloud-oblivious Applications.” In: *NSDI*. 2014, pp. 257–273.

[14] Tia Newhall et al. “Nswap: A network swapping module for linux clusters”. In: *European Conference on Parallel Processing*. Springer. 2003, pp. 1160–1169.

[15] Edmund B Nightingale et al. “Helios: heterogeneous multiprocessing with satellite kernels”. In: *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM. 2009, pp. 221–234.

[16] Bill Nitzberg and Virginia Lo. “Distributed shared memory: A survey of issues and algorithms”. In: *Computer* 24.8 (1991), pp. 52–60.

[17] Yizhou Shan et al. “LegoOS: A Disseminated, Distributed {OS} for Hardware Resource Disaggregation”. In: *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 2018, pp. 69–87.

[18] Jiaxin Shi et al. “Fast and Concurrent RDF Queries with RDMA-Based Distributed Graph Exploration.” In: *OSDI*. Vol. 16. 2016, pp. 317–332.

[19] Konstantin Shvachko et al. “The hadoop distributed file system”. In: *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*. Ieee. 2010, pp. 1–10.

[20] Andrew S Tanenbaum et al. “The Amoeba distributed operating system-a status report”. In: (1991).

[21] Andrew Warfield et al. “Parallax: Managing Storage for a Million Machines.” In: *HotOS*. 2005.

[22] Xingda Wei et al. “Fast in-memory transaction processing using RDMA and HTM”. In: *Proceedings of the 25th Symposium on Operating Systems Principles*. ACM. 2015, pp. 87–104.

[23] Wikipedia. *Non-volatile memory*. https://en.wikipedia.org/wiki/Non-volatile_memory.

[24] Qi Zhang, Lu Cheng, and Raouf Boutaba. “Cloud computing: state-of-the-art and research challenges”. In: *Journal of internet services and applications* 1.1 (2010), pp. 7–18.