中数国科超融合
技术白皮书

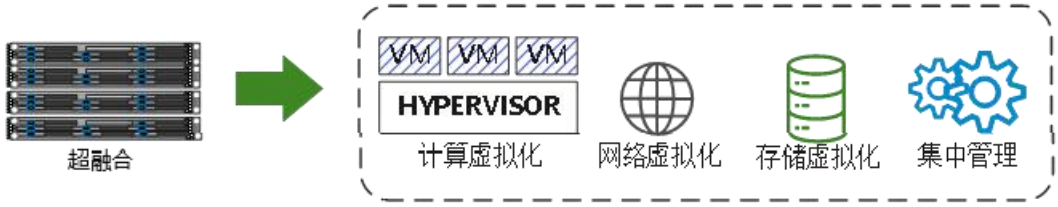
2023 年 8 月

目 录

1. 超融合概述

超融合产品是面向基础架构即服务（IaaS）推出的云数据中心基础架构解决方案。该方案集成了计算、存储、网络、安全、运维监控、云平台六大软件功能，将物理硬件、计算存储网络虚拟化资源进行统一监控和管理，可快速灵活部署业务，通过统一的管理平台便捷定义所有关键数据中心功能，实现统一的软件定义数据中心资源池，为客户提供私有云数据中心提供一站式解决方案。

超融合通过在服务器上安装超融合软件，对外提供统一的存储资源池和计算资源池。存储资源池采用分布式架构，支持完善的存储特性和弹性扩容能力；计算资源池支持 KVM 虚拟机以及 LXC 容器，并支持动态的工作负载分配。



其中超融合软件不仅支持通用的 x86 CPU 平台，也支持飞腾、龙芯等主流国产 CPU 平台。内部包含虚拟计算、虚拟存储和虚拟网络模块，通过将计算、存储、网络等资源进行虚拟化和池化，构建数据中心里所需的最小资源单元，同时通过支持对计算、存储、网络资源的管理调度功能以及基础架构运维管理、监控管理、资源管理、安全管理、访问控制管理等运维管理功能，实现对数据中心各类资源单元的统一调度分配与管理，最终满足用户构建数据中心“快速部署、弹性扩展、简便管理”等核心诉求。其软件架构如下：

2. 超融合关键模块介绍

2.1. 虚拟计算

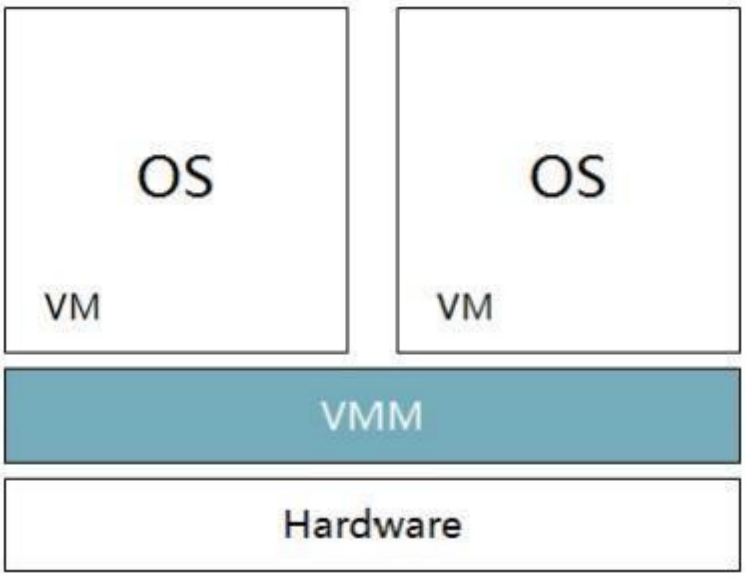
虚拟计算就是将通用服务器经过虚拟化软件，对最终用户呈现标准的虚拟机。虚拟机是由虚拟化层提供的高效、独立的虚拟计算机系统，每台虚拟机都是一个完整的系统，它具有处理器、内存、网络设备、存储设备和 BIOS，因此操作系统和应用程序在虚拟机中的运行方式与它们在物理服务器上的运行方式没有什么区别。

HCI 超融合软件通过将服务器资源虚拟化为多台虚拟机。最终用户可以在这些虚拟机上安装各种软件，挂载磁盘，调整配置，调整网络，就像普通的服务器一样使用它。

虚拟计算是超融合架构中必不可少的关键因素。对于最终用户，虚拟机比物理机的优势在于它可以很快速的发放启用，很方便的调整配置和组网，从而实现计算资源的快速部署和弹性扩展，并很容易实现备份与还原，可便捷地将整个系统（包括虚拟硬件、操作系统和配置好的应用程序）在不同的物理服务器之间进行迁移，有效增强系统可靠性。对于维护人员来讲，虚拟机复用了硬件，这样硬件更少加上云平台的自动维护能力，大大简化维护管理复杂度，同时有效降低整个 IT 系统的建设和运维成本。

2.1.1. 虚拟计算技术原理

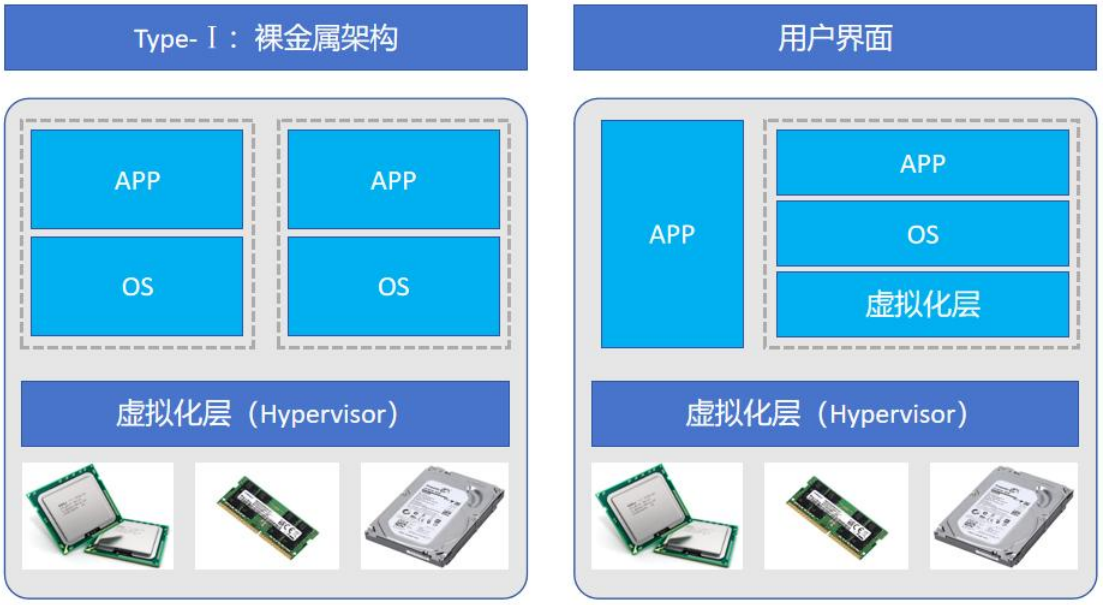
（一）Hypervisor 架构选型



Hypervisor 是一种运行在物理服务器和操作系统之间的中间软件层,可允许

多个操作系统和应用共享一套基础物理硬件，因此也可以看作是虚拟环境中的“元”操作系统，它可以协调访问服务器上的所有物理设备和虚拟机，也叫虚拟机监视器（Virtual Machine Monitor）。

Hypervisor 是所有虚拟化技术的核心。非中断地支持多工作负载迁移的能力是 Hypervisor 的基本功能。当服务器启动并执行 Hypervisor 时，它会给每一台虚拟机分配适量的内存、CPU、网络和磁盘，并加载所有虚拟机的客户操作系统。



Hypervisor，常见的 Hypervisor 分两类：

- **Type-I（裸金属型）**

指 VMM 直接运作在裸机上,使用和管理底层的硬件资源，GuestOS 对真实硬件资源的访问都要通过 VMM 来完成，作为底层硬件的直接操作者，VMM 拥有硬件的驱动程序。裸金属虚拟化中 Hypervisor 直接管理调用硬件资源，不需要底层操作系统，也可以理解为 Hypervisor 被做成了一个很薄的操作系统。这种方案的性能处于主机虚拟化与操作系统虚拟化之间。代表是 VMware ESX Server、Citrix XenServer 和 Microsoft Hyper-V，Linux KVM。

- **Type-II 型（宿主型）**

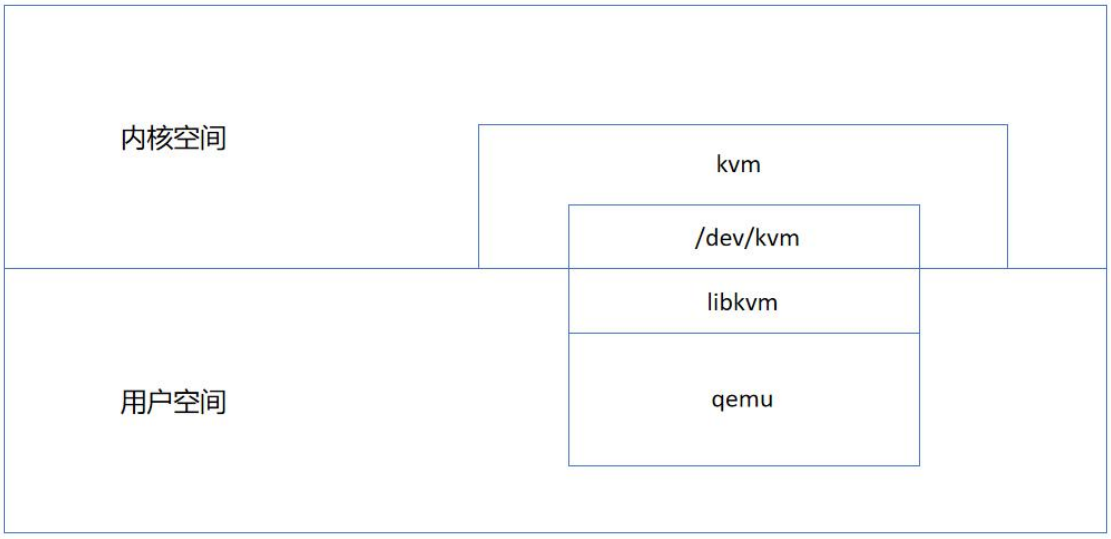
指 VMM 之下还有一层宿主操作系统，由于 Guest OS 对硬件的访问必须经过宿主操作系统，因而带来了额外的性能开销，但可充分利用宿主操作系统提供的设备驱动和底层服务来进行内存管理、进程调度和资源管理等。主机虚拟化中 VM 的应用程序调用硬件资源时需要经过:VM 内核->Hypervisor->主机内核，导致性能是三种虚拟化技术中最差的。主机虚拟化技术代表是 VMware Server （GSX）、

Workstation 和 Microsoft Virtual PC、Virtual Server 等。

由于宿主型 Hypervisor 的效率问题，采用了裸金属型 Hypervisor 中的 Linux KVM 虚拟化。

KVM(Kenerl-based Virtual Machine)是基 linux 内核虚拟化技术，自 linux2.6.20 之后就集成在 linux 的各个主要发行版本中。它使用 linux 自身的调度器进行管理，所以相对于 xen，其核心源码很少。

KVM 是基于硬件虚拟化扩展（Intel VT- X）和 QEMU 的修改版，KVM 属于 Linux kernel 的一个模块，可以用命令 modprobe 去加载 KVM 模块。加载了该模块后，才能进一步通过工具创建虚拟机。但是仅有 KVM 模块是不够的。因为用户无法直接控制内核去做事情，还必须有一个运行在用户空间的工具才行。这个用户空间的工具，我们选择了已经成型的开源虚拟化软件 QEMU，QEMU 也是一个虚拟化软件，它的特点是可虚拟不同的 CPU，比如说在 x86 的 CPU 上可虚拟一个 power 的 CPU，并可利用它编译出可运行在 power 上的 CPU，并可利用它编译出可运行在 power 上的程序。KVM 使用了 QEMU 的一部分，并稍加改造，就成了可控制 KVM 的用户空间工具了。这就是 KVM 和 QEMU 的关系。



一个普通的 linux 进程有两种运行模式：内核和用户。而 KVM 增加了第三种模式：客户模式（有自己的内核和用户模式）。在 kvm 模型中，每一个虚拟机都是由 linux 调度程序管理的标准进程。

总体来说，kvm 由两个部分组成：一个是管理虚拟硬件的设备驱动，该驱动使用字符设备/dev/kvm 作为管理接口；另一个是模拟 PC 硬件的用户空间组件，这是一个稍作修改的 qemu 进程。

（二）Hypervisor 实现

VMM (Virtual Machine Monitor)对物理资源的虚拟可以划分为三个部分：CPU 虚拟化、内存虚拟化和 I/O 设备虚拟化,其中以 CPU 的虚拟化最为关键。

经典的虚拟化方法：现代计算机体系结构一般至少有两个特权级（即用户态和核心态，x86 有四个特权级 Ring0~ Ring3）用来分隔系统软件和应用软件。那些只能在处理器的最高特权级（内核态）执行的指令称之为特权指令，一般可读写系统关键资源的指令（即敏感指令）决大多数都是特权指令（X86 存在若干敏感指令是非特权指令的情况）。如果执行特权指令时处理器的状态不在内核态，通常会引发一个异常而交由系统软件来处理这个非法访问（陷入）。

经典的虚拟化方法就是使用“特权解除”和“陷入-模拟”的方式， 即将 GuestOS 运行在非特权级，而将 VMM 运行于最高特权级（完全控制系统资源）。解除了 GuestOS 的特权级后，Guest OS 的大部分指令仍可以在硬件上直接运行，只有执行到特权指令时，才会陷入到 VMM 模拟执行（陷入-模拟）。“陷入-模拟” 的本质是保证可能影响 VMM 正确运行的指令由 VMM 模拟执行，大部分的非敏感指令还是照常运行。

因为 X86 指令集中有若干条指令是需要被 VMM 捕获的敏感指令，但是却不是特权指令（称为临界指令），因此“特权解除”并不能导致他们发生陷入模拟，执行它们不会发生自动的“陷入”而被 VMM 捕获，从而阻碍了指令的虚拟化，这也称之为 X86 的虚拟化漏洞。

X86 架构虚拟化的实现方式可分为：

1）X86“全虚拟化”（指所抽象的 VM 具有完全的物理机特性，OS 在其上运行不需要任何修改）Full 派秉承无需修改直接运行的理念，对“运行时监测，捕捉后模拟”的过程进行优化。该派内部之实现又有些差别，其中以 VMWare 为代表的基于二进制翻译 (BT) 的全虚拟化为代表，其主要思想是在执行时将 VM 上执行的 Guest OS 指令，翻译成 x86 指令集的一个子集，其中的敏感指令被替换成陷入指令。翻译过程与指令执行交叉进行，不含敏感指令的用户态程序可以不经翻译直接执行。

2）X86“半虚拟化”（指需 OS 协助的虚拟化，在其上运行的 OS 需要修改），半虚拟化的基本思想是通过修改 Guest OS 的代码，将含有敏感指令的操作，替换为对 VMM 的超调用 Hypercall，类似 OS 的系统调用，将控制权转移到 VMM，

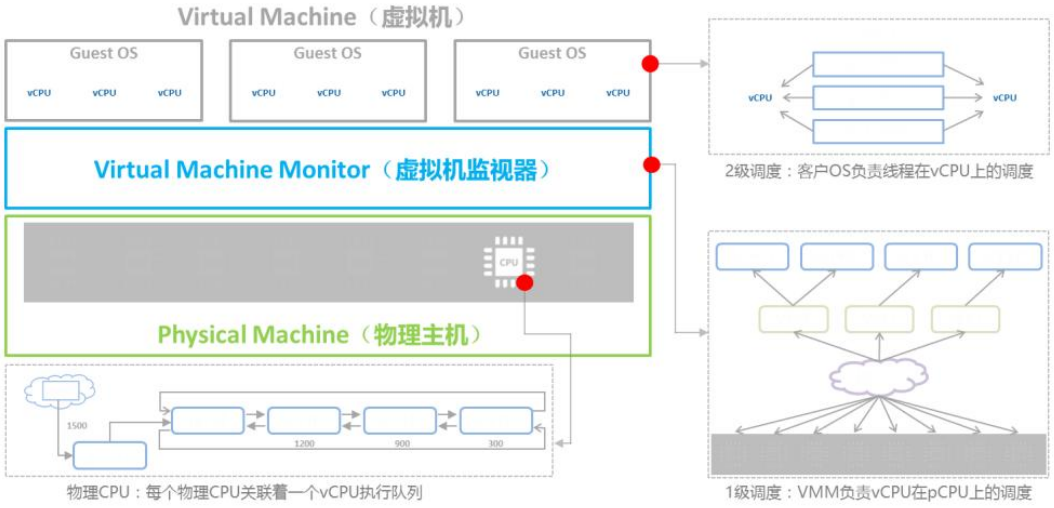
该技术因 VMM 项目而广为人知。该技术的优势在于 VM 的性能能接近于物理机，缺点在于需要修改 GuestOS（如：Windows 不支持修改）及增加的维护成本，关键修改 Guest OS 会导致操作系统对特定 hypervisor 的依赖性，因此很多虚拟化厂商基于 VMM 开发的虚拟化产品部分已经放弃了 Linux 半虚拟化，而专注基于硬件辅助的全虚拟化开发，来支持未经修改的操作系统。

3）X86 “硬件辅助虚拟化”，其基本思想就是引入新的处理器运行模式和新的指令，使得 VMM 和 Guest OS 运行于不同的模式下，Guest OS 运行于受控模式，原来的一些敏感指令在受控模式下全部会陷入 VMM，这样就解决了部分非特权的敏感指令的“陷入-模拟”难题，而且模式切换时上下文的保存恢复由硬件来完成，这样就大大提高了“陷入-模拟”时上下文切换的效率。

以 Intel VT-x 硬件辅助虚拟化技术为例，该技术增加了在虚拟状态下的两种处理器工作模式：根（Root）操作模式和非根（Non-root）操作模式。VMM 运作在 Root 操作模式下，而 Guest OS 运行在 Non-root 操作模式下。这两个操作模式分别拥有自己的特权级环，VMM 和虚拟机的 Guest OS 分别运行在这两个操作模式的 0 环。这样，既能使 VMM 运行在 0 环，也能使 Guest OS 运行在 0 环，避免了修改 Guest OS。Root 操作模式和 Non-root 操作模式的切换是通过新增的 CPU 指令（如：VMXON,VMXOFF）来完成。

硬件辅助虚拟化技术消除了操作系统的 ring 转换问题，降低了虚拟化门槛，支持任何操作系统的虚拟化而无须修改 OS 内核，得到了虚拟化软件厂商的支持。硬件辅助虚拟化技术已经逐渐消除软件虚拟化技术之间的差别，并成为未来的发展趋势。

- **vCPU 机制**

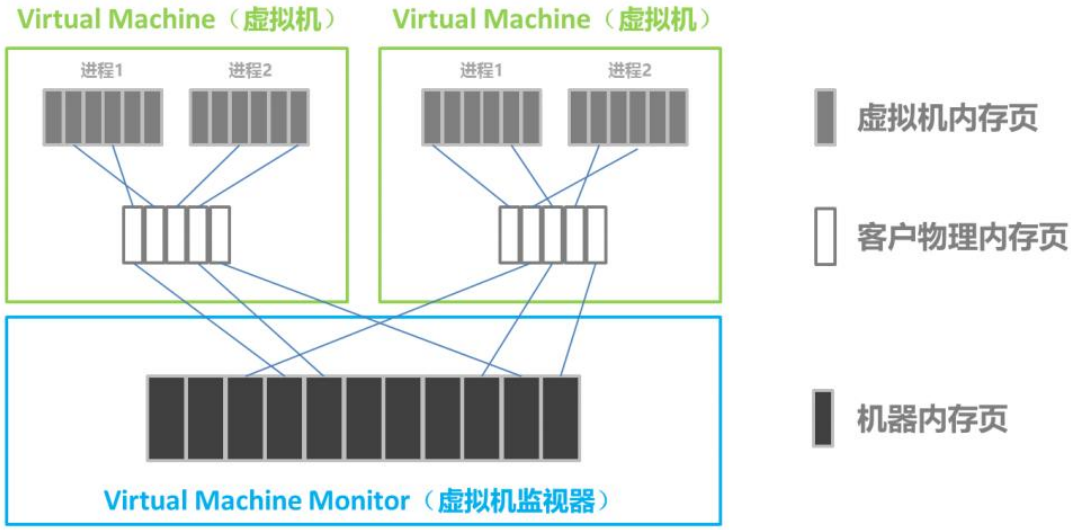


vCPU 调度机制

对虚拟机来说，不直接感知物理 CPU，虚拟机的计算单元通过 vCPU 对象来呈现。虚拟机只看到 VMM 呈现给它的 vCPU。在 VMM 中，每个 vCPU 对应一个 VMCS（Virtual-MachineControl Structure）结构，当 vcpu 被从物理 CPU 上切换下来的时候，其运行上下文会被保存在其对应的 VMCS 结构中；当 vcpu 被切换到 pcpu 上运行时，其运行上下文会从对应的 VMCS 结构中导入到物理 CPU 上。通过这种方式，实现各 vCPU 之间的独立运行。

从虚拟机系统的结构与功能划分可以看出，客户操作系统与虚拟机监视器共同构成了虚拟机系统的两级调度框架，如图所示是一个多核环境下虚拟机系统的两级调度框架。客户操作系统负责第 2 级调度,即线程或进程在 vCPU 上的调度（将核心线程映射到相应的虚拟 CPU 上）。虚拟机监视器负责第 1 级调度，即 vCPU 在物理处理单元上的调度。两级调度的调度策略和机制不存在依赖关系。vCPU 调度器负责物理处理器资源在各个虚拟机之间的分配与调度, 本质上即把各个虚拟机中的 vCPU 按照一定的策略和机制调度在物理处理单元上可以采用任意的策略来分配物理资源, 满足虚拟机的不同需求。vCPU 可以调度在一个或多个物理处理单元执行（分时复用或空间复用物理处理单元），也可以与物理处理单元建立一对一固定的映射关系（限制访问指定的物理处理单元）。

■ **内存虚拟化**



内存虚拟化三层模型

因为 VMM (Virtual Machine Monitor) 掌控所有系统资源，因此 VMM 握有整个内存资源，其负责页式内存管理，维护虚拟地址到机器地址的映射关系。因

Guest OS 本身亦有页式内存管理机制，则有 VMM 的整个系统就比正常系统多了一层映射：

A. 虚拟地址(VA)，指 Guest OS 提供给其应用程序使用的线性地址空间；

B. 物理地址(PA)，经 VMM 抽象的、虚拟机看到的伪物理地址；

C. 机器地址(MA)，真实的机器地址，即地址总线上出现的地址信号；

映射关系如下：Guest OS: PA = f(VA)、VMM: MA = g(PA)VMM 维护一套页表，负责 PA 到 MA 的映射。Guest OS 维护一套页表，负责 VA 到 PA 的映射。

实际运行时，用户程序访问 VA1，经 Guest OS 的页表转换得到 PA1，再由 VMM 介入，使用 VMM 的页表将 PA1 转换为 MA1。

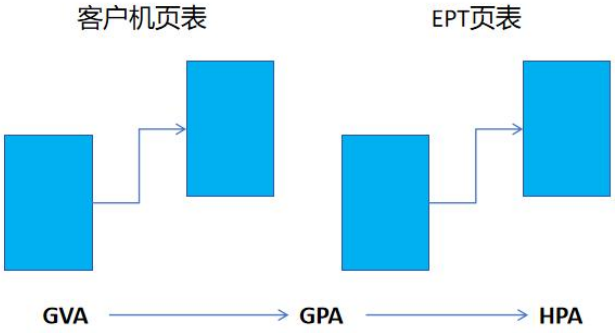
■ **页表虚拟化技术**

普通 MMU 只能完成一次虚拟地址到物理地址的映射，在虚拟机环境下，经过 MMU 转换所得到的“物理地址”并不是真正的机器地址。若需得到真正的机器地址，必须由 VMM 介入，再经过一次映射才能得到总线上使用的机器地址。如果虚拟机的每个内存访问都需要 VMM 介入，并由软件模拟地址转换的效率是很低下的，几乎不具有实际可用性，为实现虚拟地址到机器地址的高效转换， 现普遍采用的思想是：由 VMM 根据映射 f 和 g 生成复合的映射 fg，并直接将这个映射关系写入 MMU。当前采用的页表虚拟化方法主要是 MMU 类虚拟化（MMU Paravirtualization）和影子页表，后者已被内存的硬件辅助虚拟化技术所替代。

1）MMU Paravirtualization

其基本原理是：当 Guest OS 创建一个新的页表时，会从它所维护的空闲内存中分配一个页面，并向 VMM 注册该页面，VMM 会剥夺 Guest OS 对该页表的写权限，之后 GuestOS 对该页表的写操作都会陷入到 VMM 加以验证和转换。VMM 会检查页表中的每一项，确保他们只映射了属于该虚拟机的机器页面，而且不得包含对页表页面的可写映射。后 VMM 会根据自己所维护的映射关系，将页表项中的物理地址替换为相应的机器地址，最后再把修改过的页表载入 MMU。如此，MMU 就可以根据修改过页表直接完成虚拟地址到机器地址的转换。

2）内存硬件辅助虚拟化



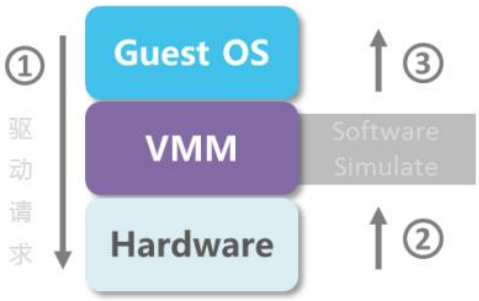
内存硬件辅助虚拟化技术原理图

内存的硬件辅助虚拟化技术是用于替代虚拟化技术中软件实现的“影子页表”的一种硬件辅助虚拟化技术，其基本原理是：GVA（客户操作系统的虚拟地址）->GPA（客户操作系统的物理地址）-> HPA（宿主操作系统的物理地址）两次地址转换都由 CPU 硬件自动完成（软件实现内存开销大、性能差）。以 VT-x 技术的页表扩充技术 Extended PageTable（EPT）为例，首先 VMM 预先把客户机物理地址转换到机器地址的 EPT 页表设置到 CPU 中；其次客户机修改客户机页表无需 VMM 干预；最后，地址转换时，CPU 自动查找两张页表完成客户机虚拟地址到机器地址的转换。使用内存的硬件辅助虚拟化技术，客户机运行过程中无需 VMM 干预，去除了大量软件开销，内存访问性能接近物理机。

■ **I/O 设备虚拟化**

VMM 通过 I/O 虚拟化来复用有限的外设资源，其通过截获 Guest OS 对 I/O 设备的访问请求，然后通过软件模拟真实的硬件，目前 I/O 设备的虚拟化方式主要有三种：设备接口完全模拟、前端 / 后端模拟、直接划分。

1）设备接口完全模拟：



即软件精确模拟与物理设备完全一样的接口，Guest OS 驱动无须修改就能驱动这个虚拟设备。

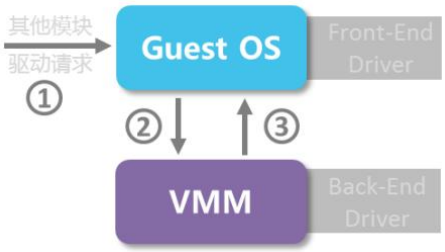
优点：没有额外的硬件开销，可重用现有驱动程序。

缺点：为完成一次操作要涉及到多个寄存器的操作，使得 VMM 要截获每个

寄存器访问并进行相应的模拟，这就导致多次上下文切换；由于是软件模拟，性能较低。

2）前端 / 后端模拟：

VMM 提供一个简化的驱动程序（后端, Back-End），Guest OS 中的驱动程序为前端(Front-End, FE)，前端驱动将来自其他模块的请求通过与 Guest OS 间的特殊通信机制直接发送给 Guest OS 的后端驱动，后端驱动在处理完请求后再发回通知给前端，VMM 即采用该方法。

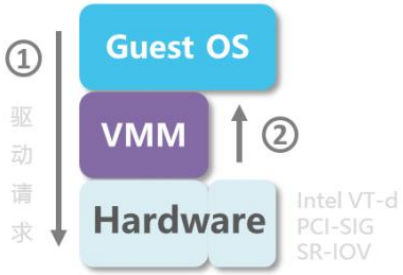


优点：基于事务的通信机制，能在很大程度上减少上下文切换开销，没有额外的硬件开销；

缺点：需要 GuestOS 实现前端驱动，后端驱动可能成为瓶颈。

3）直接划分

即直接将物理设备分配给某个 Guest OS，由 Guest OS 直接访问 I/O 设备（不经 VMM），目前与此相关的技术有 IOMMU（Intel VT-d, PCI-SIG 之 SR-IOV 等），旨在建立高效的 I/O 虚拟化直通道。



优点：可重用已有驱动，直接访问减少了虚拟化开销；

缺点：需要购买较多额外的硬件。

2.1.2. 虚拟计算关键特性

（一）内存 NUMA 技术

非统一内存访问（NUMA）是服务器 CPU 和内存设计的新架构。传统的服务器架构下把内存放到单一的存储池中，这对于单处理器或单核心的系统工作良

好。但是这种传统的统一访问方式，在多核心同时访问内存空间时会导致资源争用和性能问题。毕竟，CPU 应该可以访问所有的服务器内存，但是不需要总是保持占用。实际上，CPU 仅需要访问工作负载实际运行时所需的内存空间就可以了。

因此 NUMA 改变了内存对 CPU 的呈现方式。这是通过对服务器每个 CPU 的内存进行分区来实现的。每个分区（或内存块）称为 NUMA 节点，而和该分区相关的处理器可以更快地访问 NUMA 内存，而且不需要和其它的 NUMA 节点争用服务器上的资源（其它的内存分区分配给其它处理器）。

NUMA 的概念跟缓存相关。处理器的速度要比内存快得多，因此数据总是被移动到更快的本地缓存，这里处理器访问的速度要比通用内存快得多。NUMA 本质上为每个处理器配置了独有的整体系统缓存，减少了多处理器试图访问统一内存空间时的争用和延迟。

NUMA 与服务器虚拟化完全兼容，而且 NUMA 也可以支持任意一个处理器访问服务器上的任何一块内存区域。某个处理器当然可以访问位于不同区域上的内存数据，但是需要更多本地 NUMA 节点之外的传输，并且需要目标 NUMA 节点的确认。这增加了整体开销，影响了 CPU 和内存子系统的性能。

NUMA 对虚拟机负载不存在任何兼容性问题，但是理论上虚拟机最完美的方式应该是在某个 NUMA 节点内。这可以防止处理器需要跟其它的 NUMA 节点交互，从而导致工作负载性能下降。

（二）SR-IOV

通常针对虚拟化服务器的技术是通过软件模拟共享和虚拟化网络适配器的一个物理端口，以满足虚拟机的 I/O 需求，模拟软件的多个层为虚拟机作了 I/ O 决策，因此导致环境中出现瓶颈并影响 I/O 性能。提供的 SR-IOV 是一种不需要软件模拟就可以共享 I/O 设备 I/O 端口的物理功能的方法，主要利用 iNIC 实现网桥卸载虚拟网卡，允许将物理网络适配器的 SR-IOV 虚拟功能直接分配给虚拟机，可以提高网络吞吐量，并缩短网络延迟，同时减少处理网络流量所需的主机 CPU 开销。

技术原理：SR-IOV（Single Root I/O Virtualization）是 PCI-SIG 推出的一项标准，是虚拟通道（在物理网卡上对上层软件系统虚拟出多个物理通道，每个通道具备独立的 I/O 功能）的一个技术实现，用于将一个 PCIe 设备虚拟成多个 PCIe 设备，每个虚拟 PCIe 设备如同物理 PCIe 设备一样向上层软件提供服务。通过

SR-IOV 一个 PCIe 设备不仅可以导出多个 PCI 物理功能，还可以导出共享该 I/O 设备上的资源的一组虚拟功能，每个虚拟功能都可以被直接分配到一个虚拟机，能够让网络传输绕过软件模拟层，直接分配到虚拟机，实现了将 PCI 功能分配到多个虚拟接口以在虚拟化环境中共享一个 PCI 设备的目的，并且降低了软加模拟层中的 I/O 开销，因此实现了接近本机的性能。

允许管理程序简单地将虚拟功能映射到VM上以实现本机设备性能和隔离安全。SR-IOV 虚拟出的通道分为两个类型：

1、PF(Physical Function) 是完整的 PCIe 设备，包含了全面的管理、配置功能， Hypervisor 通过 PF 来管理和配置网卡的所有 I/O 资源。

2、VF(Virtual Funciton)是一个简化的 PCIe 设备，仅仅包含了 I/O 功能，通过 PF 衍生而来好象物理网卡硬件资源的一个切片，对于 Hypervisor 来说，这个 VF 同一块普通的 PCIe 网卡一模一样。

通过 SR-IOV 可满足高网络 IO 应用要求，无需特别安装驱动，且无损热迁移、内存复用、虚拟机网络管控等虚拟化特性。

（三）虚拟机生命周期管理

HCI 提供了虚拟机从创建至删除整个过程中的全面管理，就像人类的生命周期一样，虚拟机最基本的生命周期就是创建、使用和删除这三个状态。当然还包含如下几个状态：

- 创建虚拟机
- 虚拟机开关机、重启、挂起
- 虚拟机上的操作系统安装
- 创建模板
- 更新虚拟机硬件配置
- 迁移虚拟机及/或虚拟机的存储资源
- 分析虚拟机的资源利用情况
- 虚拟机备份
- 虚拟机恢复
- 删除虚拟机

在虚拟机生命周期内，虚拟机可能会在某一个时间点经历上述这些状态。HCI 提供了完善的虚拟机生命周期管理工具，我们可以通过对虚拟机生命周期的

规划，可以想要最大化的发挥虚拟机的作用。

（四）虚拟机热迁移

虚拟化环境中，物理服务器和存储上承载更多的业务和数据，设备故障时造成的影响更大。提供虚拟机热迁移技术，降低宕机带来的风险、减少业务中断的时间。

HCI 虚拟机热迁移技术是指把一个虚拟机从一台物理服务器迁移到另一台物理服务器上，即虚拟机保存/恢复(Save/Restore)。首先将整个虚拟机的运行状态完整保存下来，同时可以快速的恢复到目标硬件平台上，恢复以后虚拟机仍旧平滑运行，用户不会察觉到任何差异。虚拟机的热迁移技术主要被用于双机容错、负载均衡和节能降耗等应用场景。热迁移提供内存压缩技术，使热迁移效率提升一倍，可支持并发多达 4 台虚拟机同时迁移。

功能价值：

1、在设备维护过程中，通过热迁移手动将应用迁移至另一台服务器，维护结束后再迁回来，中间应用不停机，减少计划内宕机时间。

2、可结合资源动态调度策略，例如在夜晚虚拟机负荷减少时，通过预先配置自动将虚拟机迁移集中至部分服务器，减少服务器的运行数量，从而降低设备运营能耗上的支出。

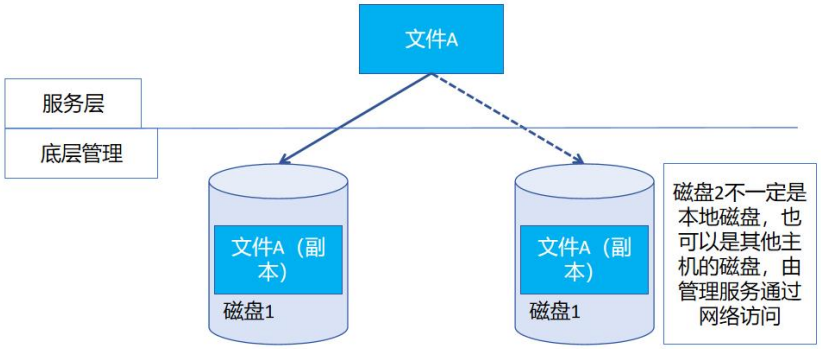
2.2. 虚拟存储

2.2.1. 虚拟存储技术原理

虚拟存储基于底层 Hypervisor 之上，通过主机管理、磁盘管理、缓存技术、存储网络、冗余副本等技术，管理集群内所有硬盘，“池化”集群所有硬盘存储的空间，通过向 VMP 提供访问接口，使得虚拟机可以进行业务数据的保存、管理和读写等整个存储过程中的操作。

（一）文件副本

所谓文件副本，即将文件数据保存多份的一种冗余技术。副本颗粒度是文件级别。例如两个副本，即把文件 A 同时保存到磁盘 1 和磁盘 2 上。并且保证在无故障情况下，两个副本始终保持一致。



技术特点：

存储池可用空间=集群全部机械磁盘空间/副本数（同构情况），因此副本是会降低实际可用容量的。

底层管理的副本对上层服务是透明的，上层无法感知副本的存在。磁盘管理、副本分布由底层服务负责，副本颗粒度是文件级。

在没有故障等异常情况下，文件副本数据是始终一致的，不存在所谓主副本和备副本之分。

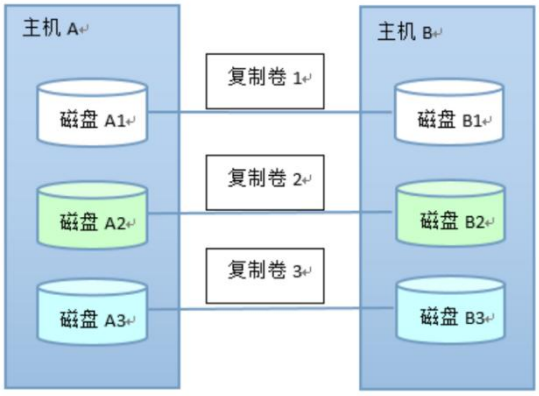
如果对文件 A 进行修改，如写入一段数据，这段数据会被同时写到两个副 本文件。如果是从文件 A 读取一段数据，则只会从其中一个副本读取。

（二）磁盘管理

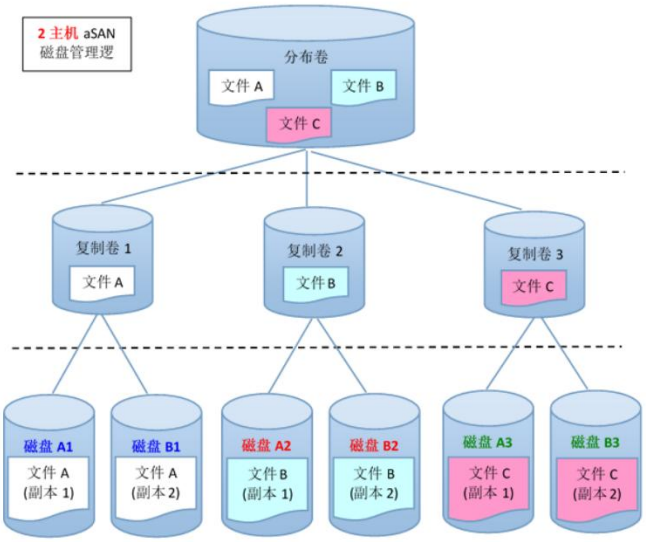
HCI 磁盘管理服务根据集群内主机数和 HCI 初始化时所选择的副本数决定集群内所有受管磁盘的组织策略。

在多主机集群下，可采用两个副本或三个副本组建 HCI 的磁盘管理，为了支持主机故障而不影响数据完整性的目标，复制卷的磁盘组的每个磁盘都必须是在不同主机上。即需要做到跨主机副本。跨主机副本的关键在于复制卷磁盘分组算法。

以下面场景为列（两台主机，每台主机各三块磁盘组建两个副本）：



当构建两副本，并且两台主机磁盘数相同时。主机间的磁盘会一一对应组成复制卷。逻辑视图如下：

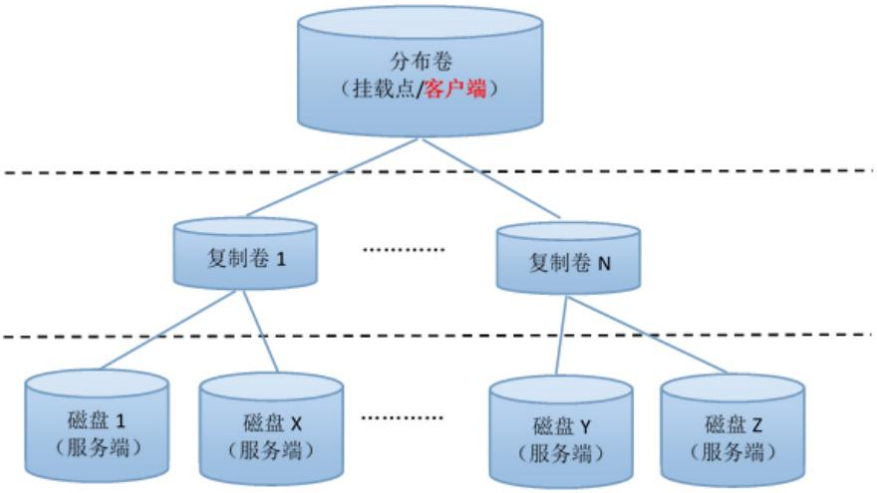


从逻辑视图上，可以看出来和前面提到的单主机逻辑视图并没有本质上的区别，只是最底层的磁盘分组时，保证了复制卷内下面的磁盘不在同一主机内，从而达到了文件跨主机副本的目标。

（三）SSD 读缓存加速原理

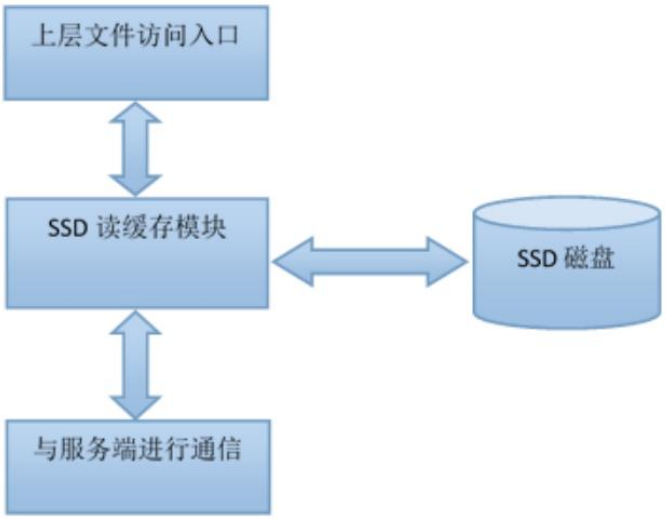
在 HCI 里面，会默认把系统内的 SSD 磁盘作为缓存盘使用，下面介绍 HCI SSD 读缓存原理。

首先需要区分 HCI 客户端和服务端概念。在 HCI 里面，负责处理底层磁盘 IO 称为服务端；负责向上层提供存储接口（如访问的挂载点）称为客户端。HCI SSD 读缓存工作在客户端,（注意：HCI 的 SSD 写缓存则工作在服务端）。逻辑视图如下：



下面抛开底层的分布卷、复制卷、磁盘分组等概念，仅在客户端上理解 SSD

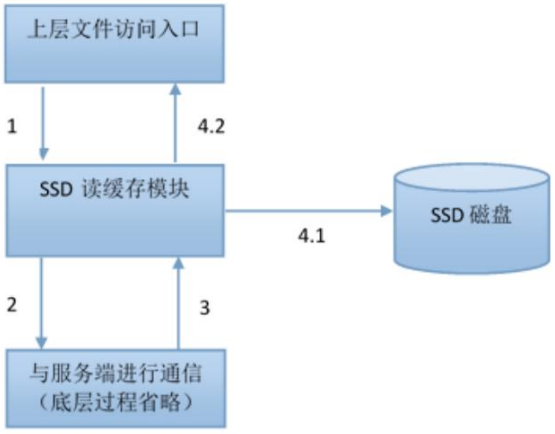
读缓存的原理。



SSD 读缓存的缓存颗粒度是按文件数据块缓存，不是文件整体。例如，A、B、C 三个文件，可以分别各缓存读过的一部分数据，没读过的部分不缓存。

简单地看，SSD 读缓存模块工作在文件访问入口和服务端通信层之间。所有对文件的 IO 动作都会经过 SSD 读缓存模块进行处理。下面分别针对首次文件读取、二次文件读取、文件写入 3 个过程说明工作流程。

■ **首次文件读取**



未缓存数据块的首次读操作步骤说明：

1、从上层下来一个针对 A 文件的区间块 [A1，A2] 的读操作，由于该数据块是首次读取，没命中 SSD 读缓存。该读操作会直接传递到下去，进入流程 2。

2、[A1，A2]的读操作继续传递到服务端，进行具体的读操作，完成后返回，进入流程 3。

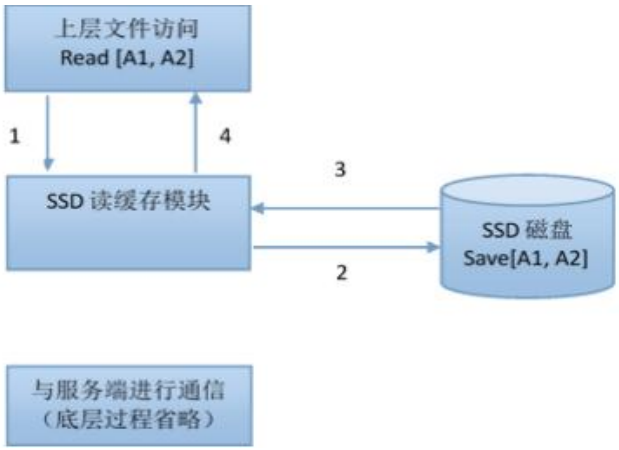
3、数据块[A1，A2]在流程 3 里面返回到 SSD 读缓存模块，进入流程 4。

4、SSD 读缓存模块会把数据块[A1，A2]复制一份保存到 SSD 磁盘并建立相

关 索引，对应 4.1。原数据块[A1，A2]继续往上返回到上层响应读操作，对应 4.2。注意 4.1、4.2 是并发进行，因此这个缓存动作不会对原操作造成延时。

5、至此，数据块[A1，A2]就被保存到 SSD 磁盘内，以备下次读取直接从 SSD 磁盘读取。

■ **二次文件读取**



针对已缓存数据块的二次读取步骤说明：

假设数据块[A1，A2]已经缓存到 SSD 磁盘内，

1. 从上层下来一个同样是针对 A 文件的区间块 [A1，A2] 的读操作。

2. 由于该数据块[A1，A2]已经有缓存，在 SSD 读缓存模块里面命中索引，从 而直接向 SSD 磁盘发起读出缓存数据块[A1，A2]的操作。

3. 缓存数据块[A1，A2]从 SSD 磁盘返回到 SSD 读缓存模块，进入流程 4

4. SSD 读缓存模块把缓存数据块[A1，A2]返回给上层。

至此，对缓存数据块[A1，A2]的重复读取直接在客户端返回，避免了服务端通信的流程，从而减少了延时和减轻了底层磁盘的 IO 压力。

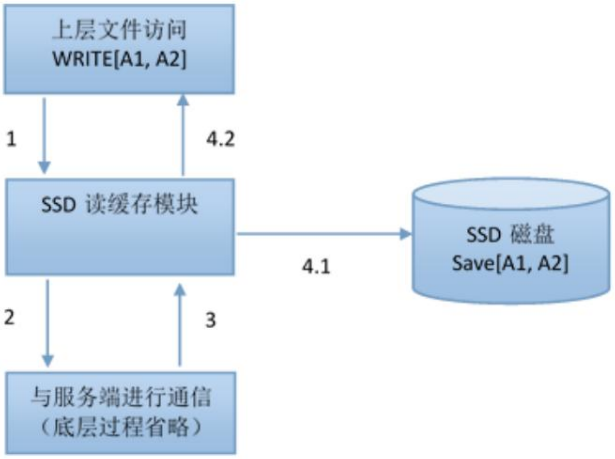
■ **文件写入**

虽然当前 HCI 实现的读缓存，但对于读缓存模块对于文件写入操作，也需要做相应的处理，以保证缓存的内容始终和底层磁盘一致，并且是最新的，但这个针对文件写入的处理并不是写缓存。

HCI 读缓存模块对写操作进行处理实质是基于最近访问原则，即最近写入的数据在不久的将来被读出的概率会比较高，例如文件共享服务器，某人传到文件服务器的文件，很快会其他人读出来下载。

HCI 读缓存对写操作的处理从实现上分为首次写预缓存、二次写更新缓存。

■ **文件块首次写预缓存**



流程说明：假设数据块[A1, A2]是首次写入。

1、写操作写来经过 SSD 读缓存模块。由于是写操作，SSD 读缓存会直接 PASS 到下层。

2、写操作一直传递到服务端，写入到底层磁盘，操作完成后会返回结果，进入流程 3。

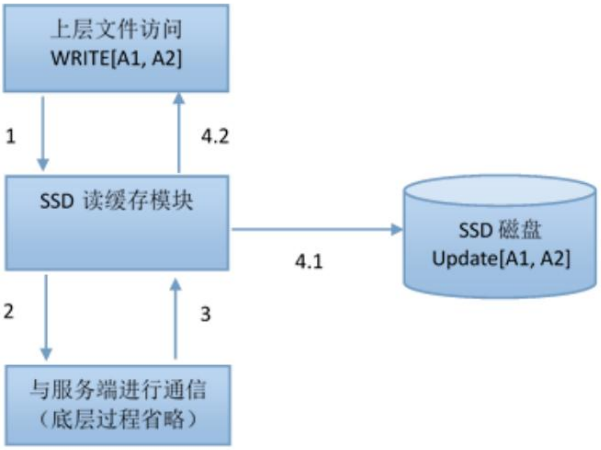
3、返回结果经过 SSD 读缓存模块，如果返回结果是成功的，表示底层数据已经成功写入，则进入流程 4。如果返回结果是失败，则不会进入流程 4，而是直接返回结果到上层。

4、SSD 读缓存模块会把数据块[A1，A2]复制一份保存到 SSD 磁盘并建立相关索引，对应 4.1。原返回结果继续往上返回到上层响应读操作，对应 4.2。注意 4.1、4.2 是并发进行，因此这个缓存动作不会对原操作造成延时。

至此，数据块[A1, A2]的写入也会保存到 SSD 磁盘上，以备下次访问。下次访问的流程与二次文件读取流程相同，从而提升了下次访问数据的速度。

■ **文件块二次写更新缓存**

SSD 读缓存文件块写更新是指对 SSD 读缓存已缓存的数据块进行更新的动作。



假设数据块[A1, A2]原来已经有缓存了，现在上层再次对[A1, A2]来一次写操作（例如更新内容）。

1、写操作写来经过 SSD 读缓存模块，由于是写操作，SSD 读缓存会直接 PASS 到下层。

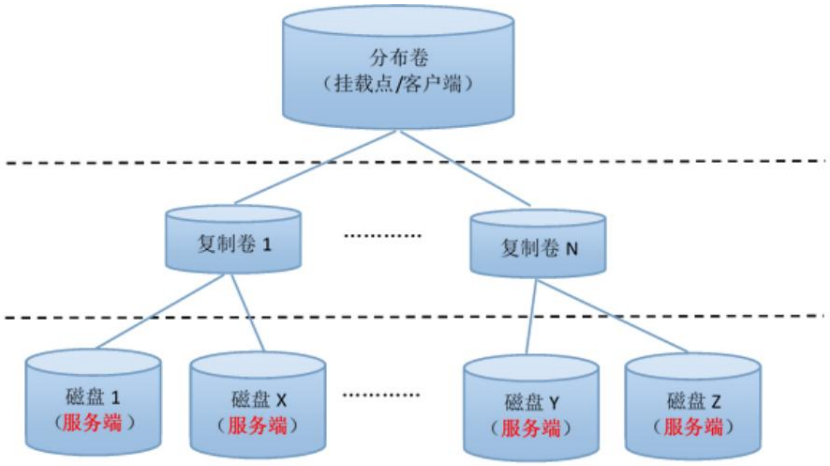
2、写操作一直传递到服务端，写入到底层磁盘，操作完成后会返回结果，进入流程 3。

3、返回结果经过 SSD 读缓存模块，如果返回结果是成功的，表示底层数据已经成功写入，可以更新 SSD 读缓存数据，进入流程 4。如果返回结果是失败，则不会进入更新流程。

4、SSD 读缓存模块会把数据块[A1，A2]复制一份更新到 SSD 磁盘并建立相关索引，对应 4.1。原返回结果继续往上返回到上层响应读操作，对应 4.2。 注意 4.1、4.2 是并发进行，因此这个缓存动作不会对原操作造成延时。

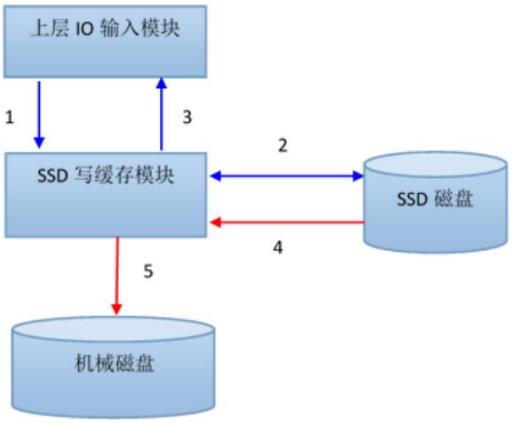
（四）SSD 写缓存加速原理

SSD 写缓存功能在 HCI2.0 开始支持。SSD 写缓存工作在服务端。由于写缓存工作在服务端，也就是说在每个副本上都有写缓存，即 SSD 写缓存也是多副本的。即使有 SSD 磁盘突然损坏，也能在副本数范围内保证数据的安全。



■ **SSD 写缓存模块结构**

SSD 写缓存原理是在机械硬盘上增加一层 SSD 写缓存层，见下图：



SSD 写缓存数据流分成蓝色和红色两部分。这两部分是同时在运行的，没有先后关系。蓝色部分是虚拟机有数据写入 SSD 缓存，红色部分是从 SSD 缓存读出数据回写到机械磁盘。流程如下：

1、上层写入数据请求到达 SSD 写缓存模块。

2、SSD 写缓存模块把数据写入到 SSD 磁盘，并获得返回值。

3、SSD 写缓存模块在确定数据写入 SSD 磁盘后，即立即返回上层模块写入成功。

4、SSD 写缓存模块在缓存数据累计到一定量后，从 SSD 磁盘读出数据。

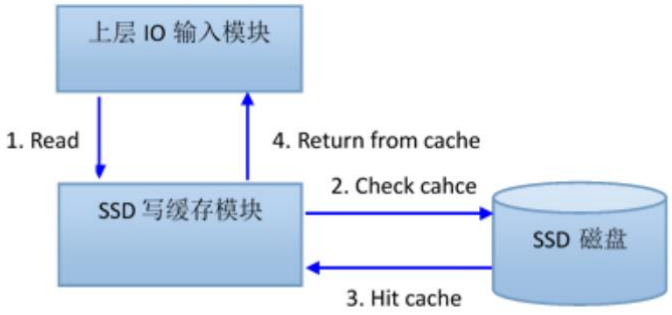
5、SSD 写缓存把从 SSD 磁盘读出的数据回写到机械磁盘。

其中，第 4、5 步是在后台自动进行的，不会干扰第 1、2、3 步的逻辑。

■ **SSD 写缓存数据读命中**

从 SSD 磁盘回写到机械磁盘是需要累积一定数据量后才会进行触发的。这时如果来了一个读数据的请求，SSD 写缓存模块会先确认该读请求是否在 SSD

写缓存数据内，如果有则从 SSD 缓存内返回；如果没有则透到机械硬盘去读取。



流程说明：

1、上层下发读请求。

2、SSD 写缓存模块先检查数据是否还在缓存内未回写。

3、命中缓存，返回数据（如果不命中缓存，则会返回从底层数据盘读取）。

4、向上层返回数据。

- **SSD 写缓存写满后处理**

如果上层持续对 SSD 写缓存进行大量不间断的数据写入，直到 SSD 写缓存空间用完。这时的上次继续写入数据的速度就会下降至约等于写缓存回写机械盘的速度。

当 SSD 磁盘用满时会出现写入数据流速度 <= 回写数据流速度。在虚拟机层面看，就是写入数据下降到机械盘速度。如果持续出现这种情况，说明 SSD 磁盘容量不足以应对业务 IO 写性能，需要增加 SSD 缓存盘解决。

- **当 SSD 磁盘故障或离线时的处理**

如前文所说，SSD 写缓存工作于服务端，有多副本机制。在多主机多副本场景下，如果一个 SSD 磁盘损坏后，其他副本的 SSD 还正常情况下，对数据安全不会造成影响。一旦 SSD 离线超过 10 分钟，缓存数据就视作失效，进入副本修

复流程。由于所有数据都是被 SSD 接管的，因此如果是误拔出 SSD 硬盘，需要在 10 分钟内插回来，否则会认为该副本数据全部需要重建。

（五）HCI 存储数据可靠性保障

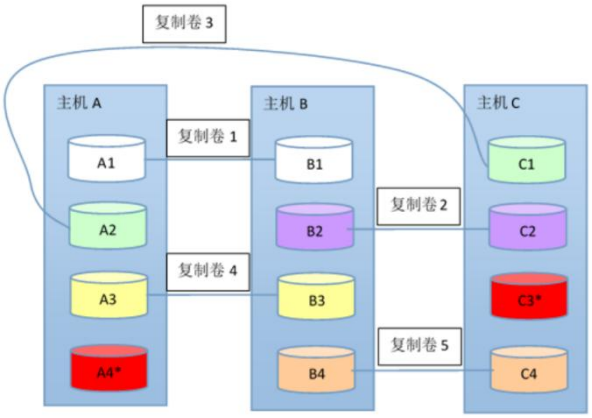
■ **磁盘故障时的保障机制**

如果在磁盘故障后，超过了设置的超时时间依然没有人工介入处理，HCI 将会自动进行数据重建，以保证数据副本数完备，确保数据可靠性。同时采用了热备盘的保障机制。

HCI 在初始化阶段会自动配置至少把集群里副本数个磁盘作为热备盘。

注意不是每个主机一个热备盘，而是一个集群里面全局使用。热备盘在初始化时不会纳入 HCI 复制卷内，只是作为一个不使用的磁盘带电存在，因此热备盘的空间不会反映到 HCI 的空间里面。

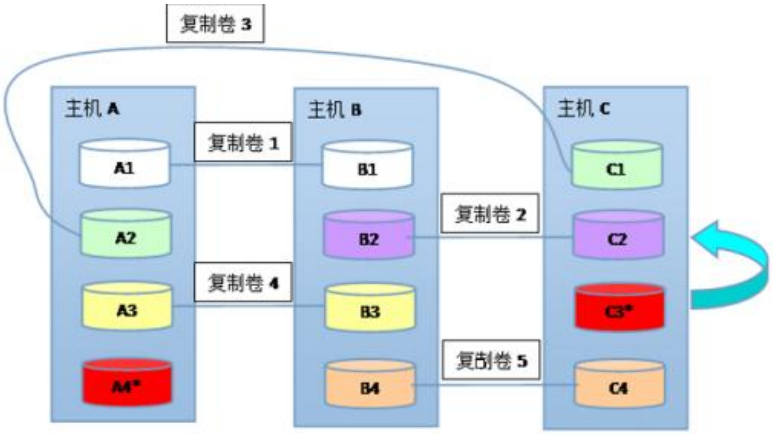
例如两个副本时会至少保留两个热备盘，三个副本时会至少保留三个热备盘。这些热备盘不会集中在一个主机上面，而是分布在不同主机上（符合副本跨主机原则）。下面以 3 主机 2 副本，每主机 4 个硬盘为例子。



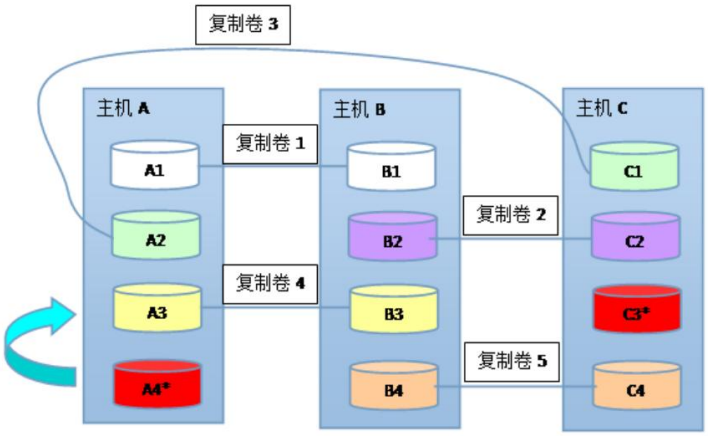
上图是 3 主机 2 副本，每主机 4 磁盘的分组例子。其中磁盘 A4、磁盘 C3 是作为热备盘保留的，并没有组成复制卷加入到 HCI 存储池内。

当任何一个主机的任意一个硬盘发生故障时，都可以按照跨主机副本原则自动使用 A4 或者 C3 来替换。

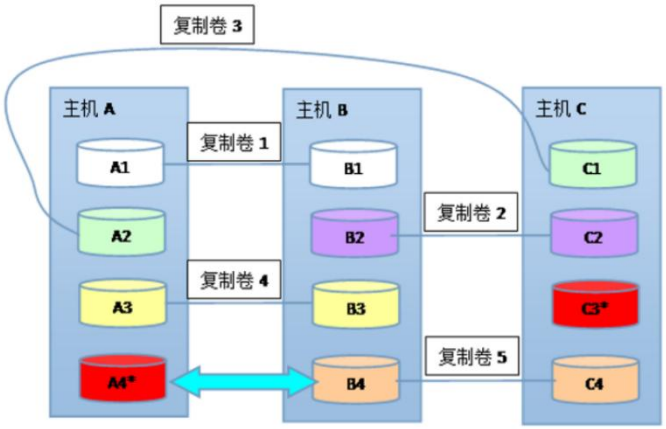
例 1：C2 损坏（C3 或者 A4 均可以用作替换



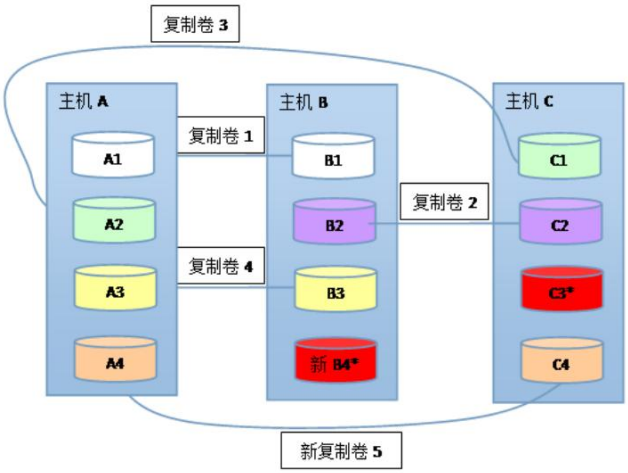
例 2：A3 损坏（C3 或者 A4 均可以用作替换）



例 3：B4 损坏（注意：这时只能用 A4 替换，原因是 C3 和 C4 同主机）



在 HCI 自动使用热备盘替换故障磁盘后，UI 上依然会显示原来的故障磁盘损坏，可以进行更换磁盘。这时新替换的硬盘会作为新热备盘使用，不需要执行数据回迁。这一点与前文没有热备盘会做数据回迁是不一样的。



以上面例子 3 为例，B4 损坏后，热备盘 A4 自动替换 B4 和 C4 建成新复制。然后人工介入，把损坏的 B4 用新磁盘替换，这时新 B4 会直接做热备盘使用，不再由数据回迁。

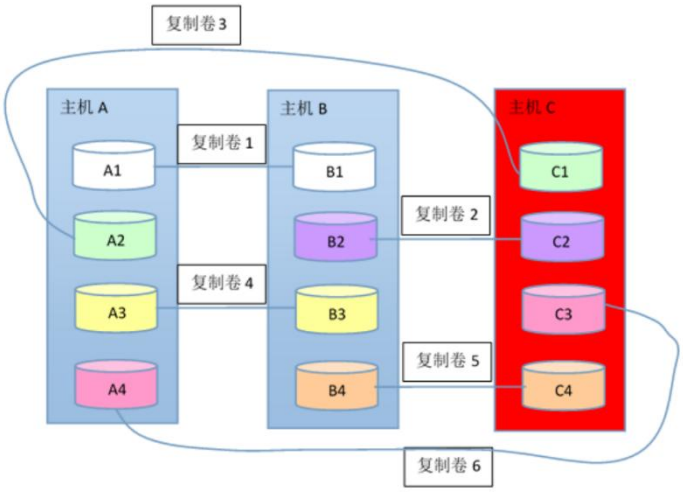
故障磁盘替换所有过程都可以带业务进行，不需要停机停止业务，就可以完成故障磁盘的替换，数据重建，相比 RAID 系统停业务重建有更大的可用性。

- **主机故障时的保障机制**

HCI 在多主机集群下，复制卷有个最高原则：跨主机建立复制卷。该原则的目的是为了达到在主机出现故障时，数据依然可用。在 2 主机 2 副本模式下，当主机 B 整个离线或，可以看到任何一个复制卷都依然有一个副本存在主机 A 上，数据依然可用，影响只是少了个副本。

在 2 主机 2 副本模式下，当主机 B 整个离线或，可以看到任何一个复制卷都依然有一个副本存在主机 A 上，数据依然可用，影响只是少了个副本。

略为复杂的例子（先不考虑有热备盘）：



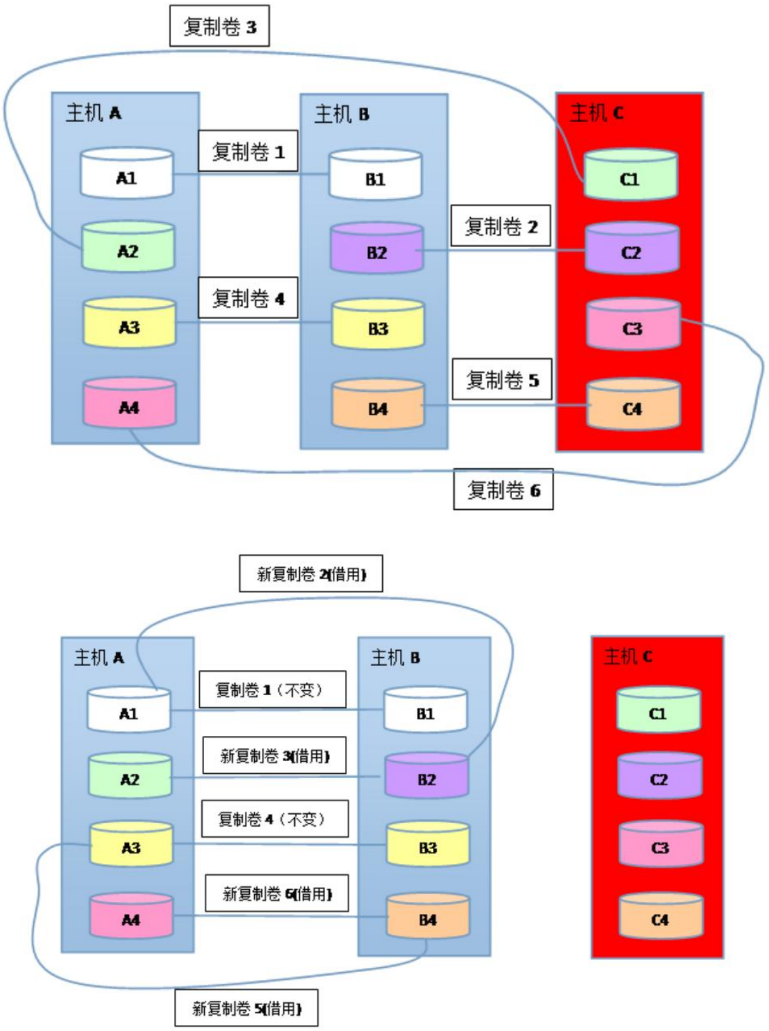
主机 C 离线后，剩余在线的复制卷任何一个都依然保持至少一个副本在线，

因此这时全局数据依然可用。

在主机故障后，在 HCI 高级设备里面有这样一个故障时间策略执行相应的处理：

假如入主机故障后直到超过设定的故障时间依然没有人工介入处理，那么 HCI 会采取自动替换动作在其他主机上重建副本。

例子：3 主机 2 副本，主机 C 出现故障。



对比上面 2 张图，可以看出在主机 C 故障并超时后，HCI 会在集群范围内寻找最佳借用磁盘组建复制卷，从而重建副本。这里的主机副本自动重建机制和单个故障硬盘的自动重建机制并没有本质差别，只是同时做了多个故障盘的重建。如果其中有热备盘，这是会自动使用热备盘。

注意，主机自动重建是有代价的，会复用其他磁盘的空间和性能。因此在条件允许情况下，应尽快替换主机。如果不想 HCI 才超时自动重建，可以到高级设置关闭主机自动重建功能。

■ **数据副本快速修复**

副本修复是指当某个磁盘出现离线再上线后，保存在上面的文件副本可能是旧数据，需要按照其他在线的文件副本进行修复的一个行为。典型的情况是主机短暂断网，导致副本不一致。

通过采用副本快速修复技术，即对于短暂离线的副本，只修复少量差异数据，从而避免了整个文件进行对比修复，达到快速修改的目的，同时，HCI 对业务 IO 和修复 IO 做了优先级控制，从而避免了副本修复 IO 对业务 IO 的影响。

2.2.2. 虚拟存储关键特性

（一）分布式存储

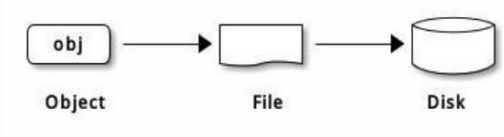
HCI 提供了一个可无限伸缩的存储集群，HCI 存储集群包含两种类型的守护进程：

Monitors 维护着集群运行图的主副本。一个 Monito 集群确保了当某个 Monitor 失效时的高可用性。存储集群客户端向 Monitor 索取集群运行图的最新副本。

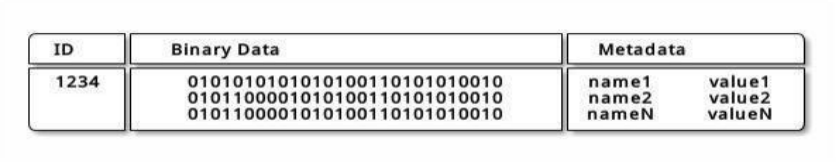
OSD 守护进程检查自身状态、以及其它 OSD 的状态，并报告给 Monitors。

存储集群的客户端和各个 OSD 守护进程使用 CRUSH 算法高效地计算数据位置，而不是依赖于一个中心化的查询表。

HCI 存储集群从 HCI 客户端接收数据——通过 HCI 块设备和 HCI 文件系统——并存储为对象。每个对象是文件系统中的一个文件，它们存储在对象存储设备上。 由 HCI OSD 守护进程处理存储设备上的读/写操作。



HCI OSD 在扁平的命名空间内把所有数据存储为对象（也就是没有目录层次）。对象包含一个标识符、二进制数据、和由名字/值对组成的元数据，元数据语义完全取决于 HCI 客户端。例如，HCI 文件系统用元数据存储文件属性，如文件所有者、创建日期、最后修改日期等等。



在传统架构里，客户端与一个中心化的组件通信（如网关、中间件、API 、前端等等），它作为一个复杂子系统的唯一入口，它引入单故障点的同时，也限制了性能和伸缩性（如果中心化组件挂了，整个系统就挂了）。

HCI 消除了集中网关，允许客户端直接和 HCI OSD 守护进程通讯。HCI OSD 守护进程自动在其它 HCI 节点上创建对象副本来确保数据安全和高可用性；为保证高可用性，Monitor 也实现了集群化。为消除中心节点，HCI 使用了 CRUSH 算法。

（二）CRUSH

HCI 客户端和 HCI OSD 守护进程都用 CRUSH 算法来计算对象的位置信息，而不是依赖于一个中心化的查询表。与以往方法相比，CRUSH 的数据管理机制更好，它很干脆地把工作分配给集群内的所有客户端和 OSD 来处理，因此具有极大的伸缩性。CRUSH 用智能数据复制确保弹性，更能适应超大规模存储。

■ **集群运行图**

HCI 依赖于 HCI 客户端和 OSD，因为它们知道集群的拓扑，这个拓扑由 5 张图共同描述，统称为“集群运行图 ”：

1、 Montior Map：包含集群的 fsid 、位置、名字、地址和端口，也包括当前版本、创建 时间、最近修改时间。

2、 OSD Map：包含集群 fsid、创建时间、最近修改时间、存储池列表、 副本数量、归置组数量、OSD 列表及其状态（如 up 、in ）。

3、 PG Map：包含归置组版本、其时间戳、最新的 OSD 运行图版本、占满率、以及各归置组详情，像归置组 ID 、upset 、actingset 、PG 状态（如 active+clean ），和各存储池的数据使用情况统计。

4、 CRUSH Map：包含存储设备列表、故障域树状结构（如设备、主机、机架、行、房间、等等）、和存储数据时如何利用此树状结构的规则。

5、 MDS Map：包含当前 MDS 图的版本、创建时间、最近修改时间，还包含了存储元数据的存储池、元数据服务器列表、还有哪些元数据服务器是 up 且 in 的。各运行图维护着各自运营状态的变更，HCI Monitor 维护着一份集群运行图

的主拷贝，包括集群成员、状态、变更、以及 HCI 存储集群的整体健康状况。

■ **高可用监视器**

HCI 客户端读或写数据前必须先连接到某个 HCI Monitor、获得最新的集群运行图副本。一个 HCI 存储集群只需要单个 Monitor 就能运行，但它就成了单一故障点（如果此 Monitor 宕机，HCI 客户端就不能读写数据了）。

为增强可靠性和容错能力，HCI 支持 Monitor Cluster；在一个 Monitor Cluster 内，延时以及其它错误会导致一到多个 Monitor 滞后于集群的当前状态，因此，HCI 的各 Monitor 例 程必须就集群的当前状态达成一致。HCI 总是使用大多数 Monitor（如：1 、2:3 、3:5 、4:6 等等）和 Paxos 算法就集群的当前状态达成一致。

■ **分布式存储工作机制**

在很多集群架构中，集群成员的主要目的就是让集中式接口知道它能访问哪些节点，然后此中央接口通过一个两级调度为客户端提供服务，在 PB 到 EB 级系统中这个调度系统必将成为最大的瓶颈。

HCI 消除了此瓶颈：其 OSD 守护进程和客户端都能感知集群，比如 HCI 客户端、各 OSD 守护进程都知道集群内其他的 OSD 守护进程，这样 OSD 就能直接和其它 OSD 守护进程和 Monitor 通讯。另外，HCI 客户端也能直接和 OSD 守护进程交互。

HCI 客户端、Monitor 和 OSD 守护进程可以相互直接交互，这意味着 OSD 可以利用本地节点的 CPU 和内存执行那些有可能拖垮中央服务器的任务。这种设计均衡了计算资源，带来几个好处：

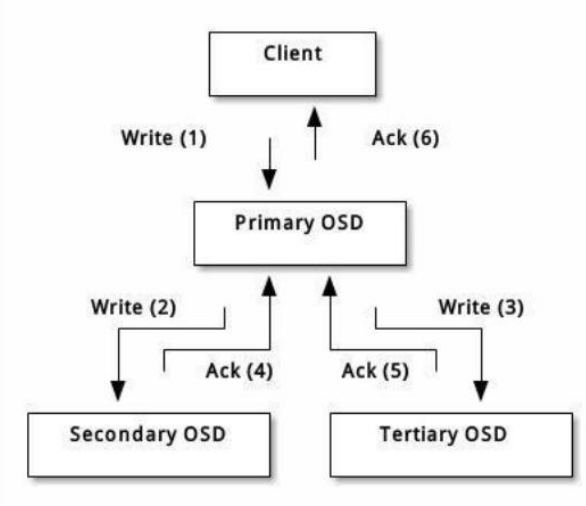
1、OSD 直接服务于客户端： 由于任何网络设备都有最大并发连接上限，规模巨大时中央化的系统其物理局限性就暴露了。HCI 允许客户端直接和 OSD 节点联系，这在消除单故障点的同时，提升了性能和系统总容量。HCI 客户端可按需维护和某 OSD 的会话，而不是一个中央服务器。

2、OSD 成员和状态：HCI OSD 加入集群后会持续报告自己的状态。在底层，OSD 状态为 up 或 down，反映它是否在运行、能否提供服务。

3、数据清洗作为维护数据一致性和清洁度的一部分，OSD 能清洗归置组内的对象。就是说，HCI OSD 能比较对象元数据与存储在其他 OSD 上的副本元数据，以捕捉 OSD 缺陷或文件系统错误（每天）。OSD 也能做深度清洗（每周），

即按位比较对象中的数据，以找出轻度清洗时未发现的硬盘坏扇区。

4、复制：和 HCI 客户端一样，OSD 也用 CRUSH 算法，用于计算副本存到哪里（也用于重均衡）。一个典型的写情形是，一个客户端用 CRUSH 算法算出对象应存到哪里，并把对象映射到存储池和归置组，然后查找 CRUSH 图来确定此归置组的主 OSD。客户端把对象写入目标归置组的主 OSD ，然后这个主 OSD 再用它的 CRUSH 图副本 找出用于放对象副本的第二、第三个 OSD ，并把数据复制到适当的归置组所对应的 第二、第三 OSD （要多少副本就有多少 OSD），最终，确认数据成功存储后反馈给客户端。



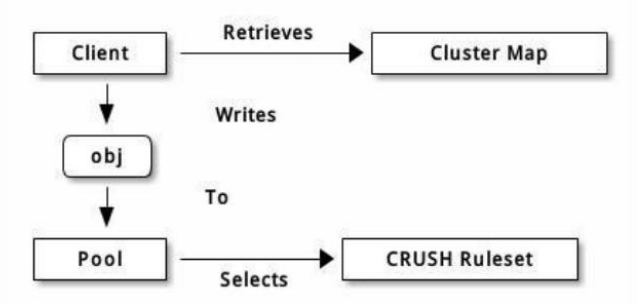
有了做副本的能力，OSD 守护进程就可以减轻客户端的复制压力，同时保证了数据的高可靠性和安全性。

（三）动态集群管理

■ **存储池**

HCI 存储系统支持“池”概念，它是存储对象的逻辑分区。

HCI 客户端从 Monitor 获取一张集群运行图，并把对象写入存储池。存储池的 size 或副本数、CRUSH 规则集和归置组数量决定着 HCI 如何放置数据存储池至少可设置以下参数：



1、对象的所有权/访问权限；

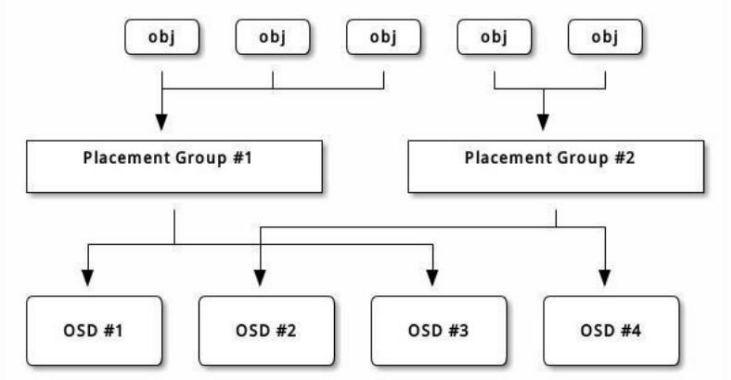
2、归置组数量；

3、使用的 CRUSH 规则集。

■ **PG 与 OSD 映射方法**

每个存储池都有很多归置组， CRUSH 动态的把它们映射到 OSD 。HCI 客户端要存对象时，CRUSH 将把各对象映射到某个归置组。

把对象映射到归置组在 OSD 和客户端间创建了一个间接层。由于 HCI 集群必须能增大或缩小、并动态地重均衡。如果让客户端“知道 ”哪个 OSD 有哪个对象，就会导致客户端和 OSD 紧 耦合；相反，CRUSH 算法把对象映射到归置组、然后再把各归置组映射到一或多个 OSD，这一间接层可以让 HCI 在 OSD 守护进程和底层设备上线时动态地重均衡。下列图表描述了 CRUSH 如何将对象映射到归置组、再把归置组映射到 OSD。



有了集群运行图副本和 CRUSH 算法，客户端就能精确地计算出到哪个 OSD 读、写某特定对象。

■ **PG ID 计算方法**

HCI 客户端绑定到某 Monitor 时，会索取最新的集群运行图副本，有了此图，客户端就能知道集群内的所有 Monitor、OSD 和元数据服务器。然而它对对象的

位置一无所知。对象位置是计算出来的。

客户端只需输入对象 ID 和存储池，这很简单：HCI 把数据存在某存储池（如 liverpool ）中。当客户端想要存命名对象（如 john、paul、george、ringo 等等）时，它用对象名，一个哈希值、存储池中的归置组数和存储池名计算出归置组来。HCI 按下列步骤计算 PG ID：

1、客户端输入存储池 ID 和对象 ID（如 pool=”liverpool”和 object-id=”john”）；

2、CRUSH 拿到对象 ID 并哈希它；

3、CRUSH 用 PG 数（如 58 ）对哈希值取模，这就是归置组 ID；

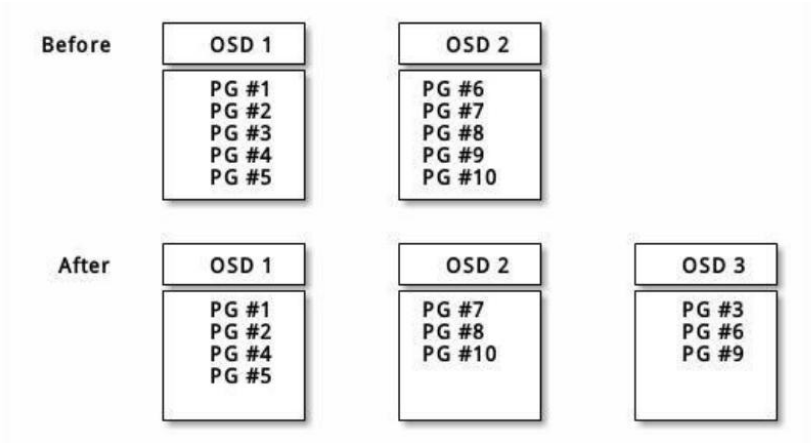
4、CRUSH 根据存储池名取得存储池 ID（如 liverpool = 4）；

5、CRUSH 把存储池 ID 加到 PG ID（如 4.58）之前。

计算对象位置远快于查询定位，CRUSH 算法允许客户端计算对象应该存到哪里，并允许客户端连接主 OSD 来存储或检索对象。

■ **新增节点负载重均衡技术**

当向 HCI 存储集群新增一个 OSD 守护进程时，集群运行图就要用新增的 OSD 更新，这个动作会更改集群运行图，因此也改变了对象位置，因为计算时的输入条件变了。下面的图描述了重均衡过程（此图很粗略，因为在大型集群里变动幅度小的多），是其中的一些而不是所有 PG 都从已有 OSD （OSD 1 和 2 ）迁移到新 OSD（OSD 3）。即使在重均衡中，CRUSH 都是稳定的，很多归置组仍维持最初的配置，且各 OSD 都腾出了些空间，所以重均衡完成后新 OSD 上不会出现负载突增。



■ **数据一致性保证**

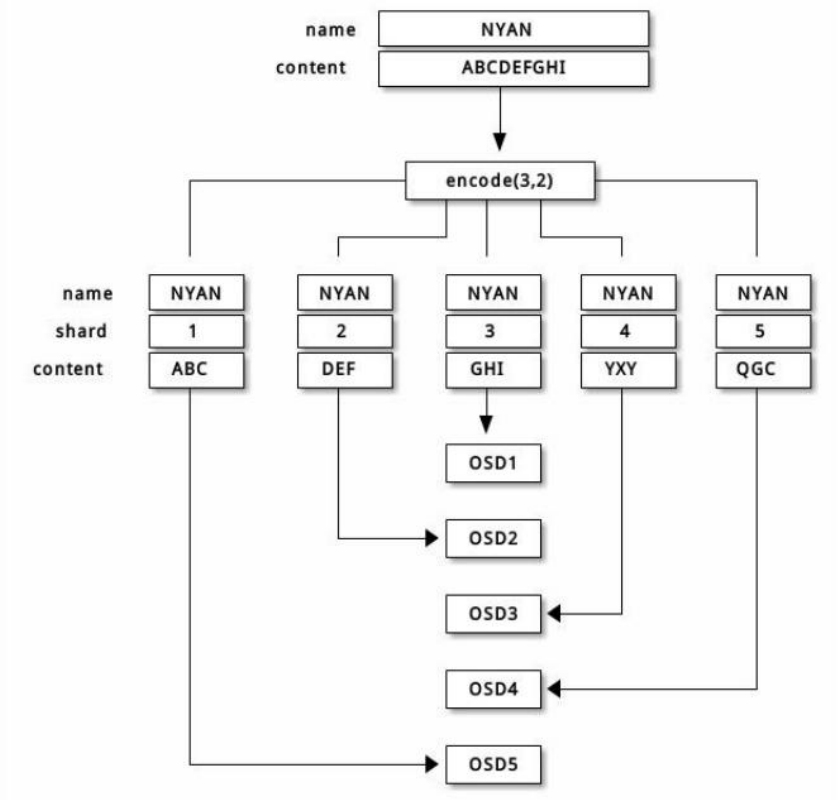
作为维护数据一致和清洁的一部分，OSD 也能清洗归置组内的对象，也就

是说，OSD 会比较归置组内位于不同 OSD 的各对象副本的元数据。清洗（通常每天执行）是为捕获 OSD 缺陷和文件系统错误，OSD 也能执行深度清洗：按位比较对象内的数据；深度清洗（通常每周执行）是为捕捉那些在轻度清洗过程中未能发现的磁盘上的坏扇区。

（四）纠删编码

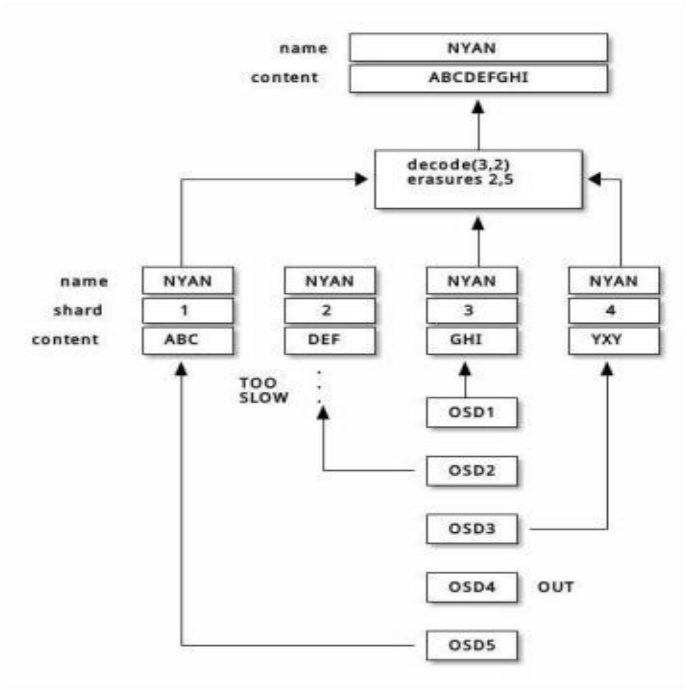
纠删码存储池把各对象存储为 K+M 个数据块，其中有 K 个数据块和 M 个编码块。此存储池的尺寸为 K+M，这样各块被存储到位于 acting set 中的 OSD ，块的位置也作为对象属性保存下来了。

当包含 ABCDEFGHI 的对象 NYAN 被写入存储池时，纠删编码函数把内容分割为三个数据块，只是简单地切割为三份：第一份包含 ABC、第二份是 DEF、最后是 GHI，若内容长度不是 K 的倍数则需填充；此函数还会创建两个编码块：第四个是 YXY、第五个是 GQC，各块分别存入 acting set 中的 OSD 内。这些块存储到相同名字（ NYAN ）的对象、但是位于不同的 OSD 上；分块顺序也必须保留，被存储为对象的一个属性（shard_t）追加到名字后面。包含 ABC 的块 1 存储在 OSD5 上、包含 YXY 的块 4 存储在 OSD3 上。



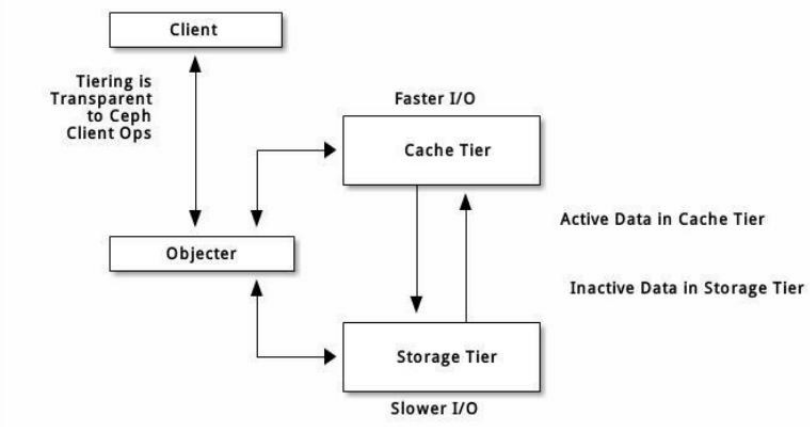
从纠删码存储池中读取 NYAN 对象时，解码函数会读取三个块：包含 ABC

的块 1 ，包含 GHI 的块 3 和包含 YXY 的块 4 ，然后重建对象的原始内容 ABCDEFGHI 。解码函数被告 知块 2 和 5 丢失了（被称为“擦除 ”），块 5 不可读是因为 OSD4 出局了。只要有三块读出就可以成功调用解码函数。OSD2 是最慢的，其数据未被采纳。



（五）缓存分级

对于后端存储层上的部分热点数据，缓存层能向 HCI 客户端提供更好的 IO 性能。缓存分层包含由相对高速、昂贵的存储设备（如固态硬盘）创建的存储池，并配置为缓存层；以及一个后端存储池，可以用纠删码编码的或者相对低速、便宜的设备，作为经济存储层。HCI 对象管理器会决定往哪里放置对象，分层代理决定何时把缓存层的对象刷回后端存储层。所以缓存层和后端存储层对 HCI 客户端来说是完全透明的。



2.3. 虚拟网络

2.3.1. 虚拟网络技术原理

网络虚拟化也是构建超融合架构中非常重要的一部分，如果在云计算、虚拟化的环境中，我们的网络如果继续采用传统 It 架构中硬件方式定义网络的话，就会存在诸多问题：

一、如何保障虚拟机在保持相应的网络策略不变的情况下进行虚机迁移。

二、虚拟化后的数据中心涉及业务众多，对外部提供云接入服务时，传统的 Vlan 技术已经无法满足业务隔离的需求，解决大规模租户和租户之间、业务和业务之间的安全隔离也是面临的首要问题。

三、虚拟化后的数据中心的业务系统的构建和上线对网络功能的快速部署、灵活弹性甚至成本，提出了更高的要求。

四、在传统网络中，不论底层的 IT 基础设施还是上层的应用，都由专属设备来完成。这些设备成本高昂，能力和位置僵化，难以快速响应新业务对网络快速、灵活自动化部署的需求。

基于上述问题，采用了业界成熟的 Overlay+NFV 的解决方案，通过 Overlay 的方式来构建大二层和实现业务系统之间的租户隔离，通过 NFV 实现网络中的所需各类网络功能资源（包括基础的路由交换、安全以及应用交付等）按需分配和灵活调度，从而实现超融合架构中的网络虚拟化。

（一）SDN

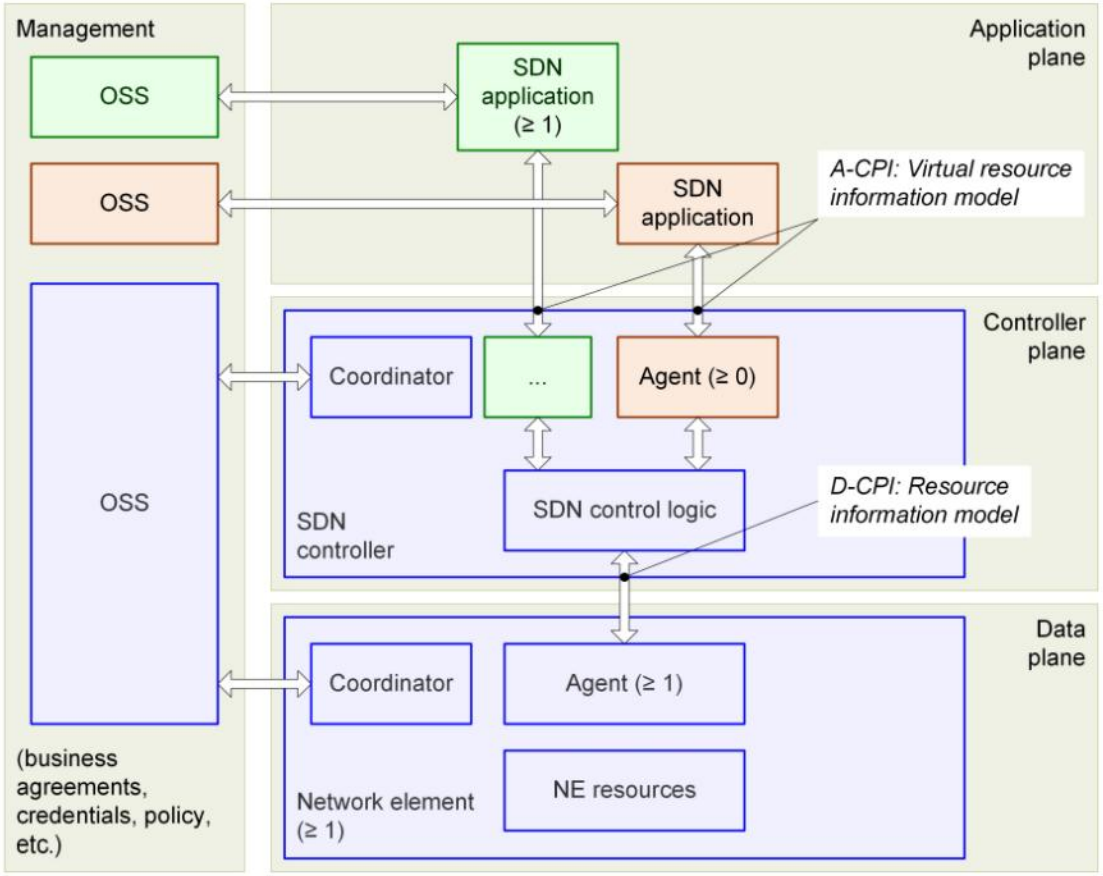
SDN（Software Defined Network，软件定义网络）是一种创新性的网络架构，它通过标准化技术（比如 openflow）实现网络设备的控制层面和数据层面的分离，进而实现对网络流量的灵活化、集中化、细粒度的控制，从而为网络的集中管理和应用的加速创新提供了良好的平台，由此可获得对网络的前所未有的可编程性、自动化和控制能力，使网络很容易适应变化的业务需求，从而建立高度可扩展的弹性网络。

从 SDN 的实现方式来看，广义和狭义两种。

广义的 SDN：主要包括网络虚拟化 NV（主要指的是 Overlay），网络功能虚拟化 NFV。

狭义的 SDN：主要指的是通过 OpenFlow 来实现 。

设备中的 SDN 实现没有采用上述的广义的 SDN 的方案，但由于实现架构上大同小异，用一副标准的 SDN 规范图来说明下：



可以这幅图看出实现 SDN 的重点在 Data plane 和 Controller plane，SDN 的核心思想就是数据面与控制面分离。

（二）NFV

以开放取代封闭，以通用替代专有——将原本传统的专业网元设备上的网络功能提取出来虚拟化，运行在通用的硬件平台上，业界称这种变化为 NFV。NFV（Network Functions Virtualisation 网络功能虚拟化）的目标是希望通过广泛采用的硬件承载各种各样的网络软件功能，实现软件的灵活加载，在数据中心、网络节点和用户端等各个位置灵活的配置，加快网络部署和调整的速度，降低业务部署的复杂度及总体投资成本，提高网络设备的统一化、通用化、适配性。

NFV 与 SDN 有很强的互补性，NFV 增加了功能部署的灵活性，SDN 可进一步推动 NFV 功能部署的灵活性和方便性。

通过 NFV 技术，将网络功能资源进行虚拟化，使得网络资源升级为虚拟化、

可流动的流态资源，Overlay 模型使流态网络资源的流动范围跳出了物理网络的束缚，可以在全网范围内按需流动，呈现出网络资源的统一池化状态，最终实现了超融合架构中网络资源的灵活定义、按需分配、随需调整。

NFV 的实现主要包含两个层面：数据平面和控制平面。传统数据平面：

在典型的虚拟化网络场景下，数据包将由网络接口卡接收，然后进行分类并生成规定的动作，并对数据包付诸实施。在传统的 Linux 模式下，系统接收数据包和将数据包发送出系统的过程占了包处理中很大一部分时间，换句话说，即使用户空间应用程序什么都不做，而只是将数据包从接收端口传送到发送端口，那么仍然会花费大量的处理时间。

当网卡从网络接收到一个数据帧后，会使用直接内存访问（DMA）将数据帧传送到针对这一目的而预先分配的内核缓冲区内，更新适当的接收描述符环，然后发出中断通知数据帧的到达。操作系统对中断进行处理，更新环，然后将数据帧交给网络堆栈。网络堆栈对数据进行处理，如果数据帧的目的地是本地套接字，那么就将数据复制到该套接字，而拥有该套接字的用户空间应用程序就接收到了这些数据。

进行传输时，用户应用程序通过系统调用将数据写入到一个套接字，使 Linux 内核将数据从用户缓冲区复制到内核缓冲区中。然后网络堆栈对数据进行处理，并根据需要对其进行封装，然后再调用网卡驱动程序。网卡驱动程序会更新适当的传输描述符环，并通知网卡有一个等待处理的传输任务。

网卡将数据帧从内核缓冲区转移到自己内置的先进先出（FIFO）缓冲区，然后将数据帧传输到网络。接着网卡会发出一个中断，通知数据帧已经成功传输，从而使内核释放与该数据帧相关的缓冲区。

传统模式下 CPU 损耗主要发生在如下几个地方：

中断处理：这包括在接收到中断时暂停正在执行的任务，对中断进行处理，并调度 soft IRQ 处理程序来执行中断调用的实际工作。随着网络流量负荷的增加，系统将会花费越来越多的时间来处理中断，当流量速度达到 10G 以太网卡的线路速度时就会严重影响性能。而对于有着多个 10G 以太网卡的情况，那么系统可以会被中断淹没，对所有的服务产生负面影响。

上下文切换：上下文切换指的是将来自当前执行线程的寄存器和状态信息加

以保存，之后再将来自被抢占线程的寄存器和状态信息加以恢复，使该线程能够从原先中断的地方重新开始执行。调度和中断都会引发上下文切换。

系统调用：系统调用会造成用户模式切换到内核模式，然后再切换回用户模式。这会造成管道冲刷并污染高速缓存。

数据复制：数据帧会从内核缓冲区复制到用户套接字，并从用户套接字复制到内核缓冲区。执行这一操作的时间取决于复制的数据量。

调度：调度程序使每个线程都能运行很短的一段时间，造成多任务内核中并发执行的假象。当发生调度定时器中断或在其他一些检查时间点上，Linux 调度程序就会运行，以检查当前线程是否时间已到。当调度程序决定应该运行另一个线程时，就会发生上下文切换。

（三）HCI 虚拟网络实现原理

■ **数据平面**

Linux 等通用操作系统，必须公平地对待网络应用程序和非网络应用程序，导致设计上达不到高 IO 吞吐，数据面设计上，借鉴了 netmap 和 dpdk 的方案，针对数据 IO 密集型网络应用程序设计。

1、支持专有网卡和通用网卡

对于 Intel 和 Broadcom 的 e1000e，igb，ixgbe，bnx2，tg3，bnx2x 等可编程网卡，支持高性能方案，对 e1000 等网卡，支持通用方案。保证硬件兼容性。

2、跨内核跨进程的全局内存池

设计并实现了零拷贝的数据面环境，一个跨内核跨进程的全局内存引用机制，真正做到网卡收包一次拷贝，所有进程共享引用的方式，数据可以从网卡传送到内核、应用层、虚拟机而无需再次拷贝。内存池自动增长，自动回收。

3、避免中断处理和上下文切换

单数据线程亲和锁定到硬件线程，避免内核和用户空间之间的上下文切换、线程切换和中断处理，同时每个线程有直接的高速缓冲，避免了缓冲区争用。

在理想情况下，当数据包到达系统时，所有处理该数据包所需的信息最好都已经在内核的本地高速缓存中。我们可以设想一下，如果当数据包到达时，查找表项目、数据流上下文、以及连接控制块都已经在高速缓存中的话，那么就可以直接对数据包进行处理，而无需“挂起”并等待外部顺序内存访问完成。

4、应用层数据面更稳定

内核态的小 BUG，可能导致系统宕机，而应用层进程，最糟糕的情况是进程死掉，我们设计了检测监控机制，在最极端的情况，即使进程意外死亡，也能秒级别做到虚拟机无感知的网络恢复。

数据平面负责报文的转发，是整个系统的核心，数据平面由多个数据转发线程和一个控制线程组成，控制线程负责接收控制进程配置的消息，数据线程是实现报文的处理。

在数据线程中实现快速路径与慢速路径分离的报文处理方式，报文的转发是基于 session 的，一条流匹配到一个 session，该条流的第一个报文负责查找各种表项，创建 session，并将查找表项的结果记录到 session 中，该条流的后续的报文只需查找 session，并根据 session 中记录的信息对报文进行处理和转发的。

系统中所有的报文都是由数据线程接收的，需要做转发的报文，不需要送到 linux 协议栈，直接在数据线程中处理后从网卡发出，对于到设备本身的报文(如 ssh，telnet，ospf, bgp, dhcp 等等)，数据线程无法直接处理，通过 TUN 接口将报文重新送到 linux 协议栈处理，从 linux 协议栈的发出的报文需经过数据线程中转后才可从折本发出。

数据面为底层处理和数据包 IO 提供了与硬件打交道的功能，而应用层协议栈在上方提供了一个优化的网络堆栈实现。与 Linux SMP 解决方案相比，降低了对 Linux 内核的依赖性，从而具有更好的扩展性和稳定性。

■ **控制平面**

有了数据面和协议栈做支持，控制面就可以实现丰富的应用功能。控制面实现了本地配套服务，一些基础功能例如 DHCP 服务，RSTP 服务，DNS 代理功能。这些内置服务可以直接提供给虚拟机，用户无需安装第三方类似软件。

2.3.2. 虚拟网络关键特性

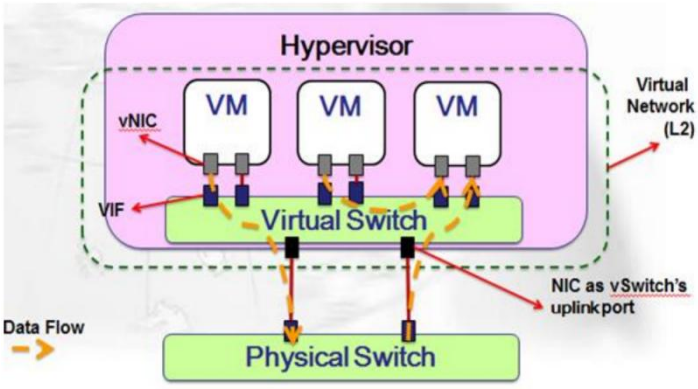
（一）虚拟交换

虚拟分布式交换机是管理多台主机上的虚拟交换机的虚拟网络管理方式，包括对主机的物理端口和虚拟机虚拟端口的管理。

提供的虚拟分布式交换机就是把分布在集群中多台主机的单一交换机逻辑

上 组成一个大的集中式交换机，减少每台虚拟交换机需要单独分别配置过程，同时为集群级别的网络连接提供一个集中控制点，使虚拟环境中的网络配置不再以主机为单位，简化虚拟机网络连接的部署、管理和监控，适合于大规模的网络部署。

虚拟分布式交换机可以保证虚拟机在主机之间迁移时网络配置的一致性，同时提供丰富的网络配置管理功能，端口动态绑定，静态绑定，IP 接入控制、虚拟机网络 Qos，实现网络资源统一管理，实时化网络监控。



支持二三层转发；

支持 access VLAN、trunk VLAN；

支持动态聚合 LACP 协议，以及聚合成员口流量动态分担；

支持 BFD 链路状态监测；

支持 STP；

支持细粒度的 Qos；

支持 IPV6

支持多种隧道协议（GRE, VXLAN, IPsec, GRE and VXLAN over IPsec）

支持通过 NetFlow sFlow IPFIX，SPAN，RSPAN，和 GRE-tunneled 镜像使虚拟机内部通讯可以被监控。

（二）虚拟防火墙

超融合软件支持防火墙功能，通过双向检测网络流量，有效识别来自网络层和应用层的内容风险，可以根据接口、方向、源 IP、源端口、目标 IP、目标端口、协议等维度进行精细化安全控制与管理。

3. 超融合核心价值

3.1. 可靠性

超融合架构，通过软件定义数据中心的方式，将计算、存储、网络包括安全形成一个大的资源池，可以灵活快速的构建数据中心里的业务系统，在不损失业务性能的情况下，还采用了多种技术来为业务系统提供了可靠性方面的保障。比如虚拟机的热迁移技术、虚拟机的 HA 技术、虚拟机数据备份和恢复、存储的多副本技术。

3.2. 灵活弹性

超融合架构，采用分布式 Scale-out 的架构设计，在业务系统需要扩展升级时，只需增加相应的服务器计算节点，及可实现容量和性能的线性提升。

3.3. 易操作性

HCI 超融合架构在软件定义网络方面，舍弃了复杂的传统硬件厂商的复杂的解决方案、采用更加灵活和简单的方式来实现 SDDC 中的整体 IT 基础服务，比如在实现 Overlay 的时候，用 Vxlan 实现多租户的业务安全隔离，但对于用户为透明无感知，不需要精通各种复杂的技术，和繁琐的配置操作，只需通简单的通过所画即所得的方式，即可构建业务系统的完逻辑。极大简化了超融合架构下的数据中心运维和管理工作。

4. 超融合产品规格

4.1. HCI 节点

HCI 采用分布式存储架构，存储节点采用标准 X86 架构服务器，根据内置硬盘密度不同细分为 3 种节点，分别为 HCI（12 盘位）、HCI（24 盘位）和 HCI（36 盘位）。

HCI 的节点配置从 3 节点起配，最大可横向扩展至 512 个节点，存储容量和性能线性增长。因此 HCI 可用于构建、维护 PB 级数据集群。

（一）CPU

HCI 元数据服务器对 CPU 敏感，它会动态地重分布它们的负载，HCI 元数据服务器支持配置双路 XEON 处理器，具有强悍的处理能力。

HCI OSD 运行着 RADOS 服务、用 CRUSH 计算数据存放位置、复制数据、维护其自己的集群运行图副本，HCI OSD 同样可配置双路 XEON 处理器，具备强大的处理能力。

HCI Monitor 用于维护集群运行图的副本，配置单 CPU 即可。

（二）内存

元数据服务器和 Monitor 必须可以尽快地提供它们的数据，所以 HCI 节点标配 64GB 内存，确保每进程可分配足够的内存空间 。

OSD 的日常运行不需要那么多内存（如每进程 500MB）；然而在恢复期间它们占用内存比较大（如每进程每 TB 数据需要约 1GB 内存）。HCI 节点最大可支持 512GB 内存。

（三）数据存储

HCI 可实现海量存储，考虑到成本和性能的平衡，HCI 可为用户规划数据存储配置。来自操作系统的并行操作和到单个硬盘的多个守护进程并发读、写请求操作会影响性能。不同的文件系统也有局限性：btrfs 尚未稳定到可以用于生产环境的程度，但它可以同时记日志并写入数据，而 xfs 和 ext4 却不能。

因为 HCI 发送 ACK 前必须把所有数据写入日志（至少对 xfs 和 ext4 来说是），因此均衡日志和 OSD 性能相当重要。

（四）硬盘驱动器

OSD 应该有足够的空间用于存储对象数据。考虑到大硬盘的每 GB 成本，我 们建议用容量更大的硬盘。单个驱动器容量越大，其对应的 OSD 所需内存就越大，特别是在重均衡、回填、恢复期间。根据经验，1TB 的存储空间大约需要 1GB 内存。

存储驱动器受限于寻道时间、访问时间、读写时间，还有总吞吐量，这些物理局限性影响着整体系统性能，尤其在系统恢复期间。因此我们推荐独立的驱动器用于安装操作系统和软件， 另外每个 OSD 守护进程占用一个驱动器。大多数“slow OSD”问题的起因都是在相同的硬 盘上运行了操作系统、多个 OSD 或

多个日志文件。鉴于解决性能问题的成本差不多会超过另外增加磁盘驱动器，你应该在设计时就避免增加 OSD 存储驱动器的负担来提升性能。

HCI 允许在每块硬盘驱动器上运行多个 OSD ，但这会导致资源竞争并降低总体吞吐量；HCI 也允许把日志和对象数据存储在相同驱动器上，但这会增加记录写日志并回应客户端的延时，因为 HCI 必须先写入日志才会回应确认了写动作。btrfs 文件系统能同时写入日志 数据和对象数据，xfs 和 ext4 却不能。因此 HCI 建议应该分别在单独的硬盘运行操作系统、OSD 数据和 OSD 日志。

（五）SSD

HCI 使用 SSD 来降低随机访问时间和读延时，同时增加吞吐量。SSD 和硬盘相比每 GB 成本通常要高 10 倍以上，但访问时间至少比硬盘快 100 倍。SSD 没有可移动机械部件，所以不存在和硬盘一样的局限性。但 SSD 的顺序读写性能很重要，在为多个 OSD 存储日志时，具有高顺序读写吞吐量的 SSD 对 HCI 的整体性能提升至关重要。

对于日志和 SSD 时还有几个重要考量：

写密集语义：记日志涉及写密集语义，SSD 写入性能好于硬盘。

顺序写入：在一个 SSD 上为多个 OSD 存储多个日志时也必须考虑 SSD 的顺序写入极限，因为它们要同时处理多个 OSD 日志的写入请求。

分区对齐：采用了 SSD 的一个常见问题是常常忽略了分区对齐，这会导致 SSD 的数据传输速率慢很多，所以请确保分区对齐了。把 OSD 的日志存到 SSD、把对象数据存储到独立的硬盘可以明显提升性能。

提升 HCIFS 文件系统性能的一种方法是从 HCIFS 文件内容里分离出元数据。HCI 提供了默认的 metadata 存储池来存储 HCIFS 元数据，所以不需要给 HCIFS 元数据创建存储池，但是可以给它创建一个仅指向某主机 SSD 的 CRUSH 运行图。

（六）硬盘类型

HCI 支持 SSD、SAS、NL-SAS 和 SATA 等多种硬盘接口和容量点选择，用户可根据存储容量和存储性能有多种选择。

（七）节点网卡

HCI 存储集群每个节点部署两个双端口 10Gbps 网卡分别用于公网（前端）

和集群网络（后端）。集群网络用于处理由数据复制产生的额外负载。使用 10Gbps 复制 1TB 数据的时间为 20 分钟。在一个 PB 级集群中， OSD 磁盘失败是常态，而非异常；在性价比合理的的前提下，系统管理员想让 PG 尽快从 degraded （降级）状态恢复到 active + clean 状态。可使用 VLAN 来提高网络和硬件可管理性使用 VLAN 来处理集，群和计算栈（如 OpenStack、CloudStack 等等）之间的 VM 流量时，采用 10G 网卡性能可以得到保障。网络的机架路由器到核心路由器应该有更大的带宽，如 40Gbps 到 100Gbps。

HCI 节点还配置了 2 个 GE 网口用于管理和部署，包括 SSH 访问、VM 映像上传、操作系统安装、端口管理等。

4.2. HCI 节点规格表

节点型号	**anmit-HCI-12S**	**anmit-HCI-24S**	**anmit-HCI-36S**
硬件规格			
体系架构	超融合一体机		
节点扩展能力	最大支持 512 个节点		
节点 CPU	Intel XEON，支持双路		
节点缓存	标配 96GB，可扩展至 512GB		
节点硬盘数	12	24	36
硬盘类型	SSD（日志存储）、SAS、NL-SAS、SATA（数据存储）		
节点网卡	2* 双端口 10Gb 网络、2*GE 网络		
存储	TOS 1.0 及以上版本		
存储服务类型	分布式块、文件		
数据冗余	多副本和 EC		
虚拟化平台	anmit-HCI 2.0 及以上版本		
文件系统			
架构	提供单一文件系统和统一命名空间，非对称架构，独立 MDS 集群，使用动态子树		
私有客户端	支持 Linux 客户端，支持用户态和内核态		
支持协议	POSIX、HDFS、NFS、CIFS		
配额	支持		
快照	支持		
负载均衡	支持		
动态扩容缩容	支持		
自动容量均衡	支持		