ChangHong HCP（Content Platform）

对象化存储

产品手册文档

2018 年 12 月

目　　录

HCP 对象化存储

一、基于对象的内容云存储

根据 SNIA 标准，内容云存储的基本单元不再是文件或块而是对象，利用对象特性彻底解决传统文件存储问题。基于对象实现内容云存储，已经获得了共识。为了满足内容云的需要，在对象基础上，各家对于内容云存储定义不同属性，根据各家理解的不同，被不同程度的实现。

对象存储产品某些关键属性，使其在市场上一夜成名，而传统的基于文件和块数据的存储系统就显得不足了。对象存储是建立公共云和私有云存储的基础。对象存储使用唯一识别符来存取数据，而非物理地址。数据根据名称和唯一识别符进行存取，这意味着存储系统要读取数据标签和对象 ID。

Web 2.0 的企业和像 Facebook 这样的社交网站，已经选择用对象存储存放用户文件、图片和视频。但对象存储并不局限于在新的云架构和 Web 2.0 中使用，事实上，对于某些内容，其可以作为 Tier 1 的存储系统使用。企业们也正在使用对象存储来扩展其高性能的、昂贵的基于块和基于文件的存储基础架构，这些位于 Tier 3 或 Tier 4 的对象存储可以把快速增长的非结构化数据存放到低成本但具有高扩展性的对象存储层。对象存储已经以用于归档任务的内容寻址存储（CAS）的方式使用了很多年。但传统的 CAS 系统与现在的对象存储有很大的区别。对象存储已经发展用于分布式云的构建，以实现成本优化——这也是云的关键，并且通过 HTTP 协议访问。

基于对象的存储系统正在引起关注，并且开始向替代扩展网络附加存储（NAS）领域进军。对象系统拥有很多吸引人的特点，包括几乎无限的可扩展性、对处理能力和高速网络的依赖性小、基于 WEB 协议的访问而不是通过传统的存储命令、客户化的元数据以及低成本使用和可利用现成的组件。

对象

不是要管理块和文件，纯粹的对象存储系统管理的是对象。更精确的讲，所有现在的对象存储系统把文件作为对象来管理。对象通过唯一的 ID 进行标识，就像在基于文件的存储系统中，文件通过路径来标识一样。对象存放在扁平的地址空间中，这样可消除基于文件的存储系统中，分层的文件系统的复杂性和扩展性的挑战。

ChangHong HCP 能够支持 1,000 亿个对象 其性能比市场上其他内容管理产品高出 470%，具备诸多先进功能包括新型“秘密共享”数据保护（专利技术）、重复数据删除技术、对象复制和通用管理。

元数据

对象由元数据（可提供对象中数据的上下文关系信息）、策略和实际数据组成。

在基于文件的存储系统中，元数据仅仅是指文件的属性；对象存储系统中的元数据可以添加任何客户化的属性。基于文件的存储系统要做到这一点，需要应用程序（数据库）处理文件相关的其它额外信息。利用客户化的元数据，你可以把和一个文件（对象）相关的所有信息都保存在对象自身内部。

“客户化元数据允许建立丰富的、自包含的文件对象，可以存储在对象存储内部，能够以较少的管理开销建设大规模的非结构化数据存储，”Terri McClure 说，马萨诸塞州米尔福德的企业策略集团（ESG）的高级分析师。

固定对象

纯对象存储代表一个固定内容的仓库；这意味着对象可以被创建、删除和读取，但不能被修改。相反，对象的修改是通过创建一个新版本的对象来实现的。因此，锁定和多用户访问的挑战——基于文件的存储系统所棘手的问题——在对象存储系统中是不存在的。

如果多个用户同时修改同一个文件（对象），对象存储系统会简单的生成这个文件的多个版本。不支持就地更新，使对象存储更加适合分布式的存储和分布式的访问。

冗余性

对象存储通过在多个节点上存储相同对象的多个副本实现冗余性和高可靠性。在创建对象的时候，它首先由一个节点创建，随后根据适当的策略复制到一个或多个其它节点。节点可以部署在同一个数据中心，也可以是地理上分开的。由于不支持就地更新，使得多节点副本对象冗余的复杂度很小。对于传统的存储系统，保留拷贝（复制）的文件和块的同步访问的多个实例是一个巨大的挑战；这是非常复杂的，只能够通过设置严格的限制条件实现，例如在定义好的延迟约束之内。

协议支持

传统的基于块和基于文件的协议在数据中心工作的很好，性能优良，延时也不是问题。但它们并不适合地理上分开的访问方式，且由于延时不可预知，所以也不适合构建云。

此外，传统的文件系统协议（CIFS 和 NFS）利用 TCP 端口进行通信，这些只在内部网络可用，很少出现在互联网上。相反，对象存储通常通过基于 HTTP 协议的 REST API 访问。命令通过 HTTP 发送到对象存储的方式非常简便：put 用来建立一个对象，get 用来读取一个对象，delete 用来清除对象而 list 用来列出对象列表。

应用软件支持与集成

由于缺乏传统的数据存储协议没有对于对象的整体访问的支持，因此依赖 REST API 需要成为访问对象存储的一体化努力。

一些商业应用，已经增加了对于对象存储集成的支持，主要连接到 Amazon S3 云存储。

云存储网关

由于业界仍然在争论标准，对象存储的集成仍然没有得到广泛的推广。对象存储网关，通常叫做云存储网关，提供了另外一种访问对象存储的方式。其定位于传统存储和对象存储之间，它们通常通过预定义的策略构建在两者之间。

云功能

因为富媒体和 WEB 2.0 应用是对象存储的核心目标，因此通过广域网或互联网进行共享访问的相关功能是非常重要的。

多租户和不同用户数据的安全隔离，对于用于企业应用的对象存储产品是必须的功能。安

全性不仅仅是指加密，还包括对于租户、命名空间以及对象访问的控制。
服务水平协议（SLA）管理和支持多种服务级别对于云的使用也非常重要。
策略引擎可以帮助 SLA 的执行，例如对象实例的数量和每个实例应该存储到哪里，这是一个任何对象存储都应该提供的设备。此外，使用云的计量和收费的自动跟踪是必不可少的。

应用场景

对象存储并不适合用于交易数据频繁变化的情况，例如数据库。它也没有设计成用来替代 NAS 的文件共享；它只是简单的做到了文件服务，没有提供锁机制，而且通过提供文件的多个不同的版本实现单个“真实”文件的共享。

- 对象存储工作在内容不经常变化的非结构化数据存储
- 存储不活跃数据的交易存储层之外的存储层，或者是归档存储
- 在云空间中，它适合用于文件内容，特别是图像和视频。
- 富媒体文件的保存

二、ChangHong 基于对象的内容云平台

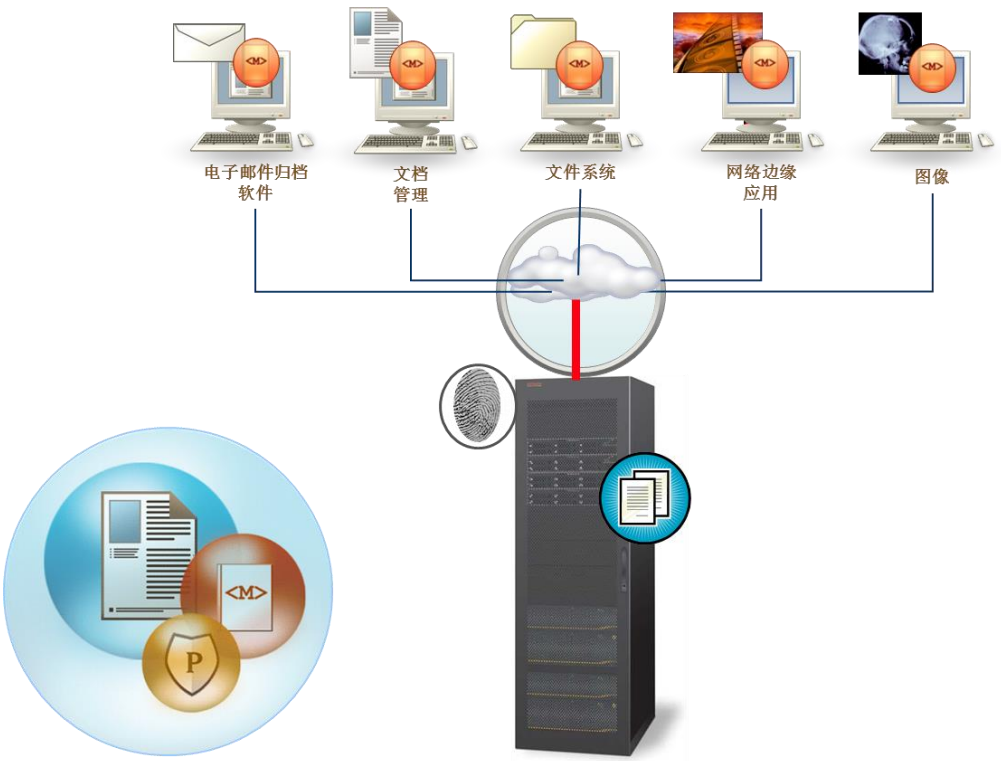
ChangHong 内容云平台构建在基础架构云上，满足爆炸式增长的非结构的长期保存和快速访问，查询的需要。即，根据业界的最新技术，采用分布集中的“云”存储架构，通过“云”中心存储和“云”端存储的协调统一，实现数据的集中存放和任意访问需要。
ChangHong 内容云平台与基础架构云一起构成数据中心的大型非结构化数据的统一存储容器，它的数据存储采用 SAIN 架构（SAN-attached Array of Independent Node）。此架构开放性好，支持开放标准的 FC 存储和使用 RAID 6 数据保护技术和支持 FC、SAS、SATA 等各种规格的硬盘。
ChangHong 内容云平台能够支持 320 亿个对象 比市场上其他类似产品高出 470%，具备容量大，功能丰富的特点。全面的数据管理功能包括数据保护（专利技术）、重复数据删除技术、对象复制，数据一致性检查。它是业界第一个使用户对服务器节点和存储容量分别进行独立扩展的平台。其他的解决方案都需要额外的服务器和处理能力来对存储进行扩展，而 HCP 减少了扩展所需的服务器节点，从而大幅降低散热要求和功耗，而且还大大简化

了管理。

存储的部署方面，节点和后端存储可实现动态配比。如：根据分析，一般文件对象的平均大小是 5K，对象数目会增长较快，在未来发展阶段可以依靠增加前端节点的数目提高对象处理性能，而保持存储空间容量和系统结构不变，以实现面向应用的动态高性价比部署。若对象大小为 5MB，那么存储容量增长将较快，通过单独增加磁盘容量满足需求。

在访问接口方面，内容云存储平台提供 NFS/CIFS/HTTP/S3/Openstack SWIFT/SMTP/NDMP 等标准访问接口，单一文件可以通过 HTTP、NFS、CIFS、S3、Openstack SWIFT 等各种协议同时访问，符合工业标准且具有良好的开放性。在未来用户应用系统中，机构中心不断产生的邮件、文件、数字化影像都可以集中到容器内。通过网络文件系统（NFS）、通用 Internet 文件系统（CIFS）、基于 Web 的分散式授权和版本控制（WebDAV）、超文本传输协议（HTTP）以及诸如存储管理主动规范（SMI-S）与现有应用系统整合，满足业务需要。

对于分布在各地的机构，根据机构的规模，选择部署不同数据收集器。对于每天大量数据产生的分支机构，在分支机构内部署基于硬件的收集器，分支机构只需要通过标准本地数据访问 NFS，CIFS，FTP 将数据放置在收集器内，它将自动将数据上传到数据中心的 HCP。每天产生少量数据的机构，通过软件方式，利用通用的 HTTP，HTTPS 协议定时传输数据到中心 HCP。



三、HCP 功能说明

HCP G10 对象化存储基本单元是对象

根据 SNIA 标准，内容云存储的基本单元不再是文件或块而是对象，利用对象特性彻底解决传统文件存储问题。每个对象是由文件数据和他的一系列属性组成的。面向对象存储不同于传统的文件系统，对象不是目录存储。每一个对象都分别指派一个元数据名或者 ID 以便在任何时候都能被检索到。

- 降低数据管理成本

  面向对象存储系统省去使企业数据系统时刻处于生产工作状态的复杂和昂贵的管理成本。面向对象存储，根据数据生命周期内的价值和不同阶段形成合理的层级和保护级别，降低生产系统数据量，达到降低数据管理成本的目的。

- 提供更好的数据保护，更高的数据可用性

  面向对象存储系统数据支持被复制成多个副本以保证数据可靠性。如果其中一份数据出现问题，数据恢复后台自动运行，而且数据可用性不会中断，性能也不会明显退化。

- 提供无限容量和可扩展性

  面向对象存储系统中，没有目录层次结构(树)，对象的存储通过 ID 值管理数据，而且不受文件(对象)数量、文件大小和文件系统容量的限制。

- 文件系统无法实现的元数据利用

  面向对象存储系统可以不需要文件名、日期和其他文件属性就可以查找文件。通过内置的搜索引擎实现数据快速定位。

- 无需备份

  对象存储系统并不需要备份。如果需要的话，多个副本可以确保数据始终保持可用状态，而且异地灾难恢复备份也可以被自动创建。一旦主集群数据不可用，可以使用备份数据。因为集群中所有内容的与副本中所存储的是一致的。而这些在文件系统中是几乎不可能发生的，它们需要克服繁琐的备份窗口和既漫长有艰难的备份还原操作。

- 自动负载平衡

  对象存储集群几乎是完全对称的。每个节点都是独立的，提供了集群的切入点，并运行相同的代码。这使得工作量可以平均分配到集群中的所有节点上，避免 NAS 和集群文件系统中常见的热节点问题的出现。自动负载均衡可以让 I/O 自动选择合理的节点，保证系统性能最大化。

- 常规移植

  在对象存储架构中，可以免却传统硬件移植或者大规模硬件升级的麻烦。对象存储结构只需要采用常规移植就可以实现。整个过程中都可以随时添加新的设备并会自动加载到集群中，而旧的设备单元也可以解除。

- 设备更换简单

  根据存档和法规要求，存储的数据需要保持数年。技术更新的成本和复杂性是一个需要考虑的重要因素，特别是连接到昂贵的专有硬件平台系统，对象存储提供平滑

设备更新工具。

- 更高的磁盘利用率

  相比块存储，对象存储可以提供更高的磁盘利用率。

- 高可用性和灾难恢复

  高可用性和灾难恢复内置在对象存储体系中。故障恢复并不需要专门 HA 配置来处理

HCP 集群及负载均衡

集群结构

HCP 采用集群架构，每两个存储节点(Storage Node)构成一个单元（CELL），至少每两个单元组成一个集群。一个本地的名称空间分配，被元数据控制和管理引擎资源监控，文件将被本地化的，在一个集群内部公平地分散。

数据存储负载均衡

当一个节点被加到一个集群中，集群的均衡策略被激活去维持每个数据和元数据的分布。物理地移动数据和元数据从已经存在的节点到一个新的节点，设法达到相同的利用率在一个集群内部的所有节点上。为了保持平台的均衡，每个节点要确保最佳的性能和扩展性即使在较少的数据和面临所有节点失效的情况下。内部为了实现数据负载均衡在进行数据迁移的过程是一个数据挪动密集的操作。在一个数据迁移中，确保数据保护策略是强制的和不导致 TPOF 违反。

前端访问负载均衡

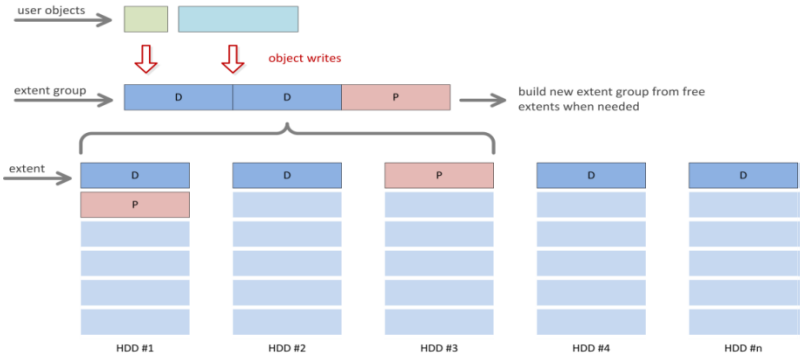
当应用程序作为前端访问对象存储，对象存储会根据负载情况分配分配处理阶段。请求的响应过程如下：

- 主机发出了一个请求去读一个文件
- 节点根据前端负载情况，分配处理具体节点。
- 节点传递请求去 Request Manager
- Request Manager 找出那一个元数据负责名称空间的详细区域。Request Manager 发出读的请求给指定的 Metadata Managers
- Metadata Manager 检查权限和标识出 Storage Manager 控制的对象被存储的磁盘
- Metadata Manager 传递回信息给 Request Manager。如果权限没有核查通过，那么这个请求被拒绝

- 假设这个权限是没问题的，Request Manager 联系 Storage Manager 传递给存储目标。如果这个控制的 Storage Managers 是本地的 Request Manager，它将最先获得这个请求 去最小化流量和相应。如果这个控制的 Storage Manager 是不正确的，那 Request Manager 将重复第三步
- 控制的 Storage Manager 传递这个对象到 Request Manager
- Request Manager 使主机满意于读的请求

HCP 的 EC 纠删码

HCP 对象型存储的 S10 存储节点采用产品化的硬件处理节点，并配置冗余双控制器，磁盘部署采用 20+6 的纠删码技术保护，保证在一个磁盘校验组中任意 6 块盘损坏或故障都不影响数据完整性和数据访问；S10 存储节点内采用冗余控制器与硬盘容量单元分离式设计，控制器任何部件故障都有冗余部件接管且不影响任何底层硬盘容量单元访问，也不会导致节点的硬盘数据重建，纠删码设计见下图：



存储纠删码设计特点为：

避免 RAID 重建的长时间和性能影响；

20+6 的 Extent Group 提供 77%的利用率和 15 个 9 的数据可用性；

不需要专有 Hot Spare，所有磁盘的可用空间可用于数据修复；

简化部署和管理，且易于扩展。

HCP 的 WORM 技和版本结合

HCP 支持通过 WORM（一写多读）技术，在保留期间内防止任何人对信息进行修改，满足内控、审计程序对信息原真性的要求。当审计人员需要检索、抽取特定的信息纪录时，能够通过专用的检索工具快速发现所有满足条件的信息。

当更改文件时，新文件被保存，旧文件成为一个新版本被自动保存。实现了数据版本级数据保护，可实现多达 10 个版本的文件版本保存。用户随时可以访问任意某一个版本文件。

HCP 重复删除和压缩技术

HCP 提供的全新数据重复数据删除和压缩服务来最大限度地利用其存储资源。重复数据删除也称为单一实例存储(SIS)。ChangHong 提供的这种新型存储服务远胜于其它同类竞争产品，它可以同时提供 hash 对比和二进制对比，因此能够确认对象是否是重复数据，从而避免了“hash collisions”，避免不同对象却具有相同的加密 hash 密钥的情况。HCP 的相关机制还能够让用户清楚地看到被删除的重复数据的数量，以及节约下来的总存储容量。

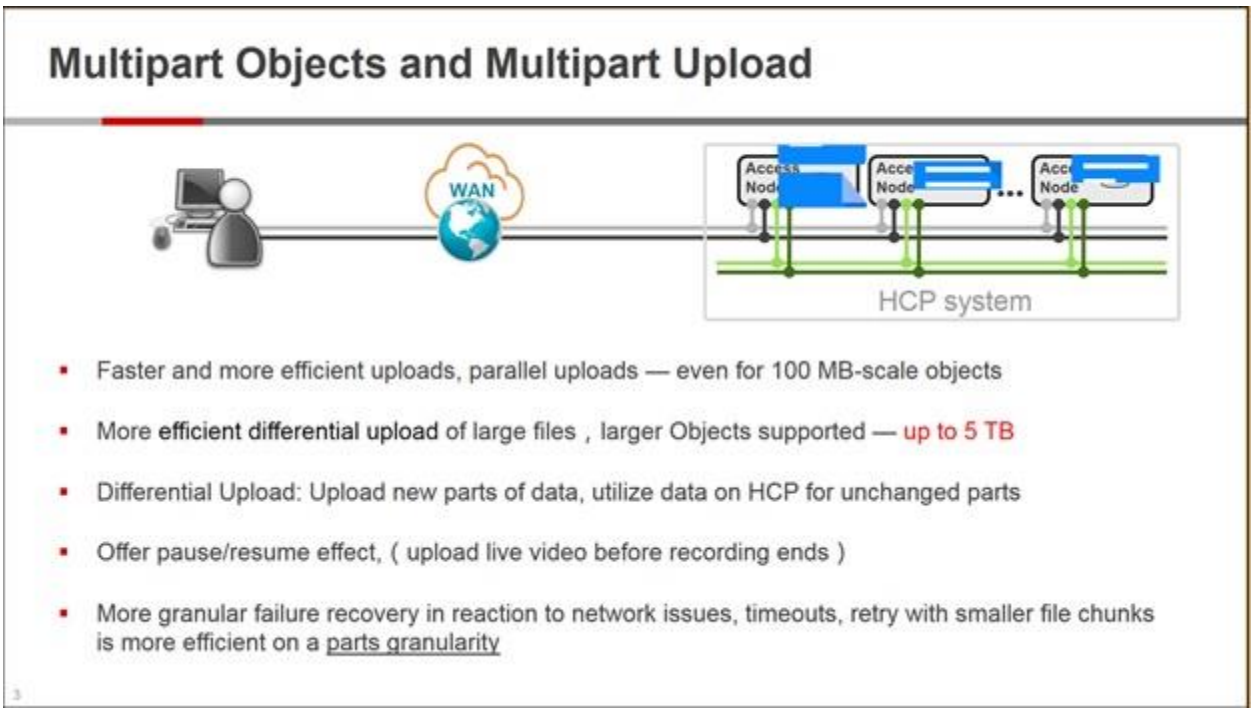
HCP 的 Duplication Elimination 是个后台的进程，HCP 会每天在合适的时间进行，客户可以在控制台进行干预它的启动和停止。该进程在意外停止后，还会持续运行。

HCP 自动检查和自动修复

HCP 提供自动检查和自动修复功能，即 HCP 会定期检查所有对象数据文件，通过检查每一个对象文件的 hash 密钥，查收是否有文件内容被修改或出现错误，如果有文件出现意外损坏或篡改，HCP 会自动修复这些文件，保证与保存时生成的 hash 密钥一致。

HCP 的断点续传和分段上传

HCP 优化了超大文件上传的性能，提供分段上传和断点续传功能。100MB 以上的文件如果开启分段上传会明显提升性能。



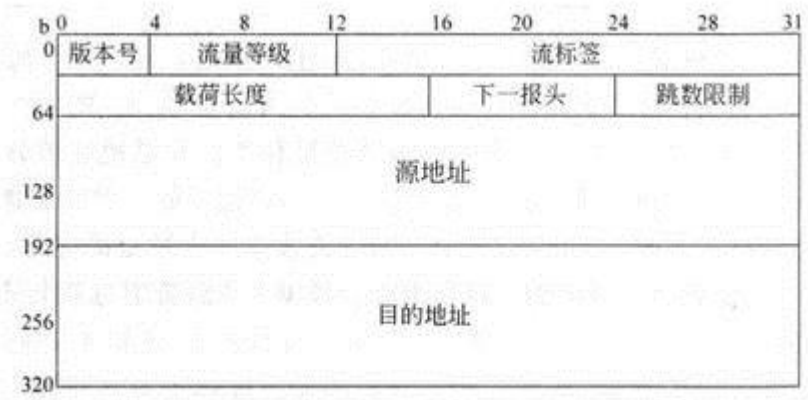
HCP 的 IPv6 支持

HCP 对象存储原生支持 Ipv6。由于 IPv4 最大的问题在于网络地址资源有限，严重制约了互联网的应用和发展。IPv6 的使用，不仅能解决网络地址资源数量的问题，而且也解决了多种接入设备连入互联网的障碍。

报文内容：

IPv6 报文的整体结构分为 IPv6 报头、扩展报头和上层协议数据 3 部分。IPv6 报头是必选报文头部，长度固定为 40B，包含该报文的基本信息；扩展报头是可选报头，可能存在 0 个、1 个或多个，IPv6 协议通过扩展报头实现各种丰富的功能；上层协议数据是该 IPv6 报文携带的上层数据，可能是 ICMPv6 报文、TCP 报文、UDP 报文或其他可能报文。

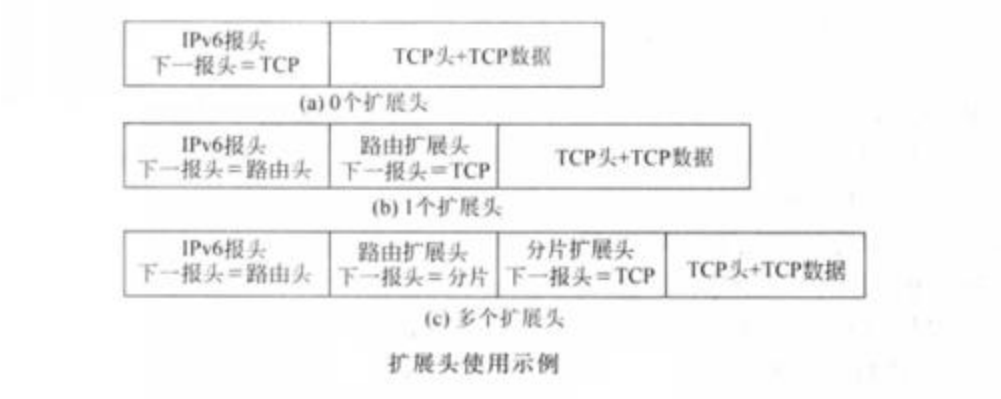
IPv6 的报文头部结构如图：



IPv6 报头结构

版本号	表示协议版本．值为 6
流量等级	主要用于 QoS
流标签	用来标识同一个流里面的报文
载荷长度	表明该 IPv6 包头部后包含的字节数，包含扩展头部
下一报头	该字段用来指明报头后接的报文头部的类型，若存在扩展头，表示第一个扩展头的类型，否则表示其上层协议的类型，它是 IPv6 各种功能的核心实现方法
跳数限制	该字段类似于 IPv4 中的 TTL，每次转发跳数减一，该字段达到 0 时包将会被丢弃
源地址	标识该报文的来源地址
目的地址	标识该报文的目的地址

扩展头部：IPv6 报文中不再有“选项”字段，而是通过“下一报头”字段配合 IPv6 扩展报头来实现选项的功能。使用扩展头时，将在 IPv6 报文下一报头字段表明首个扩展报头的类型，再根据该类型对扩展报头进行读取与处理。每个扩展报头同样包含下一报头字段，若接下来有其他扩展报头，即在该字段中继续标明接下来的扩展报头的类型，从而达到添加连续多个扩展报头的目的。在最后一个扩展报头的下一报头字段中，则标明该报文上层协议的类型，用以读取上层协议数据 ［10］ 。



扩展头使用示例

HCP 的数据可用性

HCP 对象存储通过 20+6 的纠删码、文件自愈、WORM、多版本、多副本等功能保证 15 个 9 的数据可用性。

HCP 的备份

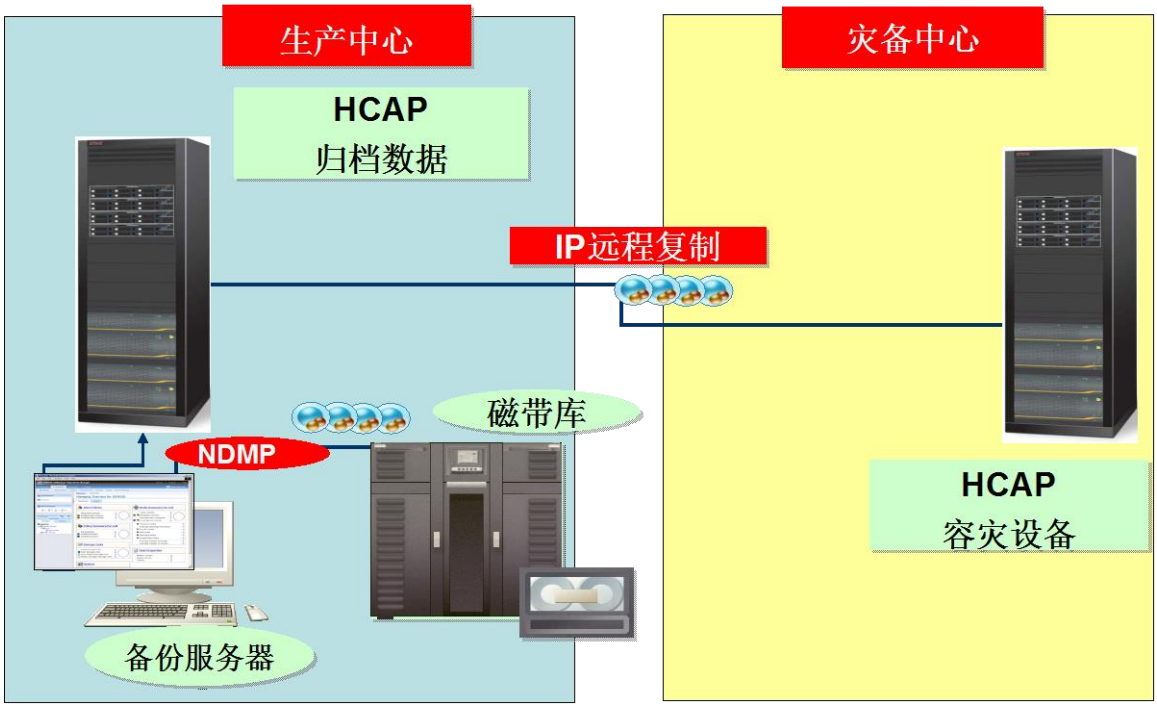
对于有些客户要求必须离线保存数据应用场景，HCP 支持 NDMP 协议的数据备份。利用开放的安全 PGP 加密，保证备份程序的安全性。

- NDMP 备份

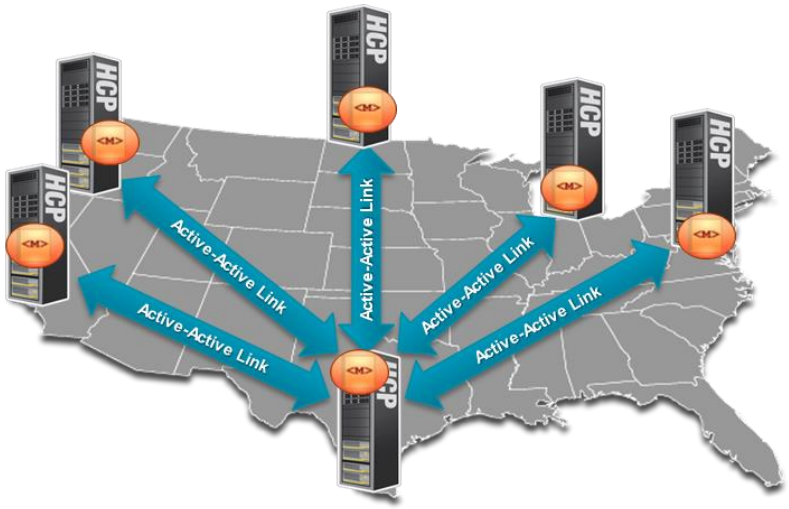
- NDMP v4 遵从数据服务器；提供标准的备份和恢复接口。提供灵活的备份工具选择如 CommVault Galaxy, NetBackup, 和很多其它的工具。
- 允许客户去维护离线、非现场备份
- 支持全量备份选项，包括多重和同时备份操作
- 已知保存的回复
- 可选的签名、压缩和加密备份集
- 基于便携的归档对象格式基础对于备份/恢复和复制

HCP 的容灾复制

复制是被设计利用不同地理区域，使用异步复制技术。这个适用于所有的数据、元数据和策略。一旦被启动它将是一个自动的服务。



HCP 对象型存储支持扩展多达 5 个对象存储跨区域部署，并可实现跨区域多活数据复制和跨区域纠删的不同保护方式，两种方式都支持在任意区域可在线访问多活数据，多活复制设计见下图：



多活复制的设计特点为：

HCP 的部署扩展到异地范围，让用户和应用可以通过最近的 HCP 存储就近访问需要的数据 ；

增加了数据可用性和宕机切换以及数据恢复的流畅性 ；

增加了产品性能和生产力，改善合作协同效率 。

四、丰富的数据集中和访问方式

本地数据访问

客户应用程序的非结构化数据已经保存在本地文件系统中，对于大量文件使用专用的网络文件系统（NAS）保存数据，以提高数据的安全性或数据处理性能。流行的网络文件系统主要包括 NFS 和 CIFS 两种，它们分别针对不同操作系统而设计。LIUNX 客户端主要使用 NFS 文件系统，WINDOWS 客户端主要使用 CIFS 文件系统。

HCP 为了满足应用程序直接保存数据到网络文件系统的需要，也支持 CIFS 和 NFS 网络文件系统，做到了应用程序不需要修改就可以保存到 HCP 内容平台，传统文件系统的数据的获取通过目录和文件名方式找到文件并最终获取到数据，HCP 使用基于对象数据存储与管理方法，每个对象被保存命名空间中，每个对象分配唯一的 ID。应用程序通过文件名和目录访问 HCP 内资源，HCP 接到请求后，自动转换为 ID，通过 ID 访问到最终数据。因此

HCP 突破了传统文件系统处理性能限制，每套 HCP 可以保存 40PB 的数据。同时由于数据被保存到网络文件系统中，数据可以被同时被其它应用程序所访问，实现数据共享。

传统文件系统存放形式完全不同的对象型存储方式，当增加一个文件到 HCP 中，HCP 自动产生 metadata 为这个文件并与这个文件封装在一起作为一个对象。当你显示这个内容数据时，HCP 文件系统显示每一个对象就像一个文件夹。一个文件包含原始的数据文件（与这个文件具备相同的名字），包扩文件的扩展名。当你打开这个目标文件，它看起来与原来的保存的文件一样。

当增加一个文件到 HCP 中，HCP 自动产生 metadata 为这个文件并与这个文件封装在一起作为一个归档目标。当你显示这个内容数据，HCP 文件系统显示每一个目标就像一个文件夹。一个文件夹包含原始的数据文件（与这个文件具备相同的名字），包扩文件的扩展名。当你打开这个目标文件，它看起来与原来的保存的文件一样。

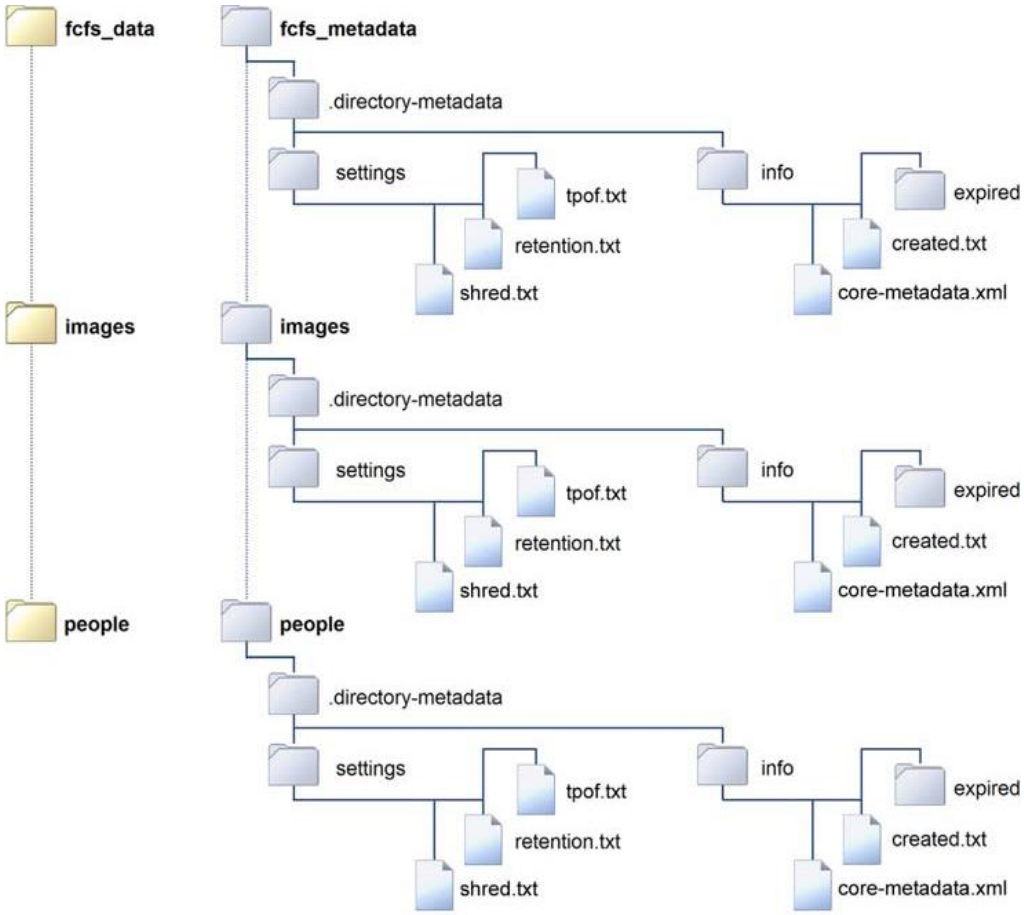
像文件一样，目录也被像一个目标一样。这些目标，无论如何，只包含 metadata ，所以 HCP 文件系统显示这个目录的名字和目录 meta 文件并没有相应的文件。当你显示这个归档通过网关，目录显示为一个标准的目录层次。

HCP 有两个最顶层的目录或安装点。

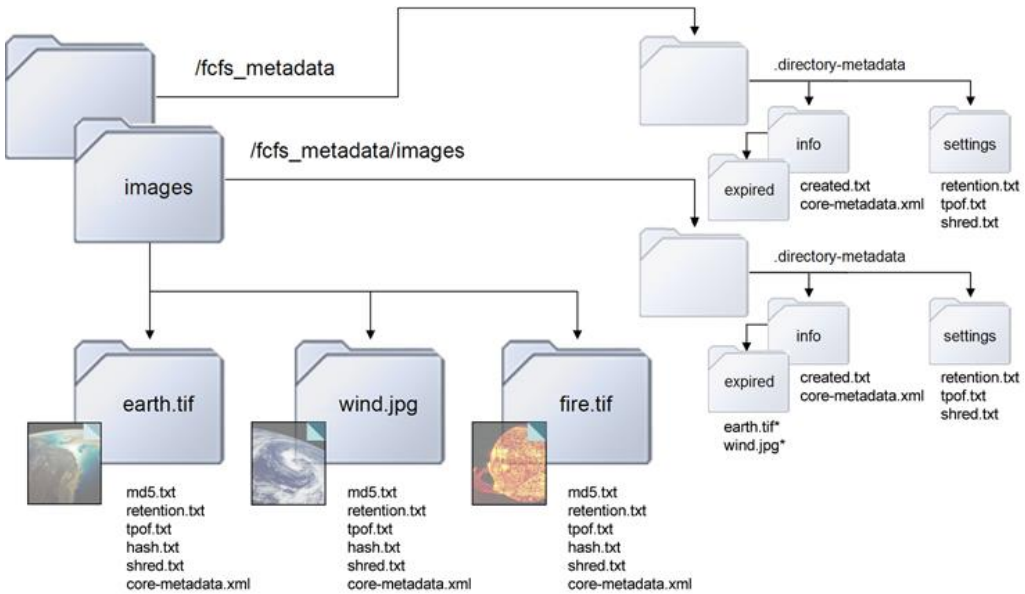
- fcfs_data 是目录的最上端包括所有的数据
- fcfs_metadata 目录的最上端包括所有的归档的 meta 文件。

HCP 存储用户可访问的 meta 数据对于每一个你添加到 HCP 的数据文件，这个目录结构在 fcfs_metadata 下平行于 fcfs_data 的目录结构。

如下图可以清晰的看到 HCP 的目录文档结构：



对于每个文件，HCP 又会针对性的产生相应的 meta 文件。如下图：



图形化及命令行接口

HCP 提供基于通过操作系统的系统工具，提供同时满足广域和局域网环境下数据集中工具。

若应用程序分布在不同地点，不同地点之间通过广域网络连接。如此环境下，使用网络文件系统的方式不能满足需要。HCP 开发了基于 JAVA 的客户端工具 HCP DM (HCP Data Migrator)。利用此工具实现数据在网络中任何位置，都可以满足数据上传到数据中心，保存到 HCP 内容云中。

HCP DM 功能：

- 复制一个或多个对象
- 复制某一个版本对象
- 删除一个或多个对象
- 删除一个对象的所有版本
- 运行特别权限删除保护期内对象
- 打开对象，包括所有版本对象
- 本地文件系统文件或目录改名
- 查询对象属性
- 建立空目录
- 增加，改变或删除对象的 Cutstom metadata
- 增加，改变或删除对象的 ACL

访问接口:

- 图形化接口
- 命令行接口

HCP 开发接口 API

为了满足应用程序与 HCP 集成的需要，HCP 全面支持 HTTP REST 协议。通过 HTTP REST 对 HCP 的所有资源进行描述，通过不同 URI 进行标识，应用程序通过 URI 来获取资源。此种实现方式类似于互联网的运作方式，在互联网的一系列的网页，每个网页都有自己的 URL，浏览器通过不同 URL 访问不同的互联网资源。

资源描述 URI

HCP 把相关资源分配成几个资源组，分别实现不同功能。

- REST：用于 HCP 数据访问
- PROC：HCP 命名信息
- MAPI：管理 API 和数据收集
- QUERY：基于元数据查询接口
- fcfs_data：缺省命名空间数据访问
- fcfs_metadata：缺省命名空间和用户元数据

资源访问方法：

对于 HCP 内部资源的操作包括获取、创建、修改和删除等操作，这些操作对应 HTTP 协议提供的 GET、POST、PUT 和 DELETE 方法。

PUT：增加对象，版本，目录和客户元数据

Actions	• *Add new object* • *Add new version* • *Add empty directory* • *Add custom metadata*			
Inputs	Input	Type	Valid Values	Require or Optional
	hcp-ns-auth token	Cookie		Required - always
	Location of object or custom metadata	URL		Required - when adding an object or object version
	Location to put object or directory	URL	URL	Required
	Type	URL parameter	directory custom-metadata	Optional - use directory when adding a directory; use custom-metadata when setting

				custom metadata
	Retention	URL parameter	retention value	Optional
	Shred	URL parameter	true, false	Optional
	Hold	URL parameter	true, false	Optional
	Index	URL parameter	true, false	Optional
Examples	Add a new object: curl -iT wind.jpg -b hcp-ns-auth=<token> "http://financing.changhong.cluster-foo.ChangHong.com/rest/dir1/wind.jpg" Add a new object with retention and shred settings: curl -iT wind.jpg -b hcp-ns-auth=<token> "http://financing.changhong.cluster-foo.ChangHong.com/rest/dir1/wind.jpg?retention=C+SEC17A&shred=true" Add an empty directory: curl -iX PUT -b hcp-ns-auth=<token> “http://financing.changhong.cluster-foo.ChangHong.com/rest/dir1?type=directory” Add custom metadata to an object: curl -iT metadata.xml -b hcp-ns-auth=<token> "http://financing.changhong.cluster-foo.ChangHong.com/rest/dir1/wind.jpg?type=custom-metadata"			

GET：获得数据

- 对象，版本，版本列表，目录，Custom Metadata
- 命名空间列表，命名空间授权，命名空间状态，命名空间数据保存类别

Actions	• *Retrieve object or version*

	• *Retrieve a list of all versions for an object* • *Retrieve custom metadata for object or version* • *Retrieve directory list* • *Retrieve list of namespaces* • *Retrieve namespace permissions* • *Retrieve namespace statistics* • *Retrieve namespace retention classes*			
Inputs	**Input**	**Type**	**Valid Values**	**Required or Optional**
	hcp-ns-auth token	Cookie		Required
	Location of object or directory	URL	URL	Required
	Type	URL parameter	custom-metadata	Optional - use custom-metadata to retrieve custom metadata for an object
	Version	URL parameter	Version ID, list	Optional - specify a version ID to retrieve a version of an object or custom metadata for that version.; use LIST to retrieve a list of version for an object
Examples	Retrieving an object: curl -b hcp-ns-auth=<token> "http://financing.changhong.cluster-foo.ChangHong.com/rest/dir1/wind.jpg" > wind.jpg Retrieving a directory listing: curl -b hcp-ns-auth=<token> "http://financing.changhong.cluster-			

	foo.ChangHong.com/rest/dir1″ > dir.xml Retrieving custom metadata XML file for an object: curl -i -b hcp-ns-auth=<token> “http://financing.changhong.cluster-foo.ChangHong.com/rest/dir1/wind.jpg?type=custom-metadata” > custom-metadata.xml

DELETE：删除对象，版本，目录和 Custom metadata

Actions	• *Delete object or latest object version* • *Purge an object (incl. all versions)* • *Delete an empty directory* • *Deleting custom metadata file for an object or object version*			
Inputs	Input	Type	Valid Values	Required or Optional
	hcp-ns-auth token	Cookie		Required
	Location of object or directory	URL	URL	Required
	Type	URL parameter	custom-metadata	Optional - use custom-metadata to delete custom metadata for an object
	purge	URL parameter	true	Optional - use to purge all versions of an object
	privileged	URL parameter	true	Optional - use for privileged delete or privileged purge
	reason	URL parameter	Text	Required if *privileged*=true. Error (400) if account does not

				have privileged permission.
Examples	Deleting an object: curl -iX DELETE -b hcp-ns-auth=<token> “http://financing.changhong.cluster-foo. changhong.com/rest/dir1/wind.jpg” Privileged deleting an object: curl -iX DELETE -b hcp-ns-auth=<token> -d "privileged=true" -d "reason=Person left the company" “http://financing.changhong.cluster-foo.changhong.com/rest/dir1/wind.jpg” Purging an object (and all its versions): curl -iX DELETE -b hcp-ns-auth=<token> -d "purge=true" http://financing.changhong.cluster-foo.changhong.com/rest/dir1/wind.jpg Deleting an empty directory: curl -iX DELETE -b hcp-ns-auth=<token> “http://financing.ChangHong.cluster-foo.ChangHong.com/rest/dir1” Deleting custom metadata for an object: curl -iX DELETE -b hcp-ns-auth=<token> "http://financing.ChangHong.cluster-foo.ChangHong.com/rest/dir1/wind.jpg?type=custom-metadata"			

HEAD：判断对象是否存在，版本，目录和 Cutstom metatadata

Actions	• *Determine existence of object or version* • *Determine existence of directory* • *Determine existence of custom metadata*			
Inputs	Input	Type	Valid Values	Required or Optional
	hcp-ns-auth	Cookie		Required

	token			
	Location of object or directory	URL	URL	Required
	type	URL parameter	custom-metadata	Optional - use *custom-metadata* to determine existence of custom metadata
	version	URL parameter	Version ID	Optional - specify a version ID to determine existence of a specific object version
Examples	Determine existence of an object: curl -I -b hcp-ns-auth=<token> “http://financing.ChangHong.cluster-foo.ChangHong.com/rest/dir1/wind.jpg” Determine existence of a directory: curl -I -b hcp-ns-auth=<token> “http://financing.ChangHong.cluster-foo.ChangHong.com/rest/dir1” Determine existence of custom metadata: curl -iI -b hcp-ns-auth=<token> “http://financing.ChangHong.cluster-foo.ChangHong.com/rest/dir1/wind.jpg?type=custom-metadata” Determine existence of custom metadata on a specific version of the object: curl -iI -b hcp-ns-auth=<token> “http://financing.ChangHong.cluster-foo.ChangHong.com/rest/dir1/wind.jpg?type=custom-metadata&version=81820007586305”			

POST：改变一个或多个系统 metadata

Actions	Setting single or multiple system metadata values			
Inputs	Input	Type	Valid Values	Required or Optional
	hcp-ns-auth token	Cookie		Required
	Location of object	URL	URL	Required
	hold	URL parameter	true, false	Optional - use to set hold on an object
	index	URL parameter	true, false	Optional - use to set search indexing for an object
	retention	URL parameter	Retention value	Optional - use to set retention value for an object
	shred	URL parameter	true, false	Optional - use to set shredding on an object
Examples	Turn off indexing functionality for an existing object: curl -i -b hcp-ns-auth=<token> "http://financing.ChangHong.cluster-foo.ChangHong.com/rest/dir1/wind.jpg" -d "index=false" Turn on shredding functionality for an existing object: curl -i -b hcp-ns-auth=<token>			

	“http://financing.ChangHong.cluster-foo.ChangHong.com/rest/dir1/wind.jpg” -d “shred=on”

HCP 使用 REST 的好处

- REST 协议本身的无状态性，保证 HCP 节点同时处理大量请求，提高 HCP 处理能力
- REST 可以利用缓存 Cache 来提高响应速度
- 浏览器即可作为客户端，简化软件需求
- 相对于其他叠加在 HTTP 协议之上的机制，REST 的软件依赖性更小
- REST 不需要额外的资源发现机制
- REST 保证软件技术演进中的长期的兼容性

五、跨系统元数据搜索

HCP 的索引式搜索引擎是基于行业领先的索引和搜索的技术，它被嵌入的内容云平台中。它提供高效的数据读取，数据索引，丰富结果处理手段。

HCP 帮助企业有效地控制非结构化数据，这些数据包含在传统的内容管理系统之内，填补了网络连接存储（NAS）设备上空白，对数据进行内容管理、内容索引和搜索的空白。通过与目录服务集成确保数据访问权限，保证数据访问安全，使用户可以搜索他们有权查看的数据。

HCP 的搜索将预处理技术与分布式技术结合，通过对查询结果的预处理来获得较高的响应速度，使用缓存技术将一些经常被查询的词的查询结果保存在内存中，当用户输入包含多个词的查询请求时，只需要对这些查询请求进行合并和重新排序即可。同时，查询利用分布式技术可将查询任务分担到多台服务器去并行进行，从而提高响应速度。

数据搜索技术

如此巨大的信息聚集必然导致有效信息获取的难度增强和垃圾信息量增加,业务需要快速精准地获取有用信息的工具。目前主流的搜索技术有数据库技术和搜索引擎技术

数据库搜索技术

数据库搜索依靠人工发现信息，并依靠人员自身的知识对信息进行分类、提取主题词、

建立关键字索引和目录分类体系。

基于数据库搜索技术，处理的数据规模有限，需要大量人工操作，索引建立的效率却比较低，信息更新也比较慢。在互联网上，基于数据库式搜索的例子有 Yahoo 和新浪分类目录等。

索引式搜索引擎

对于大量非结构化数据，数据库搜索技术已经不能满足业务需要，索引式搜索引擎已经取代传统数据查询检索方式。索引式搜索引擎依靠一个被称作蜘蛛（Spider）或机器人（Robot）的程序，根据特定网络协议和原则，自动地获取信息来建立索引，并采用一定得方法对索引库进行更新，以确保索引库与信息的实时对应。这类搜索引擎主要依靠程序自动地搜集和维护信息，从而将设计人员，开发人员，录入人员解放出来，同时索引库可以比较大，实时性也更强。目前， Google 和百度都在使用此搜索技术来满足用户检索互联网信息需要。

数据库搜索和索引式搜索比较

数据库技术和搜索引擎技术都是可以实现数据搜索。但是，它们针对不同应用场景。数据库技术研究的是如何存储和管理数据，数据库技术所涉及的具体内容主要包括：

- 通过对数据的统一组织和管理，按照指定的结构建立相应数据库；
- 数据库实现对数据库中的数据进行添加、修改、删除、处理、分析、理解、报表

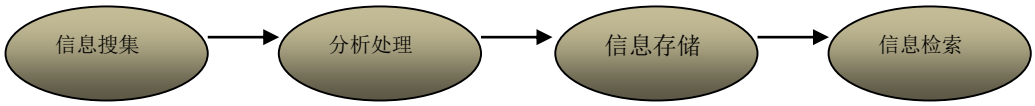
搜索引擎技术研究的是如何把非结构化的的信息收集，整理、分类、索引以产生索引库，根据关键词在索引器形成的倒排表中进行检索。针对两种搜索技术比较

	搜索引擎	数据库搜索
定位	非结构化数据	结构化数据
检索数据量	大规模	少量数据
性能	快。数据量增加，访问速度没有明显变化	较慢，特别是有大量返回结果。
搜索范围	全文检索	指定关键字

检索条件	任意数量	应用设计几个字段，数据库直接检索支持字段数量有限

索引式数据处理流程

首先搜集目标信息，然后将信息进行分析处理，并按照一定的数据结构进行存储，最后用户从这些被存储的数据中检索出有用的信息。



索引式搜索组成

索引库（Index Repository）

索引库是信息存储的地方，这里的信息已经变成倒排结构。检索器通过查询索引库获得查询最终查询的内容。

索引器

索引器(Indexer Engine)是生成索引的模块，将处理过后的信息包装成文档交给索引器，索引器会在索引库中建立该文档的索引（也就是倒排结构）。

检索器

检索器（Searcher）是信息查询的模块。当用户提交查询词后，检索器会对查询词进行分词等处理，并生成查询请求（Query）,然后在索引库中进行查询，并将查询所得结果以一定得格式呈现给用户。

读取模块

数据读取模块负责，是索引擎的一个重要模块，它从各个文件中读取文件并提取出文字信息，然后交给索引器建立索引

文档解析模块

文档解析模块用于解析特定的文件，提取出文字信息并建立文档对象，然后交给索引器进行处理。网页处理模块用于对“网络蜘蛛”（Spider）获取的网页进行解析，提取出文字信息，建立文档对象后交给索引器处理。

用户访问接口

用户界面是面向最终用户的，它从用户角度出发，提供方便的查询界面，并将查询结果按照用户的需求予以显示。

- 图形化接口
  用户通过浏览器方式在任意地点任意时间都可以访问进行检索
- 开发API
  提供开发API接口，与业务系统集成
- 访问权限管理
  通过权限管理，与文档访问权限集成，实现用户访问权限控制，做到不同人搜索到结果不同。

HCP 元数据检索接口 API 实例

基于操作的查询请求的主体是由 XML 或 format.xml JSON 请求体基于操作的查询。

基于操作的 XML 请求主体查询

包含一个 query request 出入，除非请求所有可以获得信息的操作入口,所有其他条目都是可选的。请求主体有如下格式。每个条目等级级别可以指定任何顺序：

<queryRequest>

<operation>

<count>number-of-results</count>

<lastResult>

<urlName>object-url</urlName>

<changeTimeMilliseconds>change-time-in-milliseconds.index

</changeTimeMilliseconds>

<version>version-id</version>

</lastResult>

<objectProperties>comma-separated-list-of-properties

</objectProperties>

<systemMetadata>

<changeTime>

<start>start-time-in-milliseconds</start>

<end>end-time-in-milliseconds</end>

</changeTime>

<directories>

<directory>directory-path</directory>

...

</directories>

<indexable>(true|false)</indexable>

<namespaces>

<namespace>namespace-name.tenant-name</namespace>

...

</namespaces>

<replicationCollision>(true|false)</replicationCollision>

<transactions>

<transaction>operation-type</transaction>

...

</transactions>

</systemMetadata>

<verbose>(true|false)</verbose>

</operation>

</queryRequest>

基于操作的查询 JSON 请求

JSON 请求体的操作为基础的查询必须包含一个未命名的顶级条目，除非请求所有可用信息。所有其他条目都是可选的。JSON 请求体具有以下格式。每个条目层次级别可以在任何顺序：

```
{
"operation": {
"count":"number-of-results",
"lastResult": {
"urlName":"object-url",
"changeTimeMilliseconds":"change-time-in-milliseconds.index",
"version":version-id
},
"objectProperties":"comma-separated-list-of-properties",
"systemMetadata": {
"changeTime": {
"start":start-time-in-milliseconds,
"end":end-time-in-milliseconds
},
```

```
"directories": {
"directory":["directory-path",...]
},
"indexable":"(true|false)",
"namespaces": {
"namespace":["namespace-name.tenant-name",...]

},
"replicationCollision":"(true|false)",
"transactions": {
"transaction":["operation-type",...]
}
},
"verbose":"(true|false)"
}
}
```

客户化元数据查询示例

这里是一个元数据查询请求实例，该请求将检索元数据的所有对象：

•首先是是由租户拥有的命名空间

•同时有自定义元数据，它包含一个名为部门的元素查询 JSON 格式使用一个 XML 请求和请求的结果。此外，在结果集对象的基本信息，这请求返回每个对象的内容和保留设置结果集。该请求还指定了结果集中的对象被列为以改变时间为基础的反向时间顺序。

备注：查询文件是 XML 格式的 accounting.xml

```
<queryRequest>
<object>
<query>customMetadataContent:
"department.Accounting.department"
</query>
<objectProperties>shred,retention</objectProperties>
<sort>changeTimeMilliseconds+desc</sort>
</object>
</queryRequest>
```

【命令行方式】

利用 Curl 工具提交命令

```
curl -k -H "Authorization: HCP bXl1c2Vy:3f3c6784e97531774380db177774ac8d"
-H "Content-Type: application/xml" -H "Accept: application/json"
-d @Accounting.xml "https://europe.hcp.example.com/query?prettyprint"
```

【Python 方式】

```
import pycurl
import os
curl = pycurl.Curl()
# Set the URL, command, and headers
curl.setopt(pycurl.URL, "https://europe.hcp.example.com/" +
"query?prettyprint")
curl.setopt(pycurl.SSL_VERIFYPEER, 0)
curl.setopt(pycurl.SSL_VERIFYHOST, 0)
```

```
curl.setopt(pycurl.POST, 1)
curl.setopt(pycurl.HTTPHEADER,
["Authorization: HCP bXl1c2Vy:3f3c6784e97531774380db177774ac8d",
"Content-Type: application/xml", "Accept: application/json"])
# Set the request body from an XML file
filehandle = open("Accounting.xml", 'rb')
curl.setopt(pycurl.UPLOAD, 1)
curl.setopt(pycurl.CUSTOMREQUEST, "POST")
curl.setopt(pycurl.INFILESIZE,
os.path.getsize("Accounting.xml"))
curl.setopt(pycurl.READFUNCTION, filehandle.read)
curl.perform()
print curl.getinfo(pycurl.RESPONSE_CODE)
curl.close()
filehandle.close()
```

【请求报文】

```
POST /query?prettyprint HTTP/1.1
Host: europe.hcp.example.com
Authorization: HCP bXl1c2Vy:3f3c6784e97531774380db177774ac8d
Content-Type: application/xml
Accept: application/json
Content-Length: 192
```

【返回结果】

HTTP/1.1 200 OK

Server: HCP V7.0.0.16

Transfer-Encoding: chunked

JSON response body

To limit the example size, the JSON below shows only one object in the result set.

{"queryResult:

{"query":

{"expression":"customMetadataContent:

"department.Accounting.department""},

"resultSet":[

{"version":84689494804123,

"operation":"CREATED",

"urlName":"https://finance.europe.hcp.example.com/rest/presentations/Q1_2012.ppt",

"changeTimeMilliseconds":"1334244924615.00",

"retention":0,

"shred":false},

.

.

.

],

"status":{

"message":"",

"results":12,

"code":"COMPLETE"}

}

}

Custom metadata file for the Q1_2012.ppt object

<?xml version="1.0">

<presentation>

<presentedBy>Lee Green</presentedBy>

<department>Accounting</department>

<slides>23</slides>

<date>04-01-2012</date>

</presentation>

总之，通过客户化数据查询，可以在跨系统的对象化存储数据中，迅速找到用户希望的信息并快速应用。

六、对象存储 与传统存储

有大量的基于块和基于文件的存储系统可供选择，一个明显的问题是，我们为什么需要另外一种存储技术呢？块和文件都是成熟且经过验证的，所以也许看起来好像他们可以增强以满足日益增长的分布式云计算生态系统的需求。

块存储

基于块的存储系统，磁盘块通过底层存储协议访问，像SCSI命令，开销很小而且没有其它额外的抽象层。这是访问磁盘数据最快的方式，所有高级别的任务，像多用户访问、共享、

锁定和安全通常由操作系统负责。换句话讲，基于块的存储关心所有底层的问题，但其它事情都要依靠高层的应用程序实现。所有的对象存储拥有基于块存储的节点，利用对象存储软件集合提供所有其它的功能。

文件存储（NAS）

基于块的存储系统是对象存储系统的补充，而基于文件的存储系统一般被认为是直接的竞争者。横向扩展的NAS系统的关键属性就是扩展性，对象存储也是这样，通过增加节点实现水平扩展。但由于NAS系统是基于分层文件结构的有限的命名空间，它们对于有着接近无限扩展能力的、具有扁平结构的纯对象存储来讲，所受的约束更多，对象存储仅受到对象ID的位数限制。尽管限制多多，但横向扩展的NAS系统仍然具备对象存储的诸多特性，而其欠缺的功能，像对于云存储（REST）协议的支持，厂商们正在快速的完善中，这样他们就可以把横向扩展的NAS系统划归到对象存储的类别中了。

内容寻址存储

在底层，内容寻址存储以带客户化元数据的对象方式进行文件的存储，文件的访问通过数字对象标识。通过具备强大法规遵从功能的磁盘归档空间构架，CAS通常部署在数据中心内部，所以它不需要云的功能。例如：通过互联网的分布式访问和多租户。

对象存储

大多数其他的对象存储厂商都从头开发了其对象存储的软件。通过x86架构节点，每个存储节点同时可提供计算和存储资源，可以通过简单的增加节点实现线性的容量和性能扩展。对象存储软件通常不关心硬件，而且由松散耦合的服务组成：展示层通过HTTP协议（REST）处理与客户端的接口，而且可选择传统的文件系统协议；元数据管理层管理在哪里存放数据对象、如何保护数据对象、如何分布在存储节点上；存储目标层是与存储节点的接口。

对象存储与传统存储比较

	Object Storage	File-Base Storage	Block-Based Storage
存储单元	对象，具有用户数据	文件	块

	的文件。		
自动修复	支持不可篡改，自动检查和修复数据	不支持	不支持
多版本	不支持。更新文件可以创建新对象版本	不支持	不支持
协议	通过 HTTP 的 REST 和 SOAP、CIFS、NFS、S3、SWIFT	CIFS 和 NFS	SCSI,FC,STAT
Metadata 支持	支持用户 metadata	依赖文件系统属性	依赖文件系统属性
应用场景	PB 级别文件数据和云存储，免备份保护	TB 级单一应用文件数据	传统数据和随机改变数据
服务方式	跨区域，分布集中，提供众多分支、用户服务	仅本地使用	仅本地使用