# 2@欢迎来到 Ceph 世界 — Ceph Documentation

http://docs.ceph.org.cn/

# 欢迎来到 CEPH 世界

Ceph 独一无二地在一个统一的系统中同时提供了**对象、块、和文件存储功能**。

|  |  |  |
| --- | --- | --- |
| CEPH 对象存储  * REST 风格的接口 * 与 S3 和 Swift 兼容的 API * S3 风格的子域 * 统一的 S3/Swift 命名空间 * 用户管理 * 利用率跟踪 * 条带化对象 * 云解决方案集成 * 多站点部署 * 灾难恢复 | CEPH 块设备  * 瘦接口支持 * 映像尺寸最大 16EB * 条带化可定制 * 内存缓存 * 快照 * 写时复制克隆 * 支持内核级驱动 * 支持 KVM 和 libvirt * 可作为云解决方案的后端 * 增量备份 | CEPH 文件系统  * 与 POSIX 兼容的语义 * 元数据独立于数据 * 动态重均衡 * 子目录快照 * 可配置的条带化 * 有内核驱动支持 * 有用户空间驱动支持 * 可作为 NFS/CIFS 部署 * 可用于 Hadoop （取代 HDFS ） |
| 详情见 [Ceph 对象存储](http://docs.ceph.org.cn/radosgw)。 | 详情见 [Ceph 块设备](http://docs.ceph.org.cn/rbd/rbd)。 | 详情见 [Ceph 文件系统](http://docs.ceph.org.cn/cephfs)。 |

它可靠性高、管理简单，并且是开源软件。 Ceph 的强大可以改变您公司的 IT 基础架构和海量数据管理能力。想试试 Ceph 的话看[入门](http://docs.ceph.org.cn/start)手册；想深入理解可以看[体系结构](http://docs.ceph.org.cn/architecture)一节。

# “Ceph浅析”系列之一——前言

- ThinkDiff - 博客园

http://www.cnblogs.com/bodhitree/p/4831626.html

开源技术专家章宇同学（@一棹凌烟）在C3沙龙分享过Ceph之后，最近来了劲头，一口气写了一系列《Ceph浅析》的博文，共8篇：

“Ceph浅析”系列之一——前言  
“Ceph浅析”系列之二——Ceph概况  
“Ceph浅析”系列之三——Ceph的设计思想  
“Ceph浅析”系列之四——Ceph的结构  
“Ceph浅析”系列之五——Ceph的工作原理及流程  
“Ceph浅析”系列之六——Ceph与OpenStack  
“Ceph浅析”系列之七——关于Ceph的若干想法  
“Ceph浅析”系列之八——小结  
这一系列文章对于想要初步了解Ceph文件系统的同学而言是相当不错的入门资料。

作者资料：章宇于2002年及2007年分别于清华大学电子工程系获得学士及博士学位，其后一直从事计算机系统领域的研究与开发工作，目前供职于华为技术有限公司云操作系统部门，从事OpenStack相关工作。出于工作原因和个人兴趣，作者陆续关注了一些开源项目，主要包括：KVM/QEMU，libvirt，virt-mamager，OpenStack，Open vSwitch，Ceph，Zabbix等。

Ceph无疑是目前人气最高的开源存储项目之一。因此，笔者近期也花了一些时间对其进行调研。基于对现有资料文档的学习思考，以及由此而来的心得体会，笔 者将用一系列文章对Ceph进行介绍与分析。文章内容大致涵盖Ceph的产生背景、设计思想、技术实现、主要特点、与OpenStack的联系、与 Swift的比较等等。

由于条件及精力所限，笔者并未对Ceph的源代码进行深入分析，也没有Ceph扩展开发或者部署运维的一手经验。因此，这些文章的内容必定有欠深入。之所以希望有所记录并加以分享，其目的在于以下几点：

梳理记录学习心得  
整理编写中文资料  
身体力行推动开源  
有鉴于此，希望这些文章能够为打算学习Ceph的朋友提供一个入手的思路和参考，也帮助已经初步了解Ceph的朋友查遗补缺。对于那些精通Ceph的大牛，则欢迎各位不吝赐教。对那些希望找到具体开发和部署指导与技巧的兄弟们，只好对不住各位了 :-)

这些文章中的内容全部来自公开的资料和文档，并尽可能注明出处。如有遗漏，欢迎指出。

如需转载或引用文章内容，敬请注明。请勿在未获得笔者授权的情况下将文章内容用于商业目的。谢谢。

2007年于山东大学计算机科学与技术本科专业毕业，曾在华为多年从事网络设备管理系统软件的研发和运维工作；目前供职于虎牙（YY）直播云平台组，专注于Ceph 分布式存储的架构、开发、运维等工作。 欢迎您来和我做技术交流。 REPO ：https://github.com/BodihTao 微信号：cn-qiyu

# “Ceph浅析”系列之二——Ceph概况

- ThinkDiff - 博客园 http://www.cnblogs.com/bodhitree/p/4831638.html

本文将对Ceph的基本情况进行概要介绍，以期读者能够在不涉及技术细节的情况下对Ceph建立一个初步印象。

1. 什么是Ceph？

Ceph的官方网站Ceph.com上用如下这句话简明扼要地定义了Ceph：

“Ceph is a unified, distributed storage system designed for excellent performance, reliability and scalability.”

也即，Ceph是一种为优秀的性能、可靠性和可扩展性而设计的统一的、分布式的存储系统。应该说，这句话确实点出了Ceph的要义，可以作为理解 Ceph系统设计思想和实现机制的基本出发点。在这个定义中，应当特别注意“存储系统”这个概念的两个修饰词，即“统一的”和“分布式的”。

具体而言，“统一的”意味着Ceph可以一套存储系统同时提供对象存储、块存储和文件系统存储三种功能，以便在满足不同应用需求的前提下简化部署和 运维。而“分布式的”在Ceph系统中则意味着真正的无中心结构和没有理论上限的系统规模可扩展性。在实践当中，Ceph可以被部署于上千台服务器上。截 至2013年3月初，Ceph在生产环境下部署的最大规模系统为Dreamhost公司的对象存储业务集群，其管理的物理存储容量为3PB。

**2. 为什么要关注Ceph？**

事实上，Ceph并不是一个刚刚出现的开源项目。与此相反，从最初发布到逐渐流行，Ceph走过了七年以上的漫长路程。笔者以为，之所以应当对Ceph加以了解，其原因大致有两个方面：

首先，Ceph本身确实具有较为突出的优势。

Ceph值得一提的优势颇多，包括统一存储能力、可扩展性、可靠性、性能、自动化的维护等等。本质上，Ceph的这些优势均来源于其先进的核心设计 思想，笔者将其概括为八个字——“无需查表，算算就好”。基于这种设计思想，Ceph充分发挥存储设备自身的计算能力，同时消除了对系统单一中心节点的依 赖，从而实现了真正的无中心结构。基于这一设计思想和结构，Ceph一方面实现了高度的可靠性和可扩展性，另一方面保证了客户端访问的相对低延迟和高聚合 带宽。通过后续内容的介绍，读者可以看到，Ceph几乎所有优秀特性的实现，都与这个核心设计思想有关。

其次，Ceph目前在OpenStack社区中备受重视。

OpenStack是目前最为流行的开源云操作系统。而据笔者观察，Ceph之所以在近一两年间热度骤升，其最为有力的推动因素就是 OpenStack社区的实际需求。目前而言，Ceph已经成为OpenStack社区中呼声最高的开源存储方案之一，其实际应用主要涉及块存储和对象存 储，并且开始向文件系统领域扩展。这一部分的相关情况，在后续文章中也将进行介绍。

**3. Ceph的产生与发展**

通常而言，开源项目的来源有三：一是学校里的大牛作的课题，论文发够然后开源；二是企业里的大牛搞的产品，机缘巧合于是开源；三是某些大牛突然显 灵，然后一票人跟着一起开源。每一类的例子都有不少，而不同起源的开源项目也有着自身的不同特点。具体而言，第一类项目的原理和技术上很可能颇有独到之 处，而Ceph就正在此列。相比之下，第二类项目的设计实现很可能颇为成熟，并且在开源之前或者开源初期就获得生产环境下的实际部署应用机会。这种出身背 景上的因素，对于一个开源项目的后续发展很有可能产生影响。

言归正传。Ceph项目起源于其创始人Sage Weil在加州大学Santa Cruz分校攻读博士期间的研究课题。项目的起始时间为2004年。在2006年的OSDI学术会议上，Sage发表了介绍Ceph的论文，并在该篇论文 的末尾提供了Ceph项目的下载链接。由此，Ceph开始广为人知。

Ceph使用C++语言开发。对于一个典型的强调性能的系统项目，这一选择可以理解。

作为开源项目，Ceph遵循LGPL协议。

根据Inktank官方网站上的信息，Ceph的生态系统参加下图：



Ceph ecosystem

不难看出，图中列出的厂商或组织带有明显的云计算气息。

随着Ceph的热度不断增加，Sage Weil于 2011年创立了Inktank公司以主导Ceph的开发和社区维护。目前，Ceph的发布周期为三个月。

**4. Sage Weil其人其事**

在展开后续的技术讨论之前，适度八卦Sage Weil的人生经历实在是很有必要，因为这位兄台委实是 IT男青年中凤毛麟角的在工程、研究、创业三个领域都有涉猎且都颇有建树的神人。

Sage在工程上的能力自然不必多言，而他发表Ceph论文的OSDI也是计算机操作系统领域首屈一指的最高水平学术会议。至于创业方面，Sage是 DreamHost的联合创始人，彼时是1997年，他刚上大学不久。。。有兴趣的同学可以去LinkedIn研究一下Sage的个人简历，基本上是想工作就工作，想上学就上学，想创业就创业，想读博就读博，随心所欲，天马行空，令人油然而生一种表示敬佩的冲动。

2007年于山东大学计算机科学与技术本科专业毕业，曾在华为多年从事网络设备管理系统软件的研发和运维工作；目前供职于虎牙（YY）直播云平台组，专注于Ceph 分布式存储的架构、开发、运维等工作。 欢迎您来和我做技术交流。 REPO ：https://github.com/BodihTao 微信号：cn-qiyu

分类: [存储](http://www.cnblogs.com/bodhitree/category/734690.html),[虚拟化](http://www.cnblogs.com/bodhitree/category/734685.html),[转载](http://www.cnblogs.com/bodhitree/category/737941.html)

# CEPH浅析”系列之三——CEPH的设计思想

- ThinkDiff - 博客园 http://www.cnblogs.com/bodhitree/p/4832097.html

**Ceph针对的目标应用场景**

理解Ceph的设计思想，首先还是要了解Sage设计Ceph时所针对的目标应用场景，换言之，“做这东西的目的是啥？”

事实上，Ceph最初针对的目标应用场景，就是大规模的、分布式的存储系统。所谓“大规模”和“分布式”，是指至少能够承载PB级别的数据，并且由成千上万的存储节点组成。

在大数据口号深入人心的今天，PB已经远远不是一个激动人心的系统设计目标了。但是，应该指出，Ceph项目起源于04年。那是一个商用处理器以单核为主流，常见硬盘容量只有几十GB的年代。这和现在动辄6核12线程还要双处理器、单块硬盘3TB已经司空见惯的情况是不可同日而语的。因此，理解这个设计目标，应该考虑当时的实际情况。当然，如前所述，Ceph的设计并没有理论上限，所以PB级别并不是实际应用的容量限制。

在Sage的思想中，对于这样一个大规模的存储系统，是不能以静态的眼光来看待的。对于其动态特性，笔者概括为如下三个“变化”：

**存储系统规模的变化：** 这样大规模的存储系统，往往不是在建设的第一天就能预料到其最终的规模，甚至是根本就不存在最终规模这个概念的。只能是随着业务的不断开展，业务规模的不断扩大，让系统承载越来越大的数据容量。这也就意味系统的规模自然随之变化，越来越大。

**存储系统中设备的变化：** 对于一个由成千上万个节点构成的系统，其节点的故障与替换必然是时常出现的情况。而系统一方面要足够可靠，不能使业务受到这种频繁出现的硬件及底层软件问题的影响，同时还应该尽可能智能化，降低相关维护操作的代价。

**存储系统中数据的变化：** 对于一个大规模的，通常被应用于互联网应用中的存储系统，其中存储的数据的变化也很可能是高度频繁的。新的数据不断写入，已有数据被更新、移动乃至删除。这种场景需求也是设计时必须予以考虑的。

上述三个“变化”就是Ceph目标应用场景的关键特征。Ceph所具备的各种主要特性，也都是针对这些场景特征所提出的。

**针对目标应用场景所提出的预期技术特性**

针对上述应用场景，Ceph在设计之初的几个技术特性是：

**高可靠性。** 所谓“高可靠”，首先是针对存储在系统中的数据而言，也即，尽可能保证数据不会丢失。其次，也包括数据写入过程中的可靠性，也即，在用户将数据写入Ceph存储系统的过程中，不会因为意外情况的出现造成数据丢失。

**高度自动化。** 具体包括了数据的自动replication，自动re-balancing，自动failure detection和自动failure recovery。总体而言，这些自动化特性一方面保证了系统的高度可靠，一方面也保障了在系统规模扩大之后，其运维难度仍能保持在一个相对较低的水平。

**高可扩展性。** 这里的“可扩展”概念比较广义，既包括了系统规模和存储容量的可扩展，也包括了随着系统节点数增加的聚合数据访问带宽的线性扩展，还包括了基于功能丰富强大的底层API提供多种功能、支持多种应用的功能性可扩展。

**针对预期技术特性所提出的设计思路**

针对3.2节中介绍的预期技术特性，Sage对于Ceph的设计思路基本上可以概括为以下两点：

**充分发挥存储设备自身的计算能力。** 事实上，采用具有计算能力的设备（最简单的例子就是普通的服务器）作为存储系统的存储节点，这种思路即便在当时来看也并不新鲜。但是，Sage认为这些已有系统基本上都只是将这些节点作为功能简单的存储节点。而如果充分发挥节点上的计算能力，则可以实现前面提出的预期特性。这一点成为了Ceph系统设计的核心思想。

**去除所有的中心点。** 一旦系统中出现中心点，则一方面引入单点故障点，另一方面也必然面临当系统规模扩大时的规模和性能瓶颈。除此之外，如果中心点出现在数据访问的关键路径上，事实上也必然导致数据访问的延迟增大。而这些显然都是Sage所设想的系统中不应该出现的问题。虽然在大多数系统的工程实践中，单点故障点和性能瓶颈的问题可以通过为中心点增加备份加以缓解，但Ceph系统最终采用创新的方法更为彻底地解决了这个问题。

**支撑设计思路实现的关键技术创新**

无论多么新颖奇妙的设计思路，最终落地必定需要有技术实力的支撑。而这也正是Ceph最为闪亮的地方。

Ceph最为核心的技术创新就是前面所概括的八个字——“无需查表，算算就好”。一般而言，一个大规模分布式存储系统，必须要能够解决两个最基本的问题：

**一是“我应该把数据写入到什么地方”。** 对于一个存储系统，当用户提交需要写入的数据时，系统必须迅速决策，为数据分配一个存储位置和空间。这个决策的速度影响到数据写入延迟，而更为重要的是，其决策的合理性也影响着数据分布的均匀性。这又会进一步影响存储单元寿命、数据存储可靠性、数据访问速度等后续问题。

**二是“我之前把数据写到什么地方去了”。** 对于一个存储系统，高效准确的处理数据寻址问题也是基本能力之一。

针对上述两个问题，传统的分布式存储系统常用的解决方案是引入专用的服务器节点，在其中存储用于维护数据存储空间映射关系的数据结构。在用户写入/访问数据时，首先连接这一服务器进行查找操作，待决定/查到数据实际存储位置后，再连接对应节点进行后续操作。由此可见，传统的解决方案一方面容易导致单点故障和性能瓶颈，另一方面也容易导致更长的操作延迟。

针对这一问题，Ceph彻底放弃了基于查表的数据寻址方式，而改用基于计算的方式。简言之，任何一个Ceph存储系统的客户端程序，仅仅使用不定期更新的少量本地元数据，加以简单计算，就可以根据一个数据的ID决定其存储位置。对比之后可以看出，这种方式使得传统解决方案的问题一扫而空。Ceph的几乎所有优秀特性都是基于这种数据寻址方式实现的。

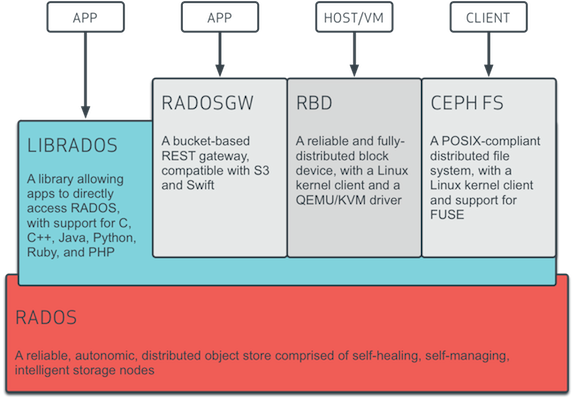
至此为止，Ceph的设计思想已经得到了较为全面深入的介绍。此后几篇文章将依次介绍Ceph的系统架构、工作原理与流程、主要特性等内容，并联系OpenStack，将Ceph和Swift加以对比分析。

2007年于山东大学计算机科学与技术本科专业毕业，曾在华为多年从事网络设备管理系统软件的研发和运维工作；目前供职于虎牙（YY）直播云平台组，专注于Ceph 分布式存储的架构、开发、运维等工作。 欢迎您来和我做技术交流。 REPO ：https://github.com/BodihTao 微信号：cn-qiyu

# Ceph浅析”系列之四——Ceph的结构

- ThinkDiff - 博客园 http://www.cnblogs.com/bodhitree/p/4832139.html

**Ceph系统的层次结构**

Ceph存储系统的逻辑层次结构如下图所示[1]。  


Ceph系统逻辑层次结构  
自下向上，可以将Ceph系统分为四个层次：

（1）基础存储系统RADOS（Reliable, Autonomic, Distributed Object Store，即可靠的、自动化的、分布式的对象存储）

顾名思义，这一层本身就是一个完整的对象存储系统，所有存储在Ceph系统中的用户数据事实上最终都是由这一层来存储的。而Ceph的高可靠、高可扩展、高性能、高自动化等等特性本质上也是由这一层所提供的。因此，理解RADOS是理解Ceph的基础与关键。

物理上，RADOS由大量的存储设备节点组层，每个节点拥有自己的硬件资源（CPU、内存、硬盘、网络），并运行着操作系统和文件系统。4.2、4.3节将对RADOS进行展开介绍。

（2）基础库librados

这一层的功能是对RADOS进行抽象和封装，并向上层提供API，以便直接基于RADOS（而不是整个Ceph）进行应用开发。特别要注意的是，RADOS是一个对象存储系统，因此，librados实现的API也只是针对对象存储功能的。

RADOS采用C++开发，所提供的原生librados API包括C和C++两种，其文档参见[2]。物理上，librados和基于其上开发的应用位于同一台机器，因而也被称为本地API。应用调用本机上的librados API，再由后者通过socket与RADOS集群中的节点通信并完成各种操作。

（3）高层应用接口

这一层包括了三个部分：RADOS GW（RADOS Gateway）、 RBD（Reliable Block Device）和Ceph FS（Ceph File System），其作用是在librados库的基础上提供抽象层次更高、更便于应用或客户端使用的上层接口。

其中，RADOS GW是一个提供与Amazon S3和Swift兼容的RESTful API的gateway，以供相应的对象存储应用开发使用。RADOS GW提供的API抽象层次更高，但功能则不如librados强大。因此，开发者应针对自己的需求选择使用。

RBD则提供了一个标准的块设备接口，常用于在虚拟化的场景下为虚拟机创建volume。目前，Red Hat已经将RBD驱动集成在KVM/QEMU中，以提高虚拟机访问性能。

Ceph FS是一个POSIX兼容的分布式文件系统。由于还处在开发状态，因而Ceph官网并不推荐将其用于生产环境中。

（4）应用层

这一层就是不同场景下对于Ceph各个应用接口的各种应用方式，例如基于librados直接开发的对象存储应用，基于RADOS GW开发的对象存储应用，基于RBD实现的云硬盘等等。

在上文的介绍中，有一个地方可能容易引起困惑：RADOS自身既然已经是一个对象存储系统，并且也可以提供librados API，为何还要再单独开发一个RADOS GW？

理解这个问题，事实上有助于理解RADOS的本质，因此有必要在此加以分析。粗看起来，librados和RADOS GW的区别在于，librados提供的是本地API，而RADOS GW提供的则是RESTful API，二者的编程模型和实际性能不同。而更进一步说，则和这两个不同抽象层次的目标应用场景差异有关。换言之，虽然RADOS和S3、Swift同属分布式对象存储系统，但RADOS提供的功能更为基础、也更为丰富。这一点可以通过对比看出。

由于Swift和S3支持的API功能近似，这里以Swift举例说明。Swift提供的API功能主要包括：

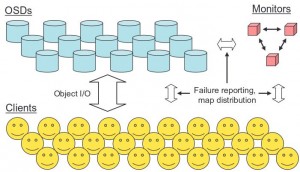
用户管理操作：用户认证、获取账户信息、列出容器列表等；  
容器管理操作：创建/删除容器、读取容器信息、列出容器内对象列表等；  
对象管理操作：对象的写入、读取、复制、更新、删除、访问许可设置、元数据读取或更新等。

由此可见，Swift（以及S3）提供的API所操作的“对象”只有三个：用户账户、用户存储数据对象的容器、数据对象。并且，所有的操作均不涉及存储系统 的底层硬件或系统信息。不难看出，这样的API设计完全是针对对象存储应用开发者和对象存储应用用户的，并且假定其开发者和用户关心的内容更偏重于账户和数据的管理，而对底层存储系统细节不感兴趣，更不关心效率、性能等方面的深入优化。

而librados API的设计思想则与此完全不同。一方面，librados中没有账户、容器这样的高层概念；另一方面，librados API向开发者开放了大量的RADOS状态信息与配置参数，允许开发者对RADOS系统以及其中存储的对象的状态进行观察，并强有力地对系统存储策略进行控制。换言之，通过调用librados API，应用不仅能够实现对数据对象的操作，还能够实现对RADOS系统的管理和配置。这对于S3和Swift的RESTful API设计是不可想像的，也是没有必要的。

基于上述分析对比，不难看出，librados事实上更适合对于系统有着深刻理解，同时对于功能定制扩展和性能深度优化有着强烈需求的高级用户。基于librados的开发可能更适合于在私有Ceph系统上开发专用应用，或者为基于Ceph的公有存储系统开发后台数据管理、处理应用。而RADOS GW则更适合于常见的基于web的对象存储应用开发，例如公有云上的对象存储服务。

**RADOS的逻辑结构**

RADOS的系统逻辑结构如下图所示[3]：  


RADOS 如图所示，RADOS集群主要由两种节点组成。一种是为数众多的、负责完成数据存储和维护功能的OSD（Object Storage Device），另一种则是若干个负责完成系统状态检测和维护的monitor。OSD和monitor之间相互传输节点状态信息，共同得出系统的总体工作状态，并形成一个全局系统状态记录数据结构，即所谓的cluster map。这个数据结构与RADOS提供的特定算法相配合，便实现了Ceph“无需查表，算算就好”的核心机制以及若干优秀特性。

在使用RADOS系统时，大量的客户端程序通过与OSD或者monitor的交互获取cluster map，然后直接在本地进行计算，得出对象的存储位置后，便直接与对应的OSD通信，完成数据的各种操作。可见，在此过程中，只要保证cluster map不频繁更新，则客户端显然可以不依赖于任何元数据服务器，不进行任何查表操作，便完成数据访问流程。在RADOS的运行过程中，cluster map的更新完全取决于系统的状态变化，而导致这一变化的常见事件只有两种：OSD出现故障，或者RADOS规模扩大。而正常应用场景下，这两种事件发生的频率显然远远低于客户端对数据进行访问的频率。

**OSD的逻辑结构**

根据定义，OSD可以被抽象为两个组成部分，即系统部分和守护进程（OSD deamon）部分。

OSD的系统部分本质上就是一台安装了操作系统和文件系统的计算机，其硬件部分至少包括一个单核的处理器、一定数量的内存、一块硬盘以及一张网卡。

由于这么小规模的x86架构服务器并不实用（事实上也见不到），因而实际应用中通常将多个OSD集中部署在一台更大规模的服务器上。在选择系统配置时，应当能够保证每个OSD占用一定的计算能力、一定量的内存和一块硬盘。同时，应当保证该服务器具备足够的网络带宽。具体的硬件配置选择可以参考[4]。

在上述系统平台上，每个OSD拥有一个自己的OSD deamon。这个deamon负责完成OSD的所有逻辑功能，包括与monitor和其他OSD（事实上是其他OSD的deamon）通信以维护更新系统状态，与其他OSD共同完成数据的存储和维护，与client通信完成各种数据对象操作等等。

Ceph系统的逻辑结构就介绍到这里。下篇文章将着重说明Ceph（主要是RADOS）的工作原理和操作流程。

2007年于山东大学计算机科学与技术本科专业毕业，曾在华为多年从事网络设备管理系统软件的研发和运维工作；目前供职于虎牙（YY）直播云平台组，专注于Ceph 分布式存储的架构、开发、运维等工作。 欢迎您来和我做技术交流。 REPO ：https://github.com/BodihTao 微信号：cn-qiyu

# “Ceph浅析”系列之五——Ceph的工作原理及流程

- ThinkDiff - 博客园 http://www.cnblogs.com/bodhitree/p/4832176.html

本文将对Ceph的工作原理和若干关键工作流程进行扼要介绍。如前所述，由于Ceph的功能实现本质上依托于RADOS，因而，此处的介绍事实上也是针对RADOS进行。对于上层的部分，特别是RADOS GW和RBD，由于现有的文档中（包括Sage的论文中）并未详细介绍，因而本文或有语焉不详之处，还请读者多多包涵。

本文将首先介绍RADOS中最为核心的、基于计算的对象寻址机制，然后说明对象存取的工作流程，之后介绍RADOS集群维护的工作过程，最后结合Ceph的结构和原理对其技术优势加以回顾和剖析。

**寻址流程**

Ceph系统中的寻址流程如下图所示[1]。  


obj\_locating

上图左侧的几个概念说明如下：

**File** —— 此处的file就是用户需要存储或者访问的文件。对于一个基于Ceph开发的对象存储应用而言，这个file也就对应于应用中的“对象”，也就是用户直接操作的“对象”。

**Ojbect** —— 此处的object是RADOS所看到的“对象”。Object与上面提到的file的区别是，object的最大size由RADOS限定（通常为2MB或4MB），以便实现底层存储的组织管理。因此，当上层应用向RADOS存入size很大的file时，需要将file切分成统一大小的一系列object（最后一个的大小可以不同）进行存储。为避免混淆，在本文中将尽量避免使用中文的“对象”这一名词，而直接使用file或object进行说明。

**PG（Placement Group）**—— 顾名思义，PG的用途是对object的存储进行组织和位置映射。具体而言，一个PG负责组织若干个object（可以为数千个甚至更多），但一个object只能被映射到一个PG中，即，PG和object之间是“一对多”映射关系。同时，一个PG会被映射到n个OSD上，而每个OSD上都会承载大量的PG，即，PG和OSD之间是“多对多”映射关系。在实践当中，n至少为2，如果用于生产环境，则至少为3。一个OSD上的PG则可达到数百个。事实上，PG数量的设置牵扯到数据分布的均匀性问题。关于这一点，下文还将有所展开。

**OSD** —— 即object storage device，前文已经详细介绍，此处不再展开。唯一需要说明的是，OSD的数量事实上也关系到系统的数据分布均匀性，因此其数量不应太少。在实践当中，至少也应该是数十上百个的量级才有助于Ceph系统的设计发挥其应有的优势。

**Failure domain** —— 这个概念在论文中并没有进行定义，好在对分布式存储系统有一定概念的读者应该能够了解其大意。

基于上述定义，便可以对寻址流程进行解释了。具体而言， Ceph中的寻址至少要经历以下三次映射：

**（1）File -> object映射**

这次映射的目的是，将用户要操作的file，映射为RADOS能够处理的object。其映射十分简单，本质上就是按照object的最大size对file进行切分，相当于RAID中的条带化过程。这种切分的好处有二：一是让大小不限的file变成最大size一致、可以被RADOS高效管理的object；二是让对单一file实施的串行处理变为对多个object实施的并行化处理。

每一个切分后产生的object将获得唯一的oid，即object id。其产生方式也是线性映射，极其简单。图中，ino是待操作file的元数据，可以简单理解为该file的唯一id。ono则是由该file切分产生的某个object的序号。而oid就是将这个序号简单连缀在该file id之后得到的。举例而言，如果一个id为filename的file被切分成了三个object，则其object序号依次为0、1和2，而最终得到的oid就依次为filename0、filename1和filename2。

这里隐含的问题是，ino的唯一性必须得到保证，否则后续映射无法正确进行。

**（2）Object -> PG映射**

在file被映射为一个或多个object之后，就需要将每个object独立地映射到一个PG中去。这个映射过程也很简单，如图中所示，其计算公式是：

hash(oid) & mask -> pgid

由此可见，其计算由两步组成。首先是使用Ceph系统指定的一个静态哈希函数计算oid的哈希值，将oid映射成为一个近似均匀分布的伪随机值。然后，将这个伪随机值和mask按位相与，得到最终的PG序号（pgid）。根据RADOS的设计，给定PG的总数为m（m应该为2的整数幂），则mask的值为m-1。因此，哈希值计算和按位与操作的整体结果事实上是从所有m个PG中近似均匀地随机选择一个。基于这一机制，当有大量object和大量PG时，RADOS能够保证object和PG之间的近似均匀映射。又因为object是由file切分而来，大部分object的size相同，因而，这一映射最终保证了，各个PG中存储的object的总数据量近似均匀。

从介绍不难看出，这里反复强调了“大量”。只有当object和PG的数量较多时，这种伪随机关系的近似均匀性才能成立，Ceph的数据存储均匀性才有保证。为保证“大量”的成立，一方面，object的最大size应该被合理配置，以使得同样数量的file能够被切分成更多的object；另一方面，Ceph也推荐PG总数应该为OSD总数的数百倍，以保证有足够数量的PG可供映射。

**（3）PG -> OSD映射**

第三次映射就是将作为object的逻辑组织单元的PG映射到数据的实际存储单元OSD。如图所示，RADOS采用一个名为CRUSH的算法，将pgid代入其中，然后得到一组共n个OSD。这n个OSD即共同负责存储和维护一个PG中的所有object。前已述及，n的数值可以根据实际应用中对于可靠性的需求而配置，在生产环境下通常为3。具体到每个OSD，则由其上运行的OSD deamon负责执行映射到本地的object在本地文件系统中的存储、访问、元数据维护等操作。

和“object -> PG”映射中采用的哈希算法不同，这个CRUSH算法的结果不是绝对不变的，而是受到其他因素的影响。其影响因素主要有二：

一是当前系统状态，也就是在《“Ceph浅析”系列之四——逻辑结构》中曾经提及的cluster map。当系统中的OSD状态、数量发生变化时，cluster map可能发生变化，而这种变化将会影响到PG与OSD之间的映射。

二是存储策略配置。这里的策略主要与安全相关。利用策略配置，系统管理员可以指定承载同一个PG的3个OSD分别位于数据中心的不同服务器乃至机架上，从而进一步改善存储的可靠性。

因此，只有在系统状态（cluster map）和存储策略都不发生变化的时候，PG和OSD之间的映射关系才是固定不变的。在实际使用当中，策略一经配置通常不会改变。而系统状态的改变或者是由于设备损坏，或者是因为存储集群规模扩大。好在Ceph本身提供了对于这种变化的自动化支持，因而，即便PG与OSD之间的映射关系发生了变化，也并不会对应用造成困扰。事实上，Ceph正是需要有目的的利用这种动态映射关系。正是利用了CRUSH的动态特性，Ceph可以将一个PG根据需要动态迁移到不同的OSD组合上，从而自动化地实现高可靠性、数据分布re-blancing等特性。

之所以在此次映射中使用CRUSH算法，而不是其他哈希算法，原因之一正是CRUSH具有上述可配置特性，可以根据管理员的配置参数决定OSD的物理位置映射策略；另一方面是因为CRUSH具有特殊的“稳定性”，也即，当系统中加入新的OSD，导致系统规模增大时，大部分PG与OSD之间的映射关系不会发生改变，只有少部分PG的映射关系会发生变化并引发数据迁移。这种可配置性和稳定性都不是普通哈希算法所能提供的。因此，CRUSH算法的设计也是Ceph的核心内容之一，具体介绍可以参考[2]。

至此为止，Ceph通过三次映射，完成了从file到object、PG和OSD整个映射过程。通观整个过程，可以看到，这里没有任何的全局性查表操作需求。至于唯一的全局性数据结构cluster map，在后文中将加以介绍。可以在这里指明的是，cluster map的维护和操作都是轻量级的，不会对系统的可扩展性、性能等因素造成不良影响。

一个可能出现的困惑是：为什么需要同时设计第二次和第三次映射？难道不重复么？关于这一点，Sage在其论文中解说不多，而笔者个人的分析如下：

我们可以反过来想像一下，如果没有PG这一层映射，又会怎么样呢？在这种情况下，一定需要采用某种算法，将object直接映射到一组OSD上。如果这种算法是某种固定映射的哈希算法，则意味着一个object将被固定映射在一组OSD上，当其中一个或多个OSD损坏时，object无法被自动迁移至其他OSD上（因为映射函数不允许），当系统为了扩容新增了OSD时，object也无法被re-balance到新的OSD上（同样因为映射函数不允许）。这些限制都违背了Ceph系统高可靠性、高自动化的设计初衷。

如果采用一个动态算法（例如仍然采用CRUSH算法）来完成这一映射，似乎是可以避免静态映射导致的问题。但是，其结果将是各个OSD所处理的本地元数据量爆增，由此带来的计算复杂度和维护工作量也是难以承受的。

例如，在Ceph的现有机制中，一个OSD平时需要和与其共同承载同一个PG的其他OSD交换信息，以确定各自是否工作正常，是否需要进行维护操作。由于一个OSD上大约承载数百个PG，每个PG内通常有3个OSD，因此，一段时间内，一个OSD大约需要进行数百至数千次OSD信息交换。

然而，如果没有PG的存在，则一个OSD需要和与其共同承载同一个object的其他OSD交换信息。由于每个OSD上承载的object很可能高达数百万个，因此，同样长度的一段时间内，一个OSD大约需要进行的OSD间信息交换将暴涨至数百万乃至数千万次。而这种状态维护成本显然过高。

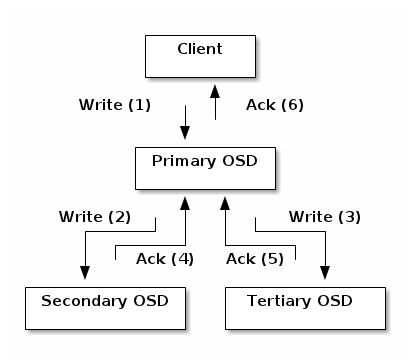
综上所述，笔者认为，引入PG的好处至少有二：一方面实现了object和OSD之间的动态映射，从而为Ceph的可靠性、自动化等特性的实现留下了空间；另一方面也有效简化了数据的存储组织，大大降低了系统的维护管理开销。理解这一点，对于彻底理解Ceph的对象寻址机制，是十分重要的。

**数据操作流程**

此处将首先以file写入过程为例，对数据操作流程进行说明。

为简化说明，便于理解，此处进行若干假定。首先，假定待写入的file较小，无需切分，仅被映射为一个object。其次，假定系统中一个PG被映射到3个OSD上。

基于上述假定，则file写入流程可以被下图表示[3]：



如图所示，当某个client需要向Ceph集群写入一个file时，首先需要在本地完成5.1节中所叙述的寻址流程，将file变为一个object，然后找出存储该object的一组三个OSD。这三个OSD具有各自不同的序号，序号最靠前的那个OSD就是这一组中的Primary OSD，而后两个则依次是Secondary OSD和Tertiary OSD。

找出三个OSD后，client将直接和Primary OSD通信，发起写入操作（步骤1）。Primary OSD收到请求后，分别向Secondary OSD和Tertiary OSD发起写入操作（步骤2、3）。当Secondary OSD和Tertiary OSD各自完成写入操作后，将分别向Primary OSD发送确认信息（步骤4、5）。当Primary OSD确信其他两个OSD的写入完成后，则自己也完成数据写入，并向client确认object写入操作完成（步骤6）。

之所以采用这样的写入流程，本质上是为了保证写入过程中的可靠性，尽可能避免造成数据丢失。同时，由于client只需要向Primary OSD发送数据，因此，在Internet使用场景下的外网带宽和整体访问延迟又得到了一定程度的优化。

当然，这种可靠性机制必然导致较长的延迟，特别是，如果等到所有的OSD都将数据写入磁盘后再向client发送确认信号，则整体延迟可能难以忍受。因此，Ceph可以分两次向client进行确认。当各个OSD都将数据写入内存缓冲区后，就先向client发送一次确认，此时client即可以向下执行。待各个OSD都将数据写入磁盘后，会向client发送一个最终确认信号，此时client可以根据需要删除本地数据。

分析上述流程可以看出，在正常情况下，client可以独立完成OSD寻址操作，而不必依赖于其他系统模块。因此，大量的client可以同时和大量的OSD进行并行操作。同时，如果一个file被切分成多个object，这多个object也可被并行发送至多个OSD。

从OSD的角度来看，由于同一个OSD在不同的PG中的角色不同，因此，其工作压力也可以被尽可能均匀地分担，从而避免单个OSD变成性能瓶颈。

如果需要读取数据，client只需完成同样的寻址过程，并直接和Primary OSD联系。目前的Ceph设计中，被读取的数据仅由Primary OSD提供。但目前也有分散读取压力以提高性能的讨论。

**集群维护**

前面的介绍中已经提到，由若干个monitor共同负责整个Ceph集群中所有OSD状态的发现与记录，并且共同形成cluster map的master版本，然后扩散至全体OSD以及client。OSD使用cluster map进行数据的维护，而client使用cluster map进行数据的寻址。

在集群中，各个monitor的功能总体上是一样的，其相互间的关系可以被简单理解为主从备份关系。因此，在下面的讨论中不对各个monitor加以区分。

略显出乎意料的是，monitor并不主动轮询各个OSD的当前状态。正相反，OSD需要向monitor上报状态信息。常见的上报有两种情况：一是新的OSD被加入集群，二是某个OSD发现自身或者其他OSD发生异常。在收到这些上报信息后，monitor将更新cluster map信息并加以扩散。其细节将在下文中加以介绍。

Cluster map的实际内容包括：

（1） Epoch，即版本号。Cluster map的epoch是一个单调递增序列。Epoch越大，则cluster map版本越新。因此，持有不同版本cluster map的OSD或client可以简单地通过比较epoch决定应该遵从谁手中的版本。而monitor手中必定有epoch最大、版本最新的cluster map。当任意两方在通信时发现彼此epoch值不同时，将默认先将cluster map同步至高版本一方的状态，再进行后续操作。

（2）各个OSD的网络地址。

（3）各个OSD的状态。OSD状态的描述分为两个维度：up或者down（表明OSD是否正常工作），in或者out（表明OSD是否在至少一个PG中）。因此，对于任意一个OSD，共有四种可能的状态：

—— Up且in：说明该OSD正常运行，且已经承载至少一个PG的数据。这是一个OSD的标准工作状态；

—— Up且out：说明该OSD正常运行，但并未承载任何PG，其中也没有数据。一个新的OSD刚刚被加入Ceph集群后，便会处于这一状态。而一个出现故障的OSD被修复后，重新加入Ceph集群时，也是处于这一状态；

—— Down且in：说明该OSD发生异常，但仍然承载着至少一个PG，其中仍然存储着数据。这种状态下的OSD刚刚被发现存在异常，可能仍能恢复正常，也可能会彻底无法工作；

—— Down且out：说明该OSD已经彻底发生故障，且已经不再承载任何PG。

（4）CRUSH算法配置参数。表明了Ceph集群的物理层级关系（cluster hierarchy），位置映射规则（placement rules）。

根据cluster map的定义可以看出，其版本变化通常只会由（3）和（4）两项信息的变化触发。而这两者相比，（3）发生变化的概率更高一些。这可以通过下面对OSD工作状态变化过程的介绍加以反映。

一个新的OSD上线后，首先根据配置信息与monitor通信。Monitor将其加入cluster map，并设置为up且out状态，再将最新版本的cluster map发给这个新OSD。

收到monitor发来的cluster map之后，这个新OSD计算出自己所承载的PG（为简化讨论，此处我们假定这个新的OSD开始只承载一个PG），以及和自己承载同一个PG的其他OSD。然后，新OSD将与这些OSD取得联系。如果这个PG目前处于降级状态（即承载该PG的OSD个数少于正常值，如正常应该是3个，此时只有2个或1个。这种情况通常是OSD故障所致），则其他OSD将把这个PG内的所有对象和元数据复制给新OSD。数据复制完成后，新OSD被置为up且in状态。而cluster map内容也将据此更新。这事实上是一个自动化的failure recovery过程。当然，即便没有新的OSD加入，降级的PG也将计算出其他OSD实现failure recovery。

如果该PG目前一切正常，则这个新OSD将替换掉现有OSD中的一个（PG内将重新选出Primary OSD），并承担其数据。在数据复制完成后，新OSD被置为up且in状态，而被替换的OSD将退出该PG（但状态通常仍然为up且in，因为还要承载其他PG）。而cluster map内容也将据此更新。这事实上是一个自动化的数据re-balancing过程。

如果一个OSD发现和自己共同承载一个PG的另一个OSD无法联通，则会将这一情况上报monitor。此外，如果一个OSD deamon发现自身工作状态异常，也将把异常情况主动上报给monitor。在上述情况下，monitor将把出现问题的OSD的状态设为down且in。如果超过某一预订时间期限，该OSD仍然无法恢复正常，则其状态将被设置为down且out。反之，如果该OSD能够恢复正常，则其状态会恢复为up且in。在上述这些状态变化发生之后，monitor都将更新cluster map并进行扩散。这事实上是自动化的failure detection过程。

由之前介绍可以看出，对于一个Ceph集群而言，即便由数千个甚至更多OSD组成，cluster map的数据结构大小也并不惊人。同时，cluster map的状态更新并不会频繁发生。即便如此，Ceph依然对cluster map信息的扩散机制进行了优化，以便减轻相关计算和通信压力。

首先，cluster map信息是以增量形式扩散的。如果任意一次通信的双方发现其epoch不一致，则版本更新的一方将把二者所拥有的cluster map的差异发送给另外一方。

其次，cluster map信息是以异步且lazy的形式扩散的。也即，monitor并不会在每一次cluster map版本更新后都将新版本广播至全体OSD，而是在有OSD向自己上报信息时，将更新回复给对方。类似的，各个OSD也是在和其他OSD通信时，将更新发送给版本低于自己的对方。

基于上述机制，Ceph避免了由于cluster map版本更新而引起的广播风暴。这虽然是一种异步且lazy的机制，但根据Sage论文中的结论，对于一个由n个OSD组成的Ceph集群，任何一次版本更新能够在O(log(n))时间复杂度内扩散到集群中的任何一个OSD上。

一个可能被问到的问题是：既然这是一种异步和lazy的扩散机制，则在版本扩散过程中，系统必定出现各个OSD看到的cluster map不一致的情况，这是否会导致问题？答案是：不会。事实上，如果一个client和它要访问的PG内部的各个OSD看到的cluster map状态一致，则访问操作就可以正确进行。而如果这个client或者PG中的某个OSD和其他几方的cluster map不一致，则根据Ceph的机制设计，这几方将首先同步cluster map至最新状态，并进行必要的数据re-balancing操作，然后即可继续正常访问。

通过上述介绍，我们可以简要了解Ceph究竟是如果基于cluster map机制，并由monitor、OSD和client共同配合完成集群状态的维护与数据访问的。特别的，基于这个机制，事实上可以自然而然的完成自动化的数据备份、数据re-balancing、故障探测和故障恢复，并不需要复杂的特殊设计。这一点确实让人印象深刻。

至此为止，本系列文章已经较为系统地介绍了Ceph的设计思想、逻辑架构、工作原理和主要操作流程。最为技术的部分已经结束。之后应该还有两篇，分别会介绍Ceph与OpenStack的故事，以及笔者个人对于Ceph的一些思考。

2007年于山东大学计算机科学与技术本科专业毕业，曾在华为多年从事网络设备管理系统软件的研发和运维工作；目前供职于虎牙（YY）直播云平台组，专注于Ceph 分布式存储的架构、开发、运维等工作。 欢迎您来和我做技术交流。 REPO ：https://github.com/BodihTao 微信号：cn-qiyu

# “CEPH浅析”系列之六——CEPH与OPENSTACK

- ThinkDiff - 博客园 http://www.cnblogs.com/bodhitree/p/4832233.html

在 《“Ceph浅析”系列之二——Ceph概况》中即已提到，关注Ceph的原因之一，就是OpenStack社区对于Ceph的重视。因此，本文将对Ceph在OpenStack中的价值进行简要介绍，并且对Ceph和Swift进行对比。

**Ceph在OpenStack中的地位**

对于一个IaaS系统，涉及到存储的部分主要是块存储服务模块、对象存储服务模块、镜像管理模块和计算服务模块。具体针对OpenStack而言，则分别对应为其中的Cinder、Swift、Glance和Nova四个项目[1]。

在块存储服务部分，Ceph目前是Cinder项目的默认存储后端。前已述及，Red Hat也已经利用自己在KVM/QEMU社区中的影响力，将RBD驱动直接集成在QEMU中。这样，虚拟机访问基于RBD实现的块设备的性能将得到优化。

在对象存储部分，Swift是OpenStack自带的对象存储实现方案。但Ceph也已经成为了Swift最强有力的竞争对手。目前Swift也在考虑采用Ceph作为自己的存储后端。关于Ceph和Swift的故事将在6.2节详细展开。

在镜像管理部分，目前Glance已经支持将Ceph作为自己的本地镜像文件缓存。

在计算服务部分，United Stack目前正在推动将Ceph FS作为Nova计算节点的本地文件系统。

整体而言，Ceph事实上是目前OpenStack生态系统中呼声最高的开源存储解决方案。这一点从笔者在OpenStack 2013 HongKong Summit上的亲身体验可以得到印证。目前，以HP、Dell、Intel等为代表的企业IT领导厂商，和以Mirantis、eNovance、United Stack为代表的若干OpenStack社区新兴厂商，都将Ceph作为重要的乃至于首选的开源存储解决方案。

笔者认为，Ceph之所以在诞生多年不温不火的情况下，迅速在OpenStack社区中受到关注，除了其他一些明显优点之外，应该还是和其支持统一存储的能力有关。这一特性恰恰是OpenStack社区所需要的。

OpenStack项目设计的准则之一就是灵活可扩展。同时，其各个成员项目的背景也不尽相同。这也就导致各个项目在涉及存储系统时所采取的选择各有差异。但是，这一现状势必导致OpenStack的部署和运维面临一定的挑战。特别是对于一些规模不大的OpenStack部署实例，如果让块存储、对象存储、镜像缓存、计算节点本地存储等模块分别采用三四种不同的后端解决方案，则一方面其部署十分麻烦，另一方面运维人员的后续工作也很繁琐。在这种情况下，如果能够采用Ceph作为一种统一存储后端，则确实可以有效缓解这一问题。当然，这只是笔者的一家直言。任何技术选择必然都有其复杂的背后原因，这里的信息仅供参考。

**Ceph与Swift：不能不说的故事，不能不作的比较**

首先对Swift项目的来龙去脉进行简单介绍，以便大家更好地了解这个项目的特性，及其背后隐藏的原因。此处关于Swift的信息主要引自[2]。

Swift最早起源于2008年，本来是Rackspace公司内部开发的用于支撑其公有云对象存储业务的后端系统。当时，Amazon的S3服务已经颇受欢迎，因此，Rackspace决定开发Swift以提供对应业务作为回应。也正是因为这个原因，Swift的设计目标十分纯粹，就是一个优秀的、可以和S3相媲美的对象存储系统。其他要求纯属多余，因此完全不在Swift开发者的考虑之列。

Swift的开发大致历时一年，并在Rackspace成功上线运营。此后，OpenStack项目于2010年正式发布。Rackspace贡献了Swift，而NASA贡献了Nova，二者成为了OpenStack最早的两个项目。其后，若干Swift开发团队的核心成员独立创业，成立了SwiftStack公司，依然活跃在相关社区。

由此可见，Swift正是一个典型的起源于公司内部的、作为正式产品开发的开源项目。从这一点而言，Swift和“学院范儿”的Ceph可谓截然不同。也正是因为这个原因，Swift获得了一个得天独厚的优势：不缺启动用户，一开始就有生产环境下的大规模部署应用案例。事实上，相对成熟、web场景下应用案例多，是Swift社区目前依然反复强调的一个优势。

从技术上讲，Swift的特点主要体现在设计目标明确，就是要做一个纯粹的对象存储系统，因此不会考虑Ceph所强调的统一存储特性。同时，为了便于和其他项目、应用集成，Swift选择了Python语言进行开发。

与之相比，Ceph同时考虑了对象存储、块存储和文件系统存储能力，且目前在OpenStack中应用最多的场景事实上是块存储。同时，Ceph在选择开发语言时，很可能主要考虑的是性能因素，因而选择了C++语言。而能够被用于块存储场景这一点，也部分印证了其性能确实比较优秀。

由此可见，Ceph和Swift的区别，本质上是由其产生背景和应用目标所导致的。对这二者进行对比，并进行技术上的评判，并不非常公平。

事实上，作为开源分布式存储系统中的两个优秀代表，Ceph和Swift的设计和特性之中，也有着不少的相通之处：

首先，二者都强调良好的可扩展性，因此都采用了无中心点结构。只不过Swift的架构中有元数据服务器，只是通过多节点扩展的方式尽可能解决了其可靠性和性能顾虑。

第二，二者都能提供可配置的高可靠性。在两者的集群中，数据的备份数都可以选择，在常见生产环境中，也都使用三备份的方式。

第三，二者都强调自动化的集群管理。Swift同样引入了自动化的集群维护能力。

由此可见，简单地强调这两者之中的某一个更为优秀，是不合理的，也是没有实际意义的。

当然，在实际使用中，毕竟还是需要进行方案选择。结合[3]文中的观点，笔者认为，合适的选择或许如下：

\*如果需要一个纯粹的对象存储系统，则选择Swift；

\*如果需要一个纯粹的块存储系统，则只能选择Ceph；

\*如果是一个小规模的、希望控制系统复杂度的OpenStack部署方案，则选择Ceph；

\*如果是一个规模较大的系统，块存储和对象存储分别有较大的业务需求，则可以考虑将二者分离，分别采用Ceph和Swift。

到本篇为止，这一系列文章对于Ceph技术内容的介绍已经基本完成。后面一篇文章，将主要是笔者学习Ceph过程中的一些思考，供有兴趣的读者一同品评。

2007年于山东大学计算机科学与技术本科专业毕业，曾在华为多年从事网络设备管理系统软件的研发和运维工作；目前供职于虎牙（YY）直播云平台组，专注于Ceph 分布式存储的架构、开发、运维等工作。 欢迎您来和我做技术交流。 REPO ：https://github.com/BodihTao 微信号：cn-qiyu

# “Ceph浅析”系列之七——关于Ceph的若干想法

- ThinkDiff - 博客园 http://www.cnblogs.com/bodhitree/p/4832251.html

本篇文章的内容，主要是笔者在调研分析Ceph过程中产生的一些思考。因为其中的内容比较自由发散，且大多是笔者的个人见解，故此另启一文进行讨论。

**关于Ceph的性能**

目前为止，本系列的文章中没有涉及到Ceph性能的详细讨论，也没有给出任何的Ceph性能数据。原因很简单：笔者本人没有机会进行详尽的Ceph性能分析研究，也没有见到比较全面的相关数据。因此，为了避免以片面的数据误导读者，便没有提供任何信息。

以笔者个人的经验而言，探讨一个系统领域的开源项目的性能，事实上并不容易。其原因在于，影响一个实际部署中系统的性能好坏的因素太多、太复杂。硬件配置、 软件版本、参数调整、应用负载及场景设置，各个方面的因素变化都会导致性能测试结果的不同。因此，很难一言蔽之，认为某个项目的性能是好还是不好。

举一个不直接相关的例子。在hypervisor领域，大家很可能会倾向于认为ESXi的性能优于KVM，但事实上，在SPECvirt性能测试结果排行榜 上，基于KVM的系统常有高居第一的时候。究其原因，除了硬件性能的因素之外，KVM有大量的配置参数可以调整，而调得好与不好会极其明显地影响系统性 能。

又比如常用的开源大数据工具软件Hadoop。同一个Hadoop集群用同样的应用程序处理同样的数据集，在配置参数不同的情况下，其最终运行时间长度可能相差数倍。

正是因为参数配置、硬件规格、软件版本、应用场景等因素都可能对性能产生明显影响，因此，对于Ceph这样一个部署方案多变、配置参数不少的系统，如何评测其系统性能，是需要审慎思考的。

反过来讲，这倒也是开源软件引出的一个生财之道。虽然软件本身是开源的，大家都可以免费下载免费安装，但能不能用好就要依靠精深的专业技能了。类似的公司国外屡见不鲜，而国内也已经开始出现。

**Ceph的架构与硬件平台之间的适应性**

Ceph自2006年正式发布以来，其基础架构（RADOS）部分并没有发生大的变化。本质上，这还是因为RADOS的设计确实优秀，有其前瞻性，因此没有必要大动筋骨。但这并不意味着没有必要对其进行适当反思。

如 前所述，2006年的时候，商用处理器的主流仍为单核，单条内存和单块硬盘的容量也都远小于现在的主流水平。但是，OSD的基本硬件资源要求并没有发生变 化。这也就意味着，在目前的典型部署方案中，一台物理服务器上很可能有数十个处理器硬件线程、数十块硬盘，于是也就承载着数十个OSD同时运行。然 而，RADOS结构的基本假定是，集群是由大量的、相互独立运行的OSD组成的，则目前的典型硬件方案有可能影响这种假定的有效性。例如，如果一台服务器 出现故障，必须关机进行维修，则意味着数十个OSD一起突然下线。由此受到影响的PG则可能多达成千上万个。这种突发性的事件对系统的自动化维护机制可能 会造成一定的压力。

由 此，笔者想到，Sage设计Ceph时面对的硬件平台，事实上应该是处理能力不需要过强、硬件规格比较简单的系统。而这种系统可能与目前的ARM架构或者 Intel Atom架构的micro-server更为相似。或许，基于micro-server部署Ceph集群，会是一种值得尝试的方向。

此外，华为和希捷合作推出了IP硬盘产品。虽然还缺乏更进一步的了解，但直观上推测，这种全新的、轻量级、智能化的存储设备，可能也是一种非常近似于Sage当年设想中的OSD的硬件平台。

**Ceph与软件定义存储**

“软件定义”这四个字可谓是目前最炙手可热、也最让人糊涂的概念之一。软件定义计算、软件定义网络、软件定义存储、软件定义数据中心，以上几个可能是目前最为常见的相关名词了。

到底什么是“软件定义”，现在还没有形成完全一致的见解。并且，参考技术发展史上的若干先例，以后也未必能形成所谓的一致见解。在这种情况下，以一个具体实例入手，可能更容易获得直观认识，并由此建立起更系统的观点。

笔者认为，对于任何一个系统而言，“软件定义”的概念，更多体现在这里：这个系统的哪些特性，比如功能或者性能，以前是固定的，或者只能进行有限的配置，而现在则可以进行方便灵活地定义和改变。

例如，对于一台物理服务器，一旦其硬件配置，如CPU、内存、硬盘等连接好，则这台服务器的规格和性能就确定了，能够通过BIOS配置等方式调整的性能和功 能范围是很有限的。但是，对于一台虚拟机而言，即便在虚拟机已经创建并安装了操作系统之后，其CPU核数及处理能力、逻辑物理内存大小及真实物理内存大 小、硬盘数量容量及读写性能、网卡型号数量及网络带宽等等特性都是可以方便灵活地通过软件方式进行控制和改变的（其中部分配置操作需要对虚拟机进行重启才 能生效），且这种配置可以由应用层软件进行控制。两相对比，则虚拟机的这种可定义性就是软件定义计算的一个直观实例。

下面再具体到存储领域加以讨论。一般而言，一个存储系统的主要特性大致包括：存储类型（文件系统？块存储？对象存储？），存储容量，存储性能（访问带宽、访 问延迟等等），存储策略（备份策略、访问安全性策略、对数据的高级处理功能等等）。参考上面所举出的软件定义计算的例子，可以想见，对于一个软件定义存储 系统而言，这些特性（至少是其中的大多数）都应该是可以通过软件方式加以定义的。

具体到Ceph而言，其最为符合软件定义存储的特性无疑是，Ceph的存储类型是可以通过软件方式定义的。同样的一个RADOS集群，可以通过安装不同的上 层软件和对应的客户端程序，实现块存储、对象存储和文件系统存储功能，这一特性对于传统的存储系统难以想象。除此之外，Ceph的存储策略，如备份策略、 后台数据处理功能等，也都可以方便地通过软件方式加以定义或扩展。因此，从这个角度出发，Ceph也可以被认为是软件定义存储的真实案例之一。

**Ceph与数据中心计算**

传统意义上，计算系统的设计是以计算为中心的。数据从存储、网络或其他设备流入处理器，经过处理后再流向存储、网络或其他设备。然而，随着待处理的数据量以 爆炸式的速度增大，也随着计算能力提高的速度超过存储和传输能力，这一处理方式可能变得不再经济，因为针对大量的数据进行频繁硬盘存取和网络传输的代价都 是非常可观的。

数据中心计算这一概念，也就是在这种背景下被提出的。其核心思想，也就是让计算在数据所在的地方发生。数据在哪里，就把计算任务发送到哪里去执行，而不要再 为了使用“强大”的计算能力把数据搬来搬去，传来传去。事实上，Hadoop的出现，就是这种数据中心计算思想的现实反映。

数据中心计算的另一实例，是目前OpenStack社区中出现的一种叫做ZeroVM的轻量级虚拟化技术[1]。ZeroVM的思想就是让计算发生在数据所在的地方。基于其官方提供的信息，目前已经实现了ZeroVM和Swift的整合，可以让处理任务直接运行在Swift的服务器端。

事实上，Ceph也提供了同样的能力。Ceph的整个设计，都是基于Sage的一个基本思想：充分发挥存储器件自身的计算能力。这种思想不仅使得OSD可以 相互配合完成数据访问操作和集群维护功能，更允许OSD将富余的计算能力提供出来，用于运行数据处理任务。

目前，RADOS提供的机制允许在OSD上直接运行可动态加载的数据处理程序插件，以便在服务器端进行数据处理工作，例如，对图片存储系统中的图片进行自动 加水印、尺寸和格式自动转换等后台操作。事实上，基于这种能力，也完全可以实现类似于Hadoop的大数据处理系统。

对于大数据而言，存储和处理是其两个关键的技术领域。由于Ceph自身就是优秀的存储系统，又具备直接承载计算任务的能力，因此，面向大数据的数据中心计算很可能是Ceph的潜在应用方向之一。

**Ceph在实际应用中可能存在的问题**

到目前位置，本系列文章基本上都是在介绍Ceph的各种优势与特长。但是，任何系统都不可能是十全十美的，本着鸡蛋里挑骨头、吹毛求疵的精神，还是要在这里吐槽几句。

从非技术角度出发，Ceph的最大问题是火起来的时间不够长，因此可以参考的文档还不是很多，中文的尤其如此。但这个没有办法，只能众人拾柴火焰高，一点一滴作贡献。

此外，对Ceph诟病最多的可能还是不够成熟云云。但一个开源项目总是用得人多了才会成熟的，而Ceph目前正在这个过程中，所以需要的还是时间和参与。

另外，以笔者的感觉，Ceph的高度自动化可能也是个双刃剑。好处固然是很多的，但弊端就是系统的运行状态不完全在管理员控制之下，系统中会有若干自动触发而不是管理员触发的操作。这个特点可能会给系统状态的监测和控制带来一些复杂度，需要管理员去适应。

**基于Ceph的产业需求和可能的商业机会**

特此声明：这一节的内容纯属crazy idea，不构成投资建议:-)

首先，Ceph的安装部署和性能优化必然成为突出的需求。因此，将Ceph和商用服务器整合成易于部署、性能出色的各类存储解决方案，应该是可以考虑的方向之一。

同时，由于Ceph自身对于OSD硬件平台的特殊假设，以及由此导致的优化空间，则在成本合理的前提下，开发更加适用于Ceph OSD的定制硬件平台（类似于micro-server或者IP硬盘等），并突出存储的高密度、低功耗、高可维护性等特点，也可能成为一种选择。

此外，针对Ceph集群的专用集群监控、性能分析等工具软件也可能会有一定的需求。

最后，基于Ceph的后台数据处理软件工具包也值得考虑。

2007年于山东大学计算机科学与技术本科专业毕业，曾在华为多年从事网络设备管理系统软件的研发和运维工作；目前供职于虎牙（YY）直播云平台组，专注于Ceph 分布式存储的架构、开发、运维等工作。 欢迎您来和我做技术交流。 REPO ：https://github.com/BodihTao 微信号：cn-qiyu

# “CEPH浅析”系列之八——小结

- ThinkDiff - 博客园 http://www.cnblogs.com/bodhitree/p/4832260.html

最初决定写这些文章的时候，本打算大致记录一下，几千字也就了事了。可是越写越觉得东西多，不说明白总有些不甘心，于是就越写越长，到这儿为止貌似已经有一万七千多字了。除了博士论文之外，应该是没有写过更长的东西了。 但是，即便如此，仍然有很多内容没能涉及，只能将来有机会再加以补充了。

之所以花这么多时间在这些文章上，归根结底还是因为Ceph是个有意思的东西，多看一看，多想一想，总能有些新收获，很有趣。即便Ceph最终不能大红大紫，凭着其精彩的设计思想和巧妙的技术实现，应该还是会在存储技术领域留下一笔的。如果Ceph能够借着OpenStack的东风，逐渐走向成熟，并受到更为广泛的接受和应用，则更是研究、工程、应用相互贯通的一个经典案例，值得认真研究。因此，无论从哪个角度出发，关注Ceph都是值得的。

就先写到这吧。

2007年于山东大学计算机科学与技术本科专业毕业，曾在华为多年从事网络设备管理系统软件的研发和运维工作；目前供职于虎牙（YY）直播云平台组，专注于Ceph 分布式存储的架构、开发、运维等工作。 欢迎您来和我做技术交流。 REPO ：https://github.com/BodihTao 微信号：cn-qiyu