# 开源分布式存储架构概览

Mike Wang (WMike@outlook.com)

2017.04.09

# 提纲

- 基本思想和关注的问题
- 四种开源分布式存储系统概览
  - HDFS
  - GlusterFS
  - OpenStack Swift
  - Ceph
- 以上四种分布式存储系统功能对比

# 基本思想

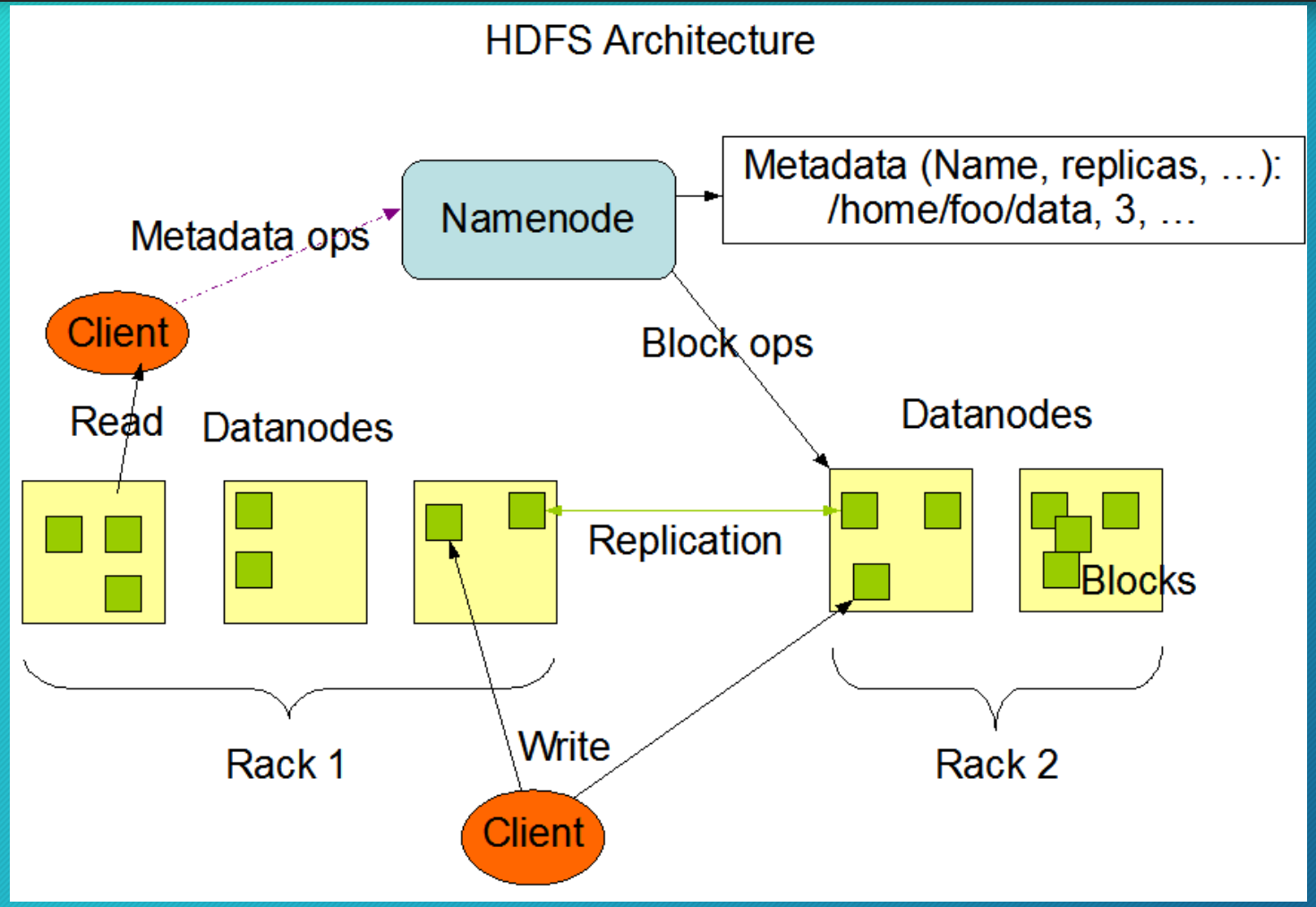
- 大文件被切割成多个小数据块，分布在多台存储节点上
  - 存储和服务的负载均衡
- 使用多副本或者EC编码提高数据持久性
  - 数据副本的存放位置会考量失败域：硬盘、主机、机架、同一交换机负责区域、同一供电系统负责区域、机房、数据中心等。
  - 定期检查数据完整性，提前发现错误，并自动修复错误
- 元数据管理方式
  - 集中管理方式（性能瓶颈，难于扩展，单点失效等。例如GFS, HDFS）
  - 分布式管理方式
    - 目录树静态分区（容易导致负载不均衡。例如Coda）
    - 目录哈希动态分区（难于实现。 例如CephFS）
  - 无服务器管理，客户端根据路径哈希值定位存储服务器（目录浏览问题，客户端负载高。例如GlusterFS）

# 关注的问题

- 分散元数据管理，避免出现单点失效或性能瓶颈
- 在控制成本的基础上，提高数据持久性(多副本或EC编码)和可用性(权衡CAP因素)
- 当集群节点数量发生变化时，尽可能减少数据重新分布的数量
- 大规模部署时，易于管理和全面监控非常重要
- 对高速硬件的支持：SSD, NVMe, RDMA
- 同一集群提供多种存储服务(如Ceph)
- 性能，还是性能


hadoop
HDFS

# 总体结构



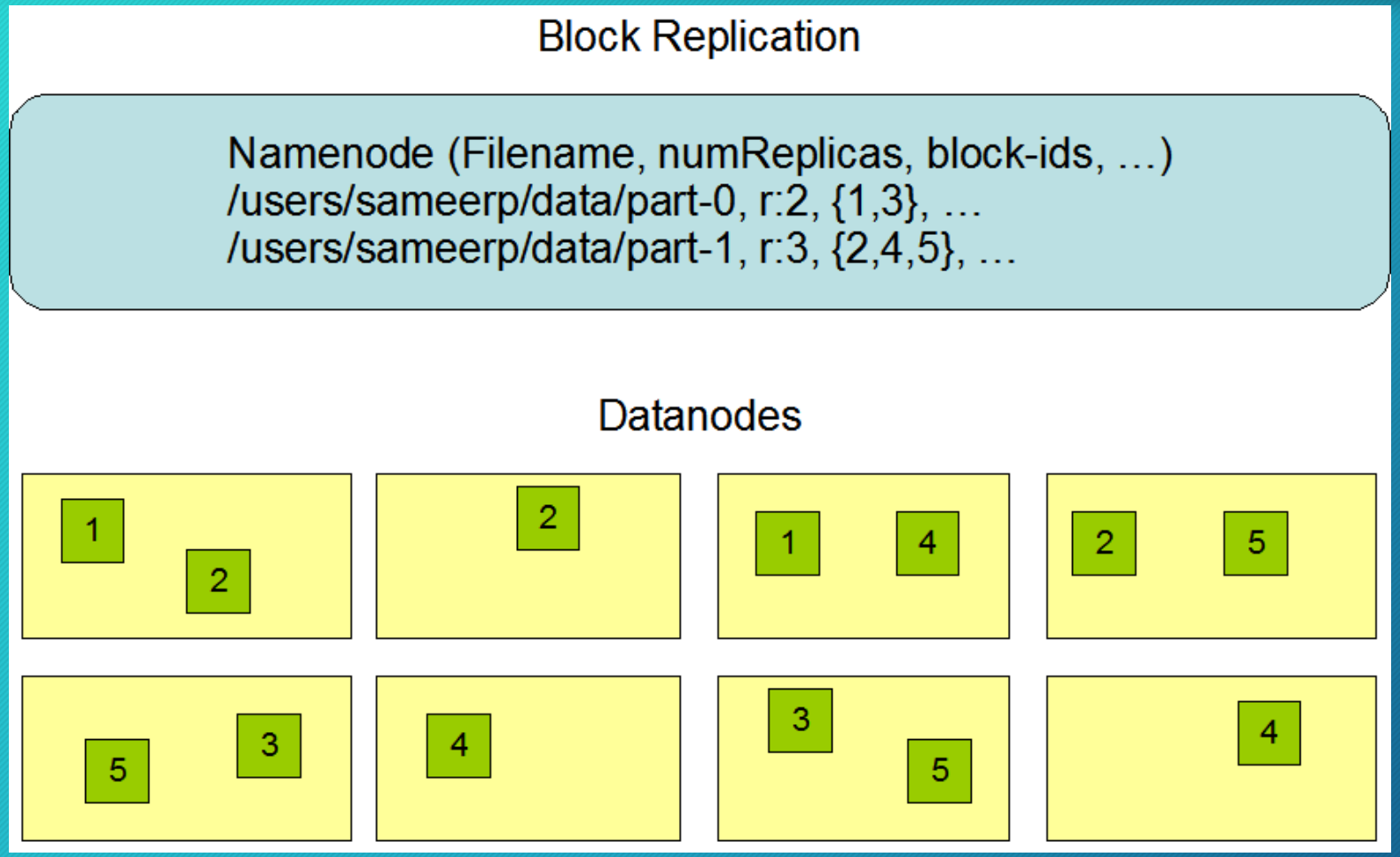

**Name Node**
文件系统元数据（文件目录结构、文件数据对应关系、文件属性），文件访问（open、close、rename等），集群维护

**Data Node**
保存文件数据副本
运行应用程序（如MapReduce）

**Client**
与Name Node交互获得文件数据的位置，与Data Node交互得到文件数据

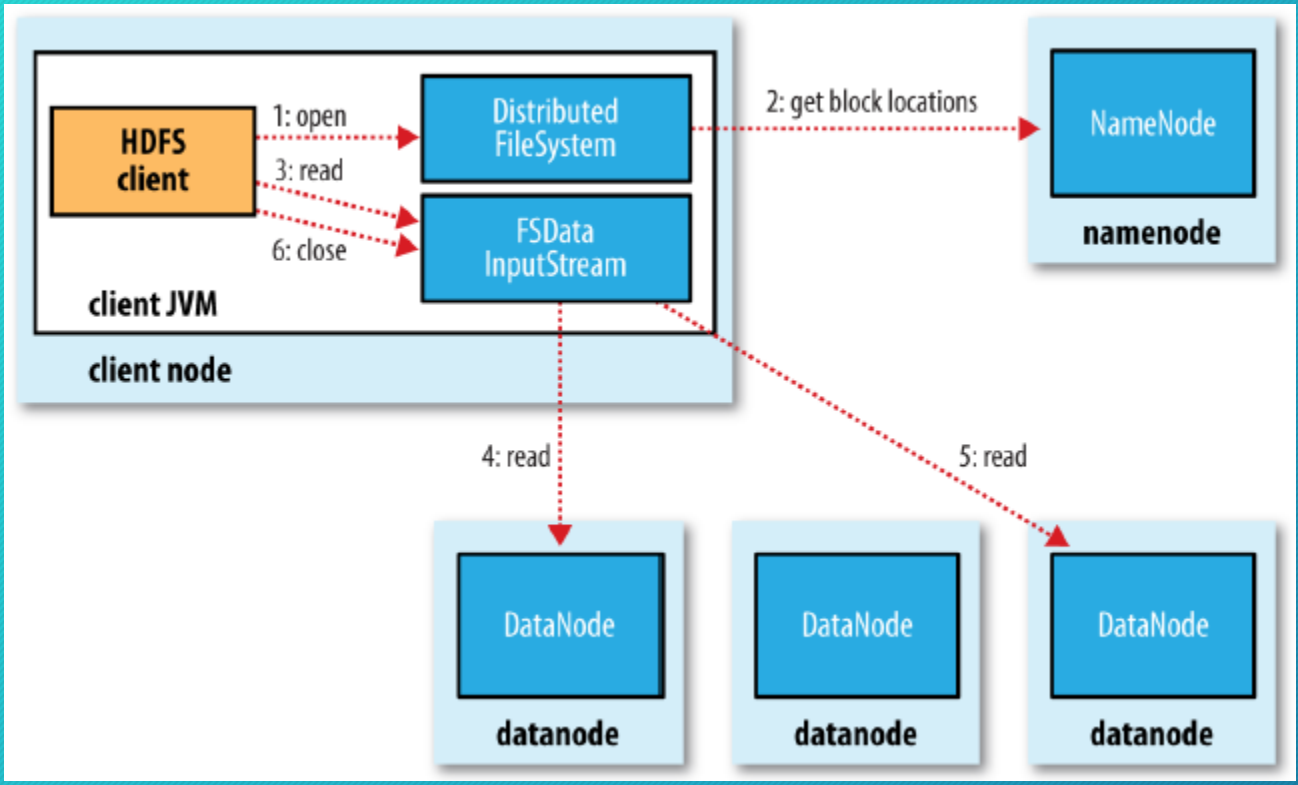
一个HDFS集群由一个Name Node和数量众多的 Data Node组成。

# 数据副本分布机制



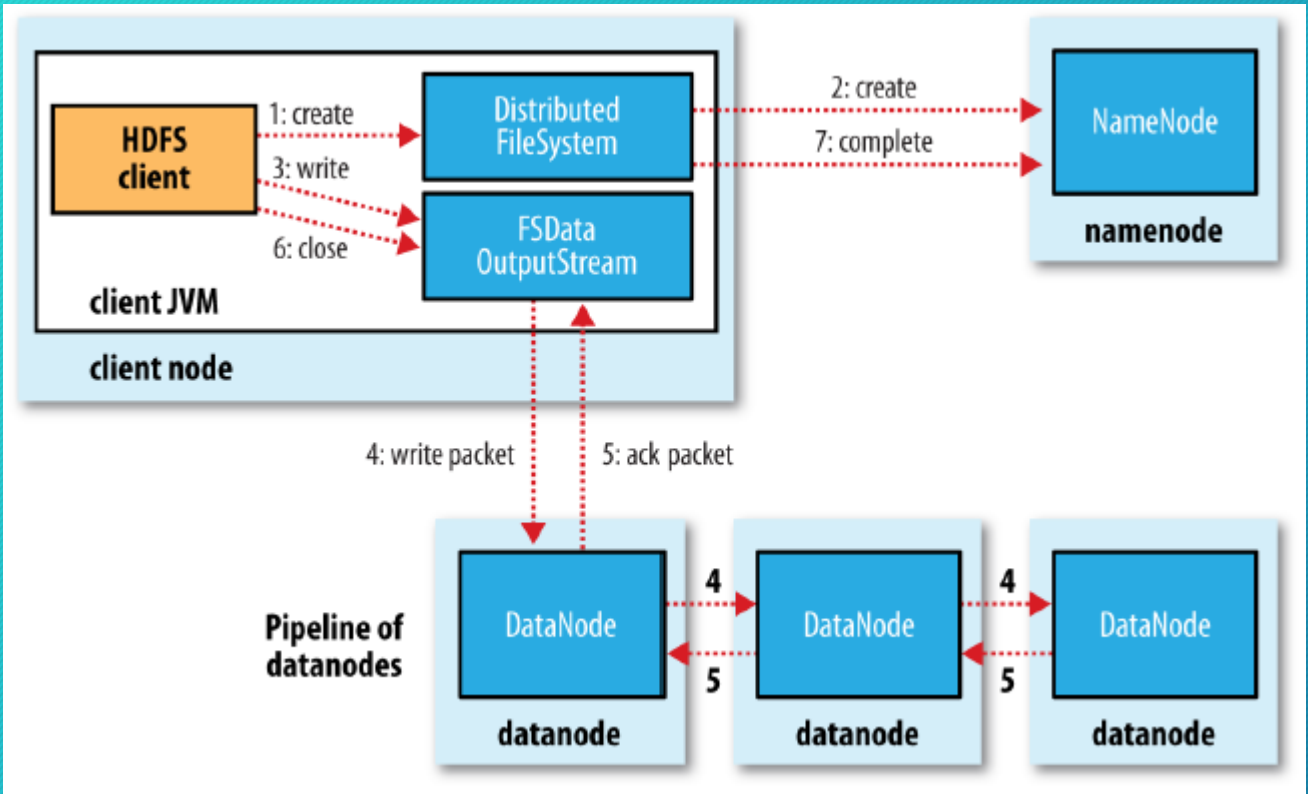

- 文件被按照固定长度切块
  - 缺省块大小为128MB
- 数据块的副本保存在不同的Data Node上
  - 缺省保存3个副本
  - 跨机架存储：一个副本在本机架Data Node上，两个副本在另一个机架的两个Data Node上（减少跨switch网络访问）
- 以上两个参数的配置粒度可达到文件级别
- 客户端选择离自己最近（根据机架位置）的Data Node读取数据

# 客户端读文件流程




1. 上层应用程序调用客户端库函数open方法尝试打开指定文件；
2. 客户端联系Name Node获得文件前几个block的位置（Data Node地址及Block ID）。
3. 应用程序调用read方法开始读取数据。客户端函数库会根据网络拓扑（机架位置）选择离自己最近的Data Node读取数据。
4. 从选定的Data Node读取第一块数据。完成后关闭连接。
5. 重新选择最适合读取第二块数据的Data Node，然后读取数据。
6. 应用程序调用close方法关闭文件

# 客户端写文件流程




Pipeline设计可以让每个Data Node节点的网卡的输入和输出得到充分利用。客户端在写文件前需要从NN获得租约，在写的过程中定期更新租约。文件关闭时租约撤销。

1. 上层应用程序调用create方法创建文件；
2. 客户端函数库联系Name Node创建指定文件（此时文件未与任何数据块关联）；
3. 应用程序调用write方法开始向内部缓存写数据。当缓存数据量达到特定值时，向Name Node请求分配一个block。Name Node会根据副本定额选择若干Data Node作为一个pipeline；
4. 客户端将数据块发送给pipeline中的第一个Data Node，该Data Node收到数据后本地存储，并转发给下一个Data Node；
5. Pipeline里的每个Data Node会向前一个节点确认收到数据。当第一个节点收到确认后，向客户端确认数据保存完成。
6. 应用程序调用close方法关闭文件。客户端函数库会确保缓存数据都被发送，并收到确认。
7. 客户端函数库通知Name Node完成文件写入。

# 写文件过程中的异常处理

- 客户端异常
  - 客户端在写文件前需要在NN获得租约。
  - 客户端需要定时更新租约。文件关闭时，租约自动释放。
  - NN定时检查租约更新情况，以判断客户端状态。
    - 对于未写完的数据块，HDFS将选择其中长度最短的，并将此长度和新的Generation Stamp(GS)更新到所有副本的DN上。
- DN异常
  - 客户端可以检测到DN异常，并将DN从流水线中移除
  - 客户端可能申请新的DN(与设置相关)，并与剩余的DN节点组成新的流水线
  - 数据块的Generation Stamp(GS)加1，客户端使用新的GS向新的流水线继续发送数据
    - GS可以用于判断数据块的多个副本中哪一个是有效的
    - NN的复制监控功能会检测到数据块的副本数目不足的情况，并引导DN增加副本数

# Name Node的数据持久化

- Name Node在内存中保存：
  - (1)目录树结构以及相关属性，以及文件与数据块的映射关系
  - (2)数据块与Data Node的映射关系
- Name Node使用文件保存：
  - Fsimage保存某时刻内存中元数据的状态(不包括#2)
  - Edits log保存fsimage之后对元数据的修改操作(记录edits log之后才向客户端返回成功)
- Name Node启动时需：
  - 加载最新fsimage以及之后的全部edits log
  - Data Node向Name Node汇报自身存储的数据块列表，形成数据块与Data Node的对应关系
- 为防止因edits log过多导致Name Node启动时间过长，Standby Name Node(或Secondary Name Node)需要定期合并edits log和fsimage，形成新的fsimage

# Name Node存储目录

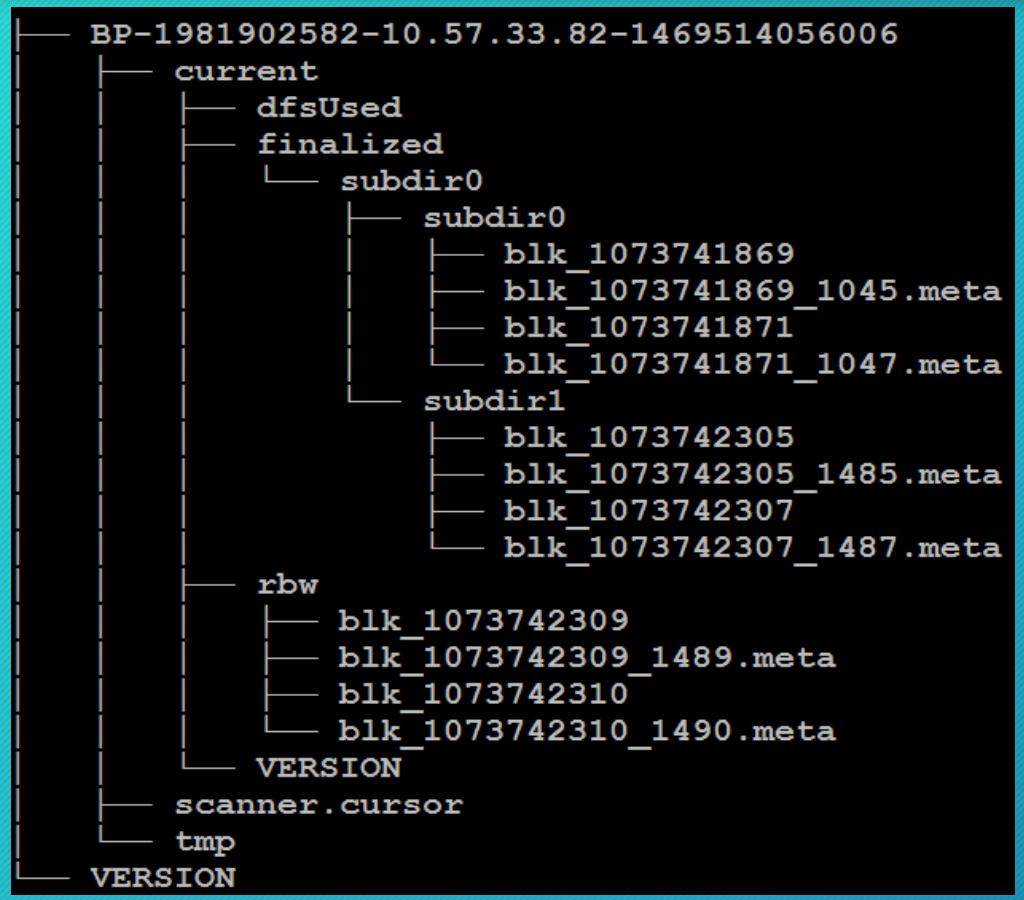
```
├── current
│   ├── edits_0000000000000004159-0000000000000004177
│   ├── edits_0000000000000004178-0000000000000004221
│   ├── edits_0000000000000004222-0000000000000004222
│   ├── edits_inprogress_0000000000000004223
│   ├── fsimage_0000000000000003867
│   ├── fsimage_0000000000000003867.md5
│   ├── fsimage_0000000000000004177
│   ├── fsimage_0000000000000004177.md5
│   ├── seen_txid
│   └── VERSION
├── in_use.lock
└── namenode-formatted
```

VERSION文件内容如下：

```
[root@wanzh02-cos7-uefi2 current]# cat VERSION
#Tue Jul 26 02:20:56 EDT 2016
namespaceID=1845997632
clusterID=CID-92186be3-7040-4edd-87bf-bc2e840b4c6a
cTime=0
storageType=NAME_NODE
blockpoolID=BP-1981902582-10.57.33.82-1469514056006
layoutVersion=-63
```

- In_use.lock文件保存当前Name Node进程的进程ID。用于防止多个进程尝试打开同一组数据文件
- Seen_txid文件保存最后一个checkpoint或edit log roll的最后一个transaction ID。Name Node启动时通过此文件检查是否有edit丢失。
- fsimage_<end transaction ID>文件包含完整的metadata数据。*.md5文件用于数据完整性检查
- edits_inprogress__<start transaction ID>文件是当前正在使用的edit日志
- edits_<start transaction ID>-<end transaction ID>文件保存已经完成的日志数据

# Data Node存储目录



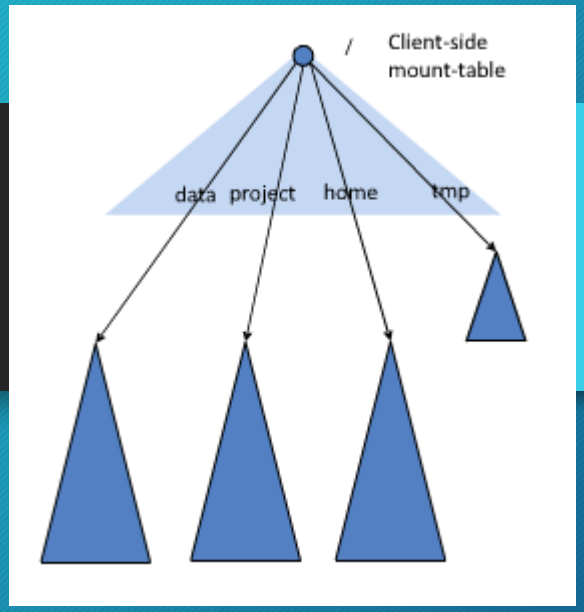

- BP-<random integer>-<Name Node-IP address>-<creation timestamp>目录下保存单个namespace下的所有文件数据
- Finalized目录下保存已经完成的数据文件
- Rbw是”replica being written”的缩写。该目录保存正在被写的数据文件。
- dfsUsed文件包含HDFS占用的字节数以及上次更新文件的时间戳。Data Node服务在启动时读取此文件，在退出时更新此文件。
- Blk_<block_id>_<Generation_Stamp>.meta包含数据块的checksum信息
  - GS可以用于检测一个数据块的多个副本中，谁是最新的

# 添加和移除Data Node

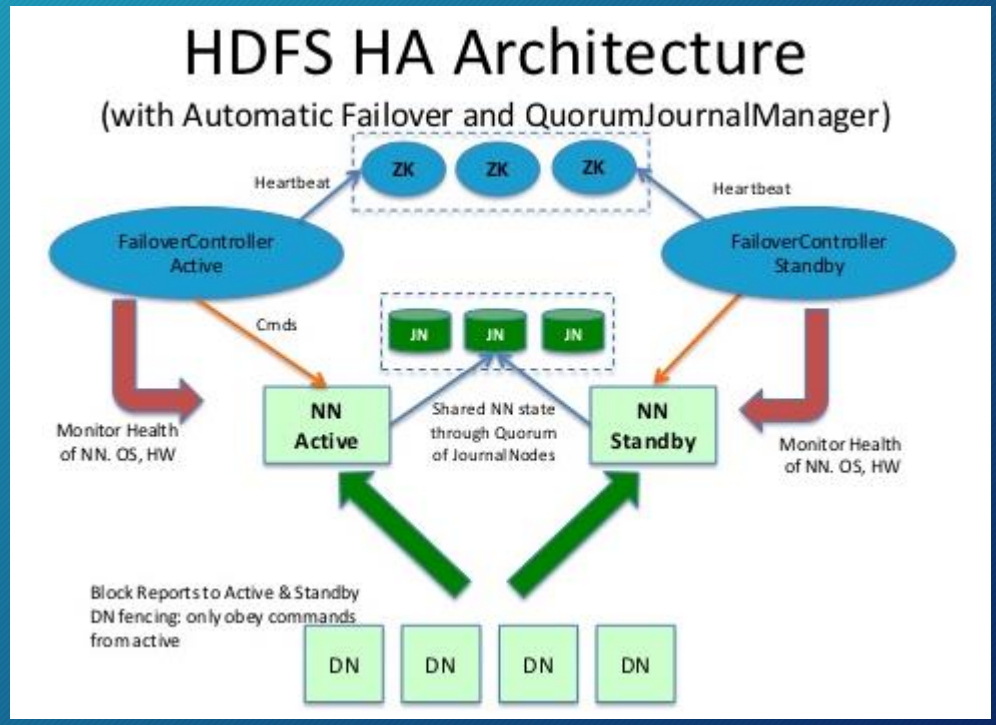
- HDFS集群使用namespace ID(整数)来标识一个文件系统。只有namespace ID相匹配(或无namespace ID)的data node才可以加入集群。
- Name Node使用Storage ID(UUDI)来唯一标识一个Data Node。即使Data Node的IP地址发生变化，也可以被正确识别。
- 添加Data Node
  - 新写入的文件会自动利用新加入的Data Node
  - 需要手工运行工具，从原有Data Node上移动部分数据到新加入的Data Node上，以达到各Data Node间利用率的平衡
- 移除Data Node
  - 通知Name Node指定Data Node将被移除
  - Name Node启动数据复制过程，确保在Data Node被移除的情况下，数据副本数仍然满足要求
  - 最终将指定Data Node从集群中移除

# 减轻集中元数据管理的副作用

- HDFS Federation
  - 集群中有多个Name Node，独立管理属于自己的命名空间
  - 集群里的Data Node接受所有的Name Node的管理，但数据块存储上是分开的
  - 每个Name Node命名空间就像普通文件系统中的一个卷
- 为Name Node提供active+standby模式的HA功能
  - ZK用于失效检测，以及确保只能有一个NN处于Active状态
  - JN用于保存edit log，必须为奇数个
    - NN只有在将修改操作保存在N/2+1个JN后，才向客户端返回成功
  - DN同时向两个NN汇报数据块保存列表




HDFS Federation



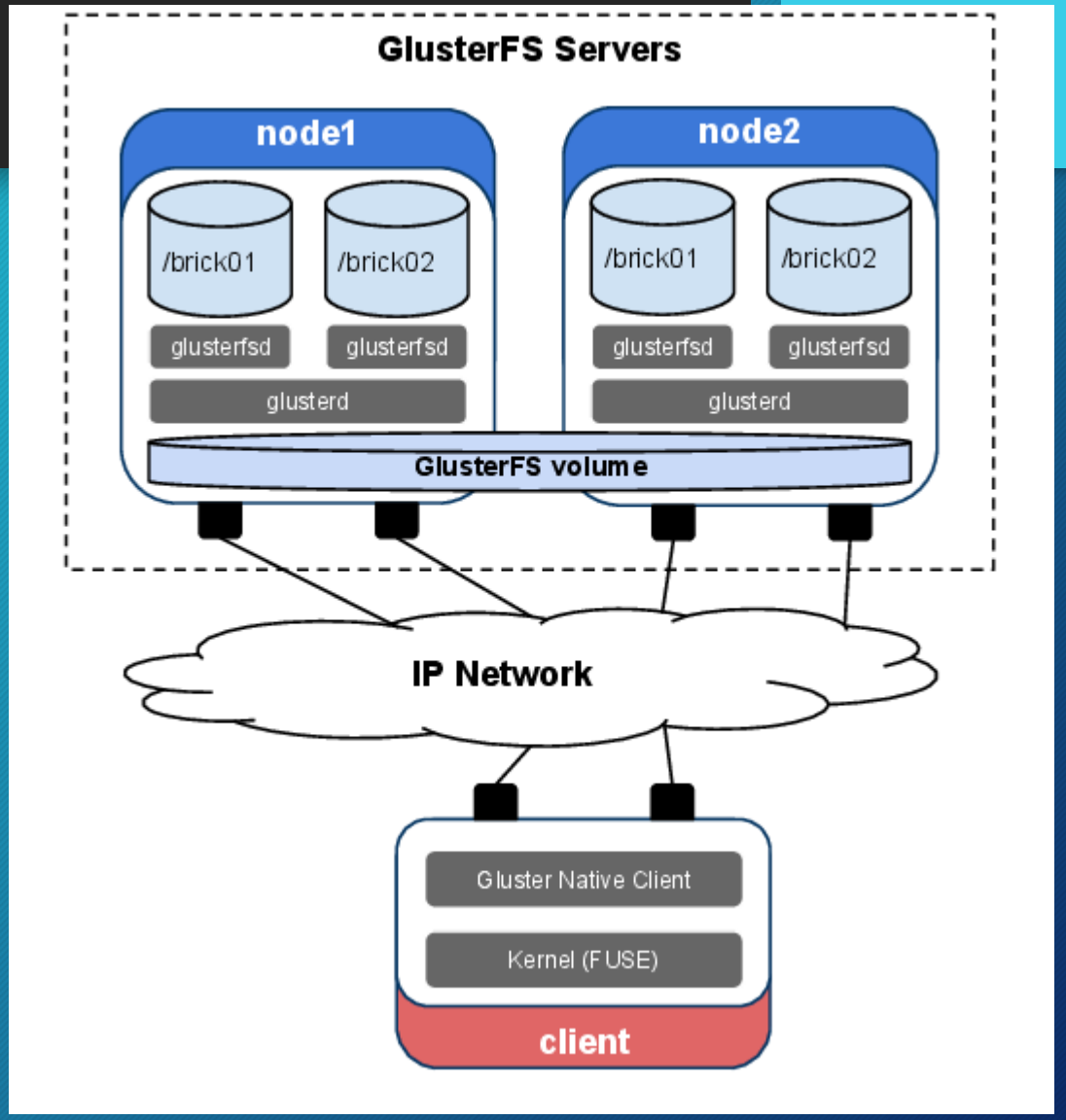
# 特点分析

- 优势
  - 可以提供data locality(即数据读写优先选择当前机器的硬盘)，适合那些“计算”和“存储”二合一的集群应用场景
  - 提供与普通文件系统相似的操作体验（特别是提供目录结构）
  - 与Hadoop生态系统联系紧密
- 劣势
  - NameNode作为元数据存储中心: (1)容易成为性能瓶颈; (2)难于容量扩展;(3)较大单点失效风险;
  - 只支持新写和附加写，不支持随机写操作
  - 适合大文件存储，不适合小文件存储
  - 存储相关的功能实现相对较慢（如“纠删码”功能刚刚出现在最新的3.0版本中）
  - 使用Java语言实现，难于和其他语言集成，难于利用硬件优化特性(例如：DPDK/RDMA)
  - 较少用于非Hadoop生态系统的环境


GlusterFS

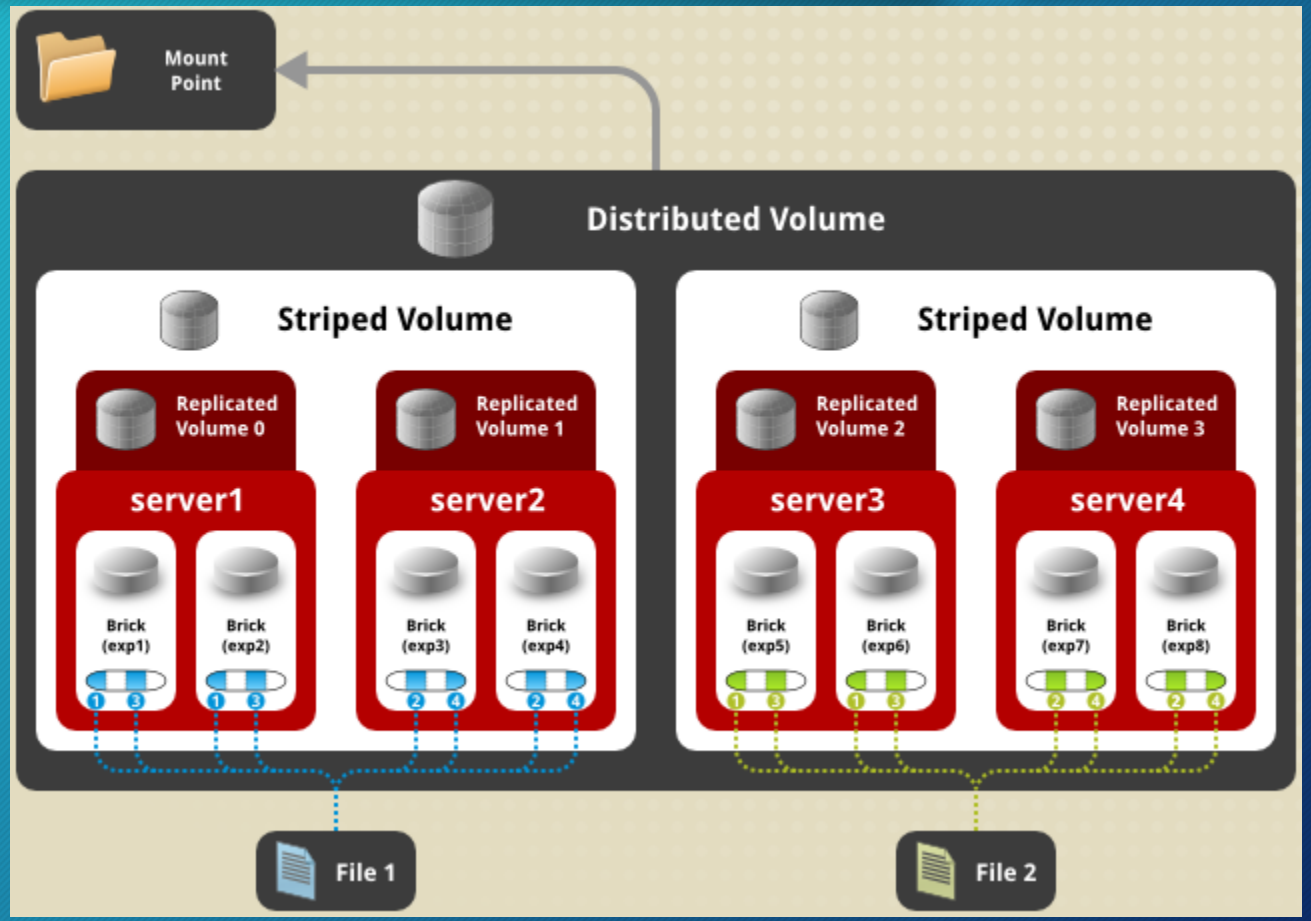
# 总体结构

- 服务器端
  - 全对称结构，无中心节点
  - Brick: 用于存储文件数据的目录，类似于NFS的export
  - Glusterfsd: 文件服务，每个为卷提供存储的brick会对应一个该服务进程
  - Glusterd: 管理服务，如管理集群包含的存储结点，Brick等，同时为客户端的mount操作提供支持（如提供卷的配置信息）
  - Volume: 逻辑概念，一个卷可以跨越多个brick或多个物理存储结点
- 客户端
  - 直接与一个卷所包含的全部存储结点通过TCP/IP网络或Infiniband RDMA通信
  - Native Client: 通过客户端库函数访问文件服务
  - FUSE: 通过mount点，向普通文件系统那样访问文件服务



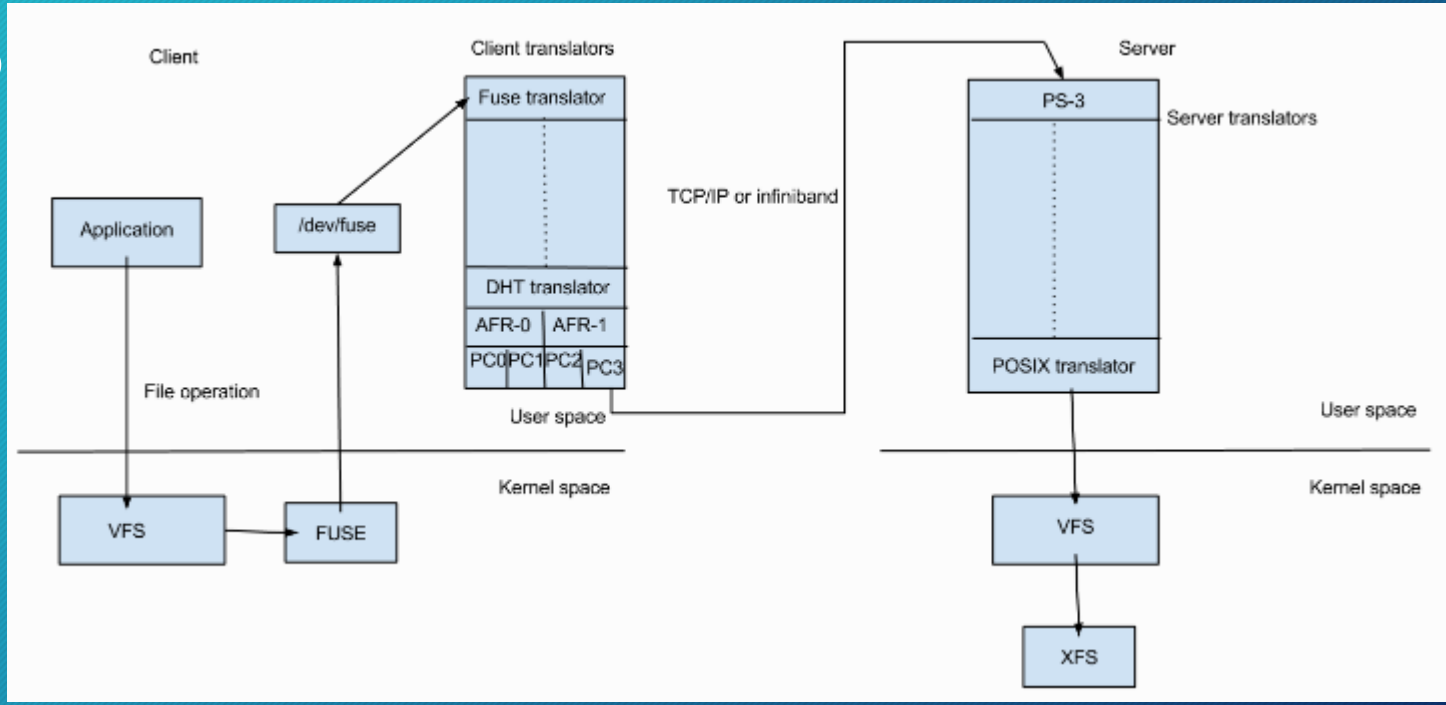
# 卷类型

- 支持多种卷类型
  - Distributed: 文件被随机保存在不同存储节点上(缺省)
  - Replicated: 每块数据在多个存储节点上保存多个副本。用以提高数据持久性。
  - Striped: 文件被切块，并保存在不同的存储节点上。用以支持大文件及改善负载均衡。
  - Distributed + Replicated
  - Distributed + Striped



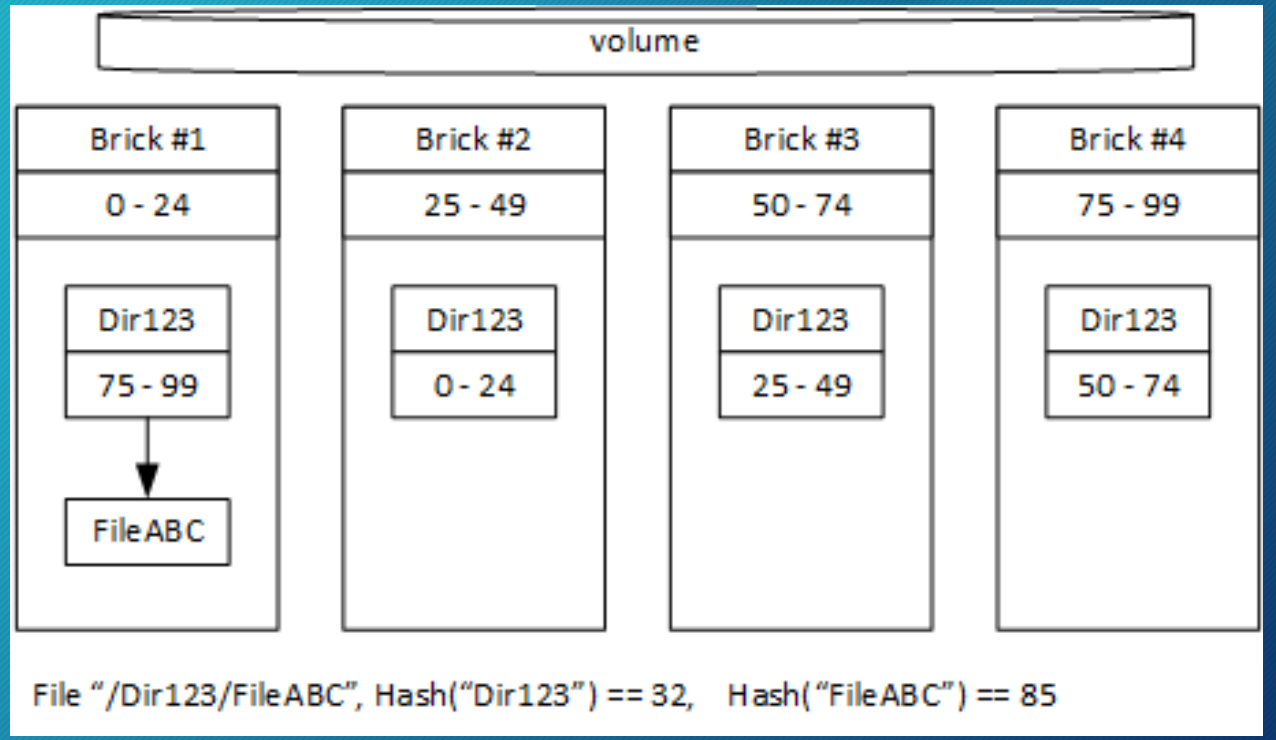
# C/S交互

- DHT Translator
  - 找到数据应该存储的位置（Brick）
- AFR Translator
  - 将数据副本复制给多个存储节点
- Protocol Client Translator
  - 最底层模块，执行具体数据发送任务
- 客户端直接与各存储结点做数据交互



# 数据分布方法（DHT，但非传统定义）

- 数据位置查询完全由客户端完成，不存在单独的元数据节点
- 目录会被创建在所有的子卷上，而文件只会保存在其中的一个子卷上（多副本模式除外）
  - 目录的扩展属性中保存有哈希区间信息，该信息决定哪些文件将被分配到该目录下
- 通过“目录GFID+文件名”的一致性哈希，与父目录的哈希区间信息比较，将文件分配到特定的子卷上
  - 目前尚未使用“虚拟节点”方法减少存储节点数量发生变化时的数据迁移




0x0 - 0x7ffffffe:

```
# file: /data/brick1/gv0/test/
trusted.glusterfs.dht=0x0000000100000000000000007ffffffe
```

0x7fffffff - 0xffffffff:

```
# file: /data/brick2/gv0/test/
trusted.glusterfs.dht=0x0000000100000007fffffffffffffff
```

# 目录和文件创建流程

根据目录哈希值确定目标子卷(brick)

向目标子卷发送mkdir命令

向其他子卷发送mkdir命令

以目标子卷为起点，为该目录在各存储结点上分配哈希区间，并存储于目录的扩展属性中

创建目录

计算文件名哈希值，根据父目录哈希区间选择目标子卷subvol #1

若subvol #1空间不足，根据空间使用率重新选择目标子卷，并在subvol #1上创建指向新选定目标子卷的链接

在最终选定的目标子卷上创建文件

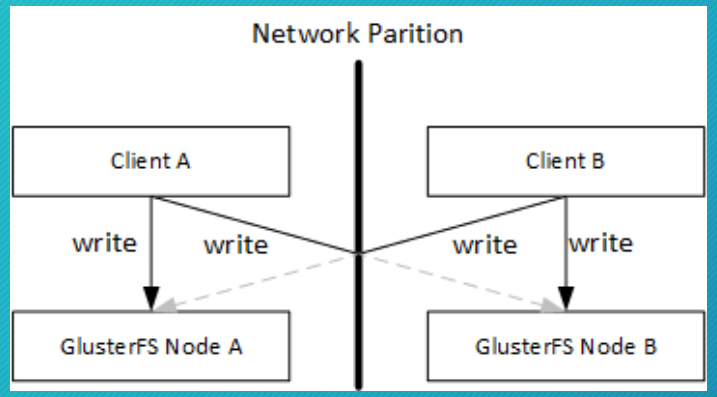
创建文件

# 脑裂(split brain)问题

多副本可以提高数据持久性。当用其来提高服务可用性时，可能带来数据一致性问题。

**例子：**

假设在机器A和B上配置一个2副本的卷。两个客户端程序分别运行在A和B上。当A和B之间网络连通时，每个客户端都会同时更新A和B上的副本。但当A和B之间连接中断时，则每个客户端只会更新自己当前机器上的副本。等网络再次连通后，就会出现数据不一致的问题：A和B都认为自己的数据是正确的。




**检测机制：**

文件和目录的扩展属性中保存有除自身以外的全部副本的更新状态(Changelog)。更新前加1，完成后减1。正常时，它们的值为0。如果操作完成后的ChangeLog更新没有完成，则trusted.afr.dirty被置位。

```
# file: /data/brickA/gv1/test/file
trusted.afr.dirty=0x000000000000000100000000
trusted.afr.gv1-client-1=0x000000020000000200000000
```

A和B双副本，停掉节点B后更新文件，在节点A上查看到的情况。

**避免方法：**

(1)服务器端多数派仲裁(Server quorum)：

集群中的一个节点，当不能联系到设定数量的其他节点时，就停止其上的所有brick。

(2)客户端多数派仲裁(Client quorum)：

详见GlusterFS帮助文档。

# 添加和移除存储节点

- 数据迁移量如右图
- GlusterFS对添加节点做了优化。优化后的数据迁移量如下。

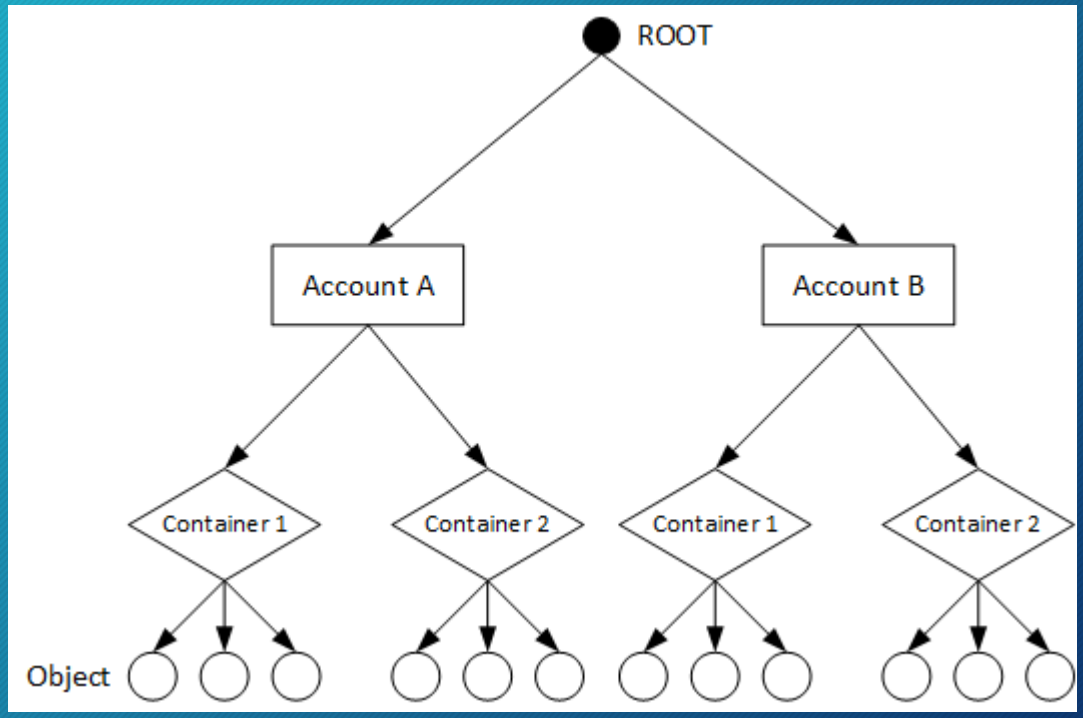
# 特点分析

- 优势
  - 提供Posix兼容的文件系统访问接口
  - 适合大文件存储
  - 全对称结构，无中心节点
  - 提供跨区域复制功能
  - 文件被直接保存在存储节点上（striped类型除外），易于手工数据恢复
  - 软件设计采用堆栈式架构，非常适合插件式扩展
- 弱势
  - 当存储节点数量发生变动时，为达到数据平衡而移动的数据量较多
  - 数据平衡前，对不存在的文件的查找性能会显著下降（需要遍历所有存储节点以查找文件）
  - 目录的创建和读取(ls)性能较差，需要访问全部存储节点
  - 缺少集中监控模块


# Swift

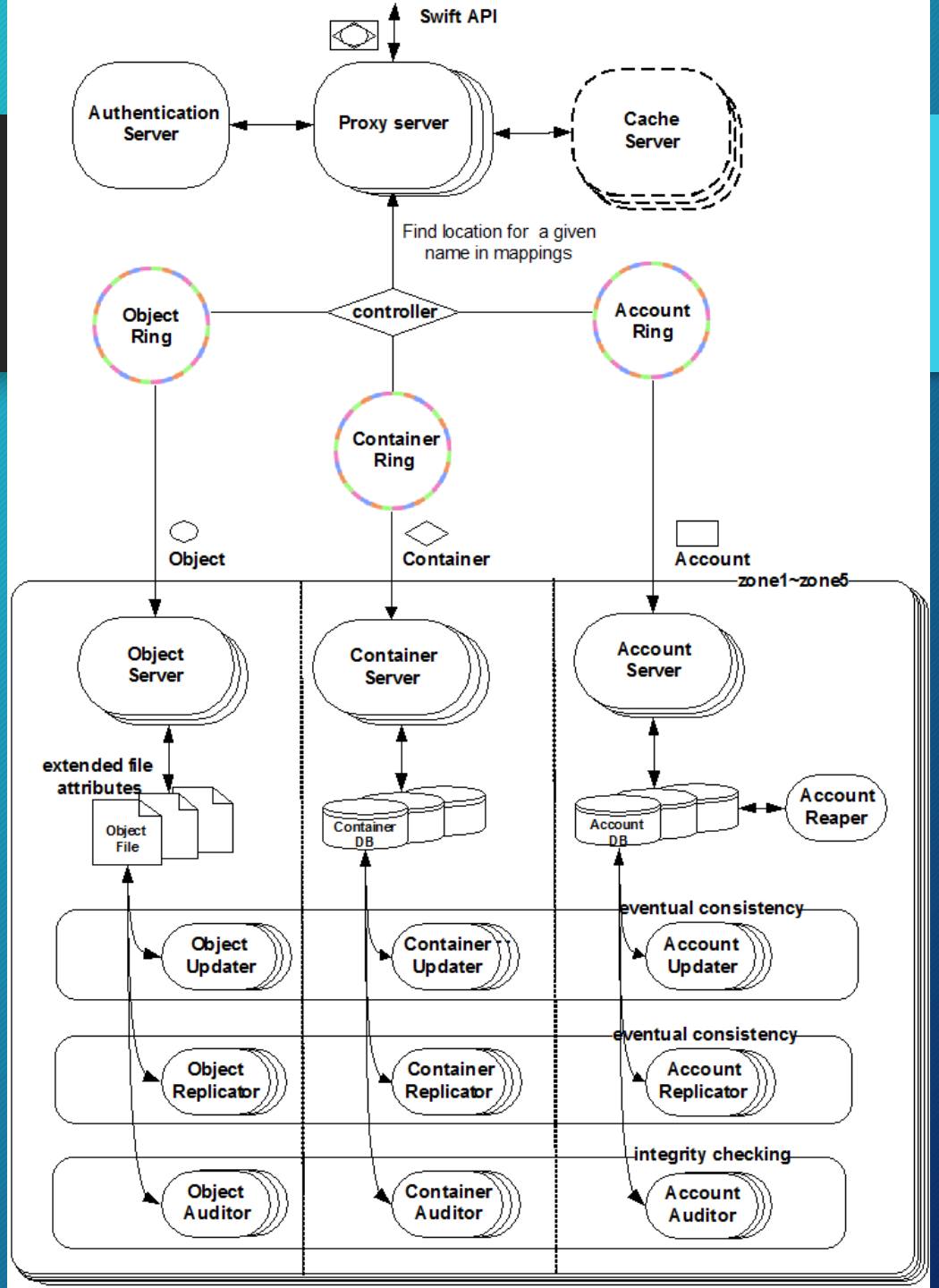
# OpenStack Swift存储概念

- 分布式对象存储
  - Swift专有API
- 最终一致性原则
- 概念体系与Azure Blob Storage相似
  - 存储账号
    - 可理解为一个项目，不同于云管理账号
  - 容器
    - 即其他对象存储中的Bucket
  - 对象
    - 以文件形式直接存储在存储节点上（大于5GB时会分块存储）
    - 对象属性存储于文件的扩展属性中



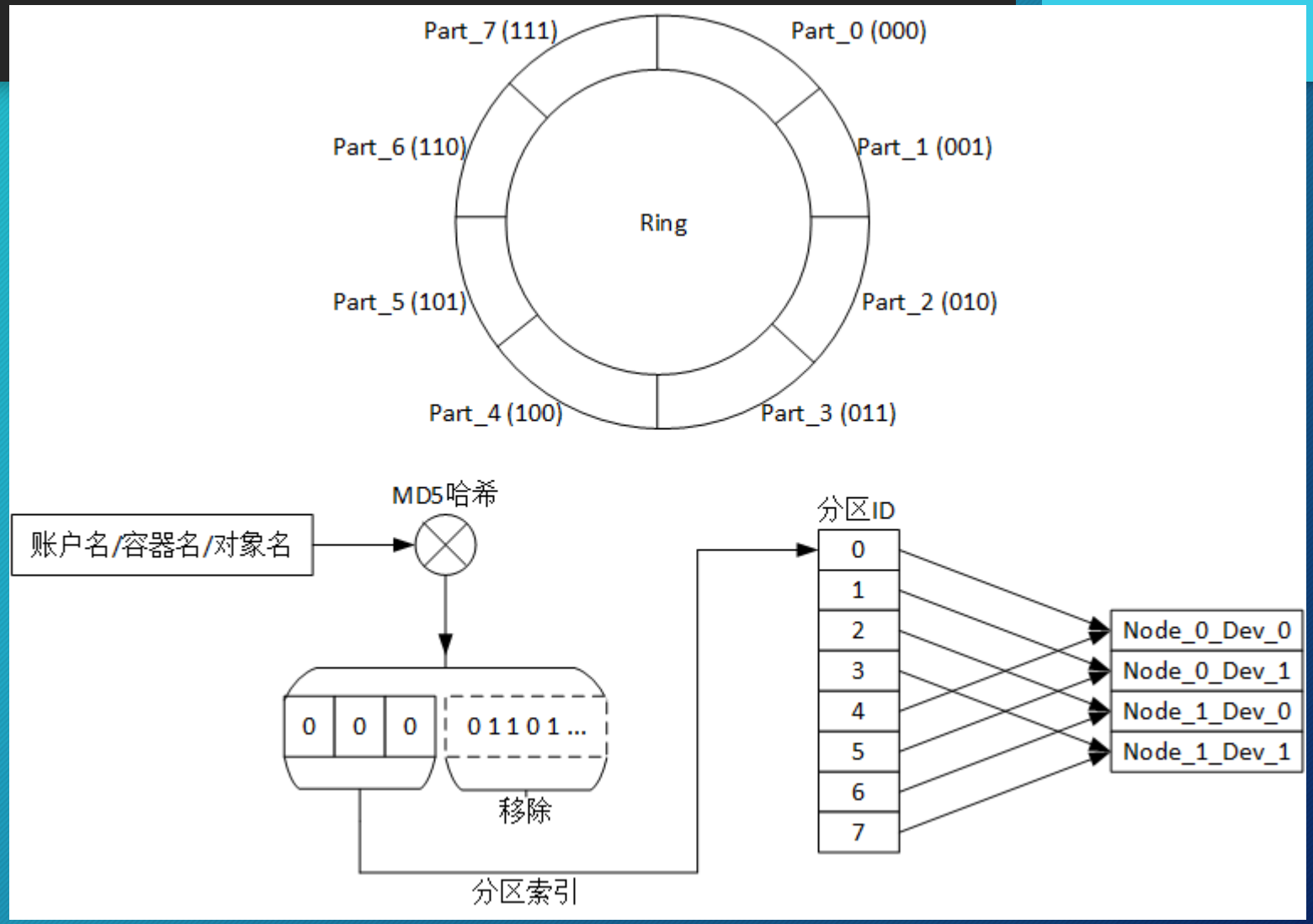
# 总体结构

- Proxy Server
  - 接受用户请求，从哈希环中查找对象数据位置，读取/写入存储节点。
- Ring (哈希环)
  - 对象存储到物理位置的映射关系。该映射会考虑数据的对称分布、失效域、副本数量等问题。
- Object Server
  - 以文件形式存储对象数据。对象属性使用文件扩展属性来存储。文件名包含对象名字的哈希值和时间戳。
- Container Server
  - 存储一个container下的全部对象列表（使用sqlite数据库），但并不知道这些对象的存储位置。
- Account Server
  - 存储一个account下的全部container的列表（使用sqlite数据库）



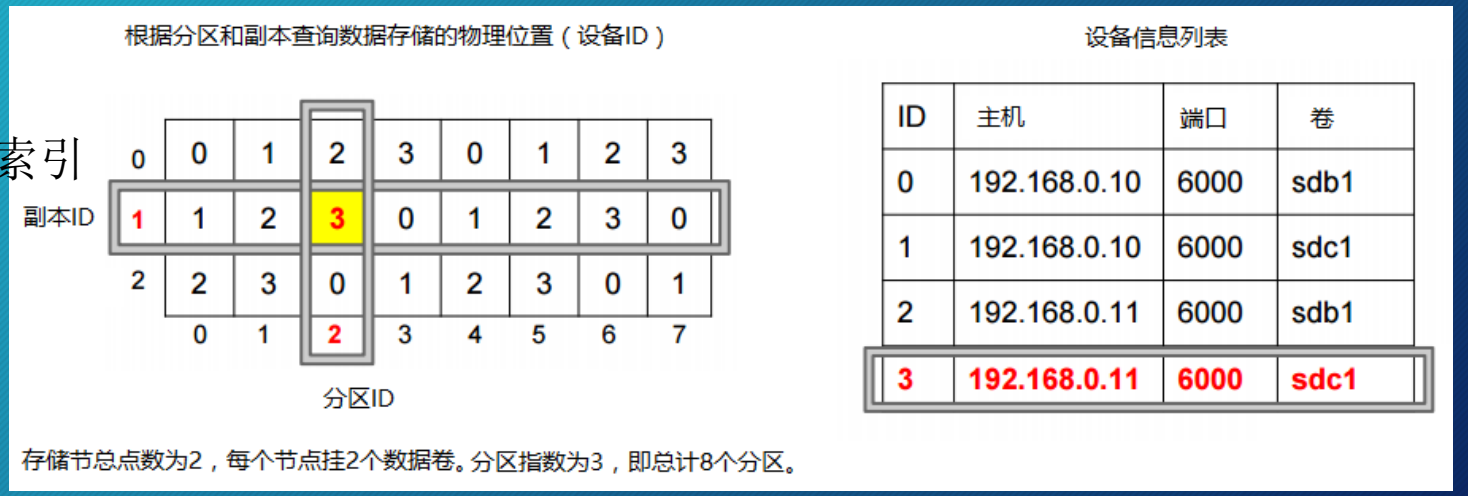
# Ring简介（一致性哈希）

- 固定取哈希值的若干位(partition power)，并将其形成的数字分布在圆环空间里，每个部分叫作分区(partition)
- 将设备编号，并建立“副本”到“分区”到“设备”的对应关系
  - 此对应关系需要将副本分布在不同的“失效域”里
- 根据对象路径的哈希值查询到其应该存储的位置
- 每个Swift对象（Account/Container/Object）都有独立的Ring，每个Storage policy有独立的Ring

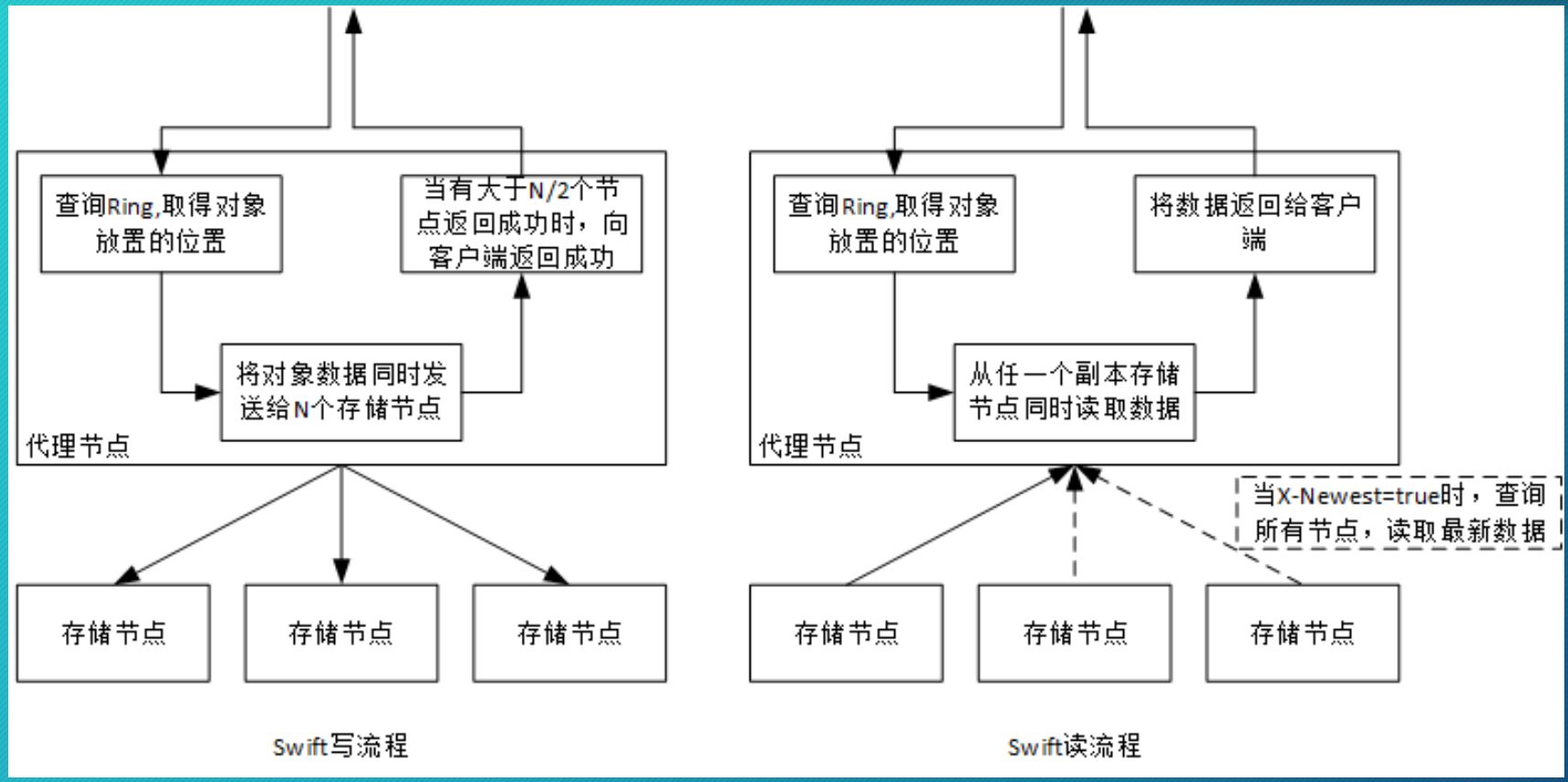


# Ring简介（一致性哈希）

- Ring实际上是一个由外部工具生成的静态文件，需要被分发到集群的各个服务器上
- 文件的内容为Python的结构体，主要包含：
  - 设备信息列表
  - “副本”到“分区”到“设备”的映射表
  - 分区指数，该参数决定：
    - 一共会有多少个分区
    - 集群最多可以有多少个物理节点
    - 截取哈希值的多少个bit位作为分区索引



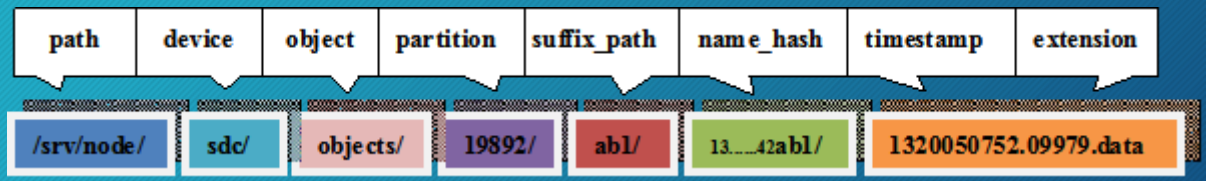
# 主体读写流程




Swift存储在文件系统上的文件名带有时间戳，可以用来区分对象的新旧版本。
当X-Newest=true时，虽然查询对象最新版本，但仍属最终一致性。

# 对象副本复制

- 在partition目录下存有一个哈希列表文件hashes.pkl，其中每一项对应每个suffix_path的内容哈希
- 计算suffix_path内容的哈希值，并更新hashes.pkl
- 与该partition其他副本的hashes.pkl对比，确定哪些suffix_path需要同步
- 使用rsync进行suffix_path目录内容同步




对象文件路径结构：
**Name_hash**=MD5(“/account/container/object” + HASH_PATH_SUFFIX)
**Suffix_path** = name_hash[-3:]

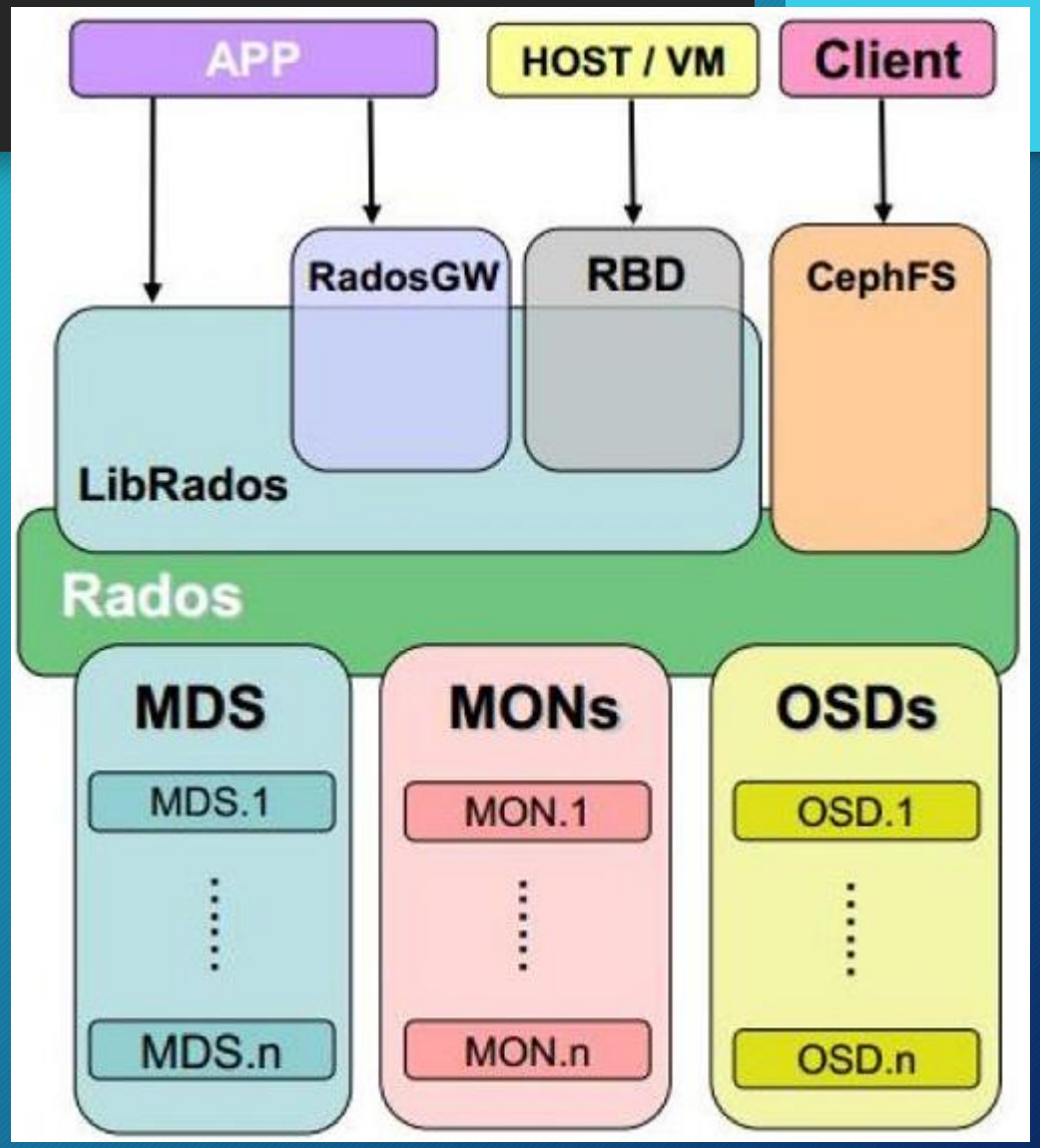
# 特点分析

- 优势
  - 支持多地域(region)部署：同时支持“跨地域集群”和“两个独立集群的容器之间跨地域复制”。后者是部分公有云存储采用的异地容灾策略。
  - 与OpenStack生态系统结合紧密
  - 服务器软件设计得易于插件扩展
- 劣势
  - 最终一致性策略，需要客户端程序特殊处理
  - Swift独有的访问接口
    - 需通过额外代理支持S3访问接口
  - 只支持对象存储
  - sqlite数据库限制同一容器下对象的数量
  - 对象复制过程可能会导致存储结点性能下降
- 限制
  - 不支持随机写和附加写操作


ceph

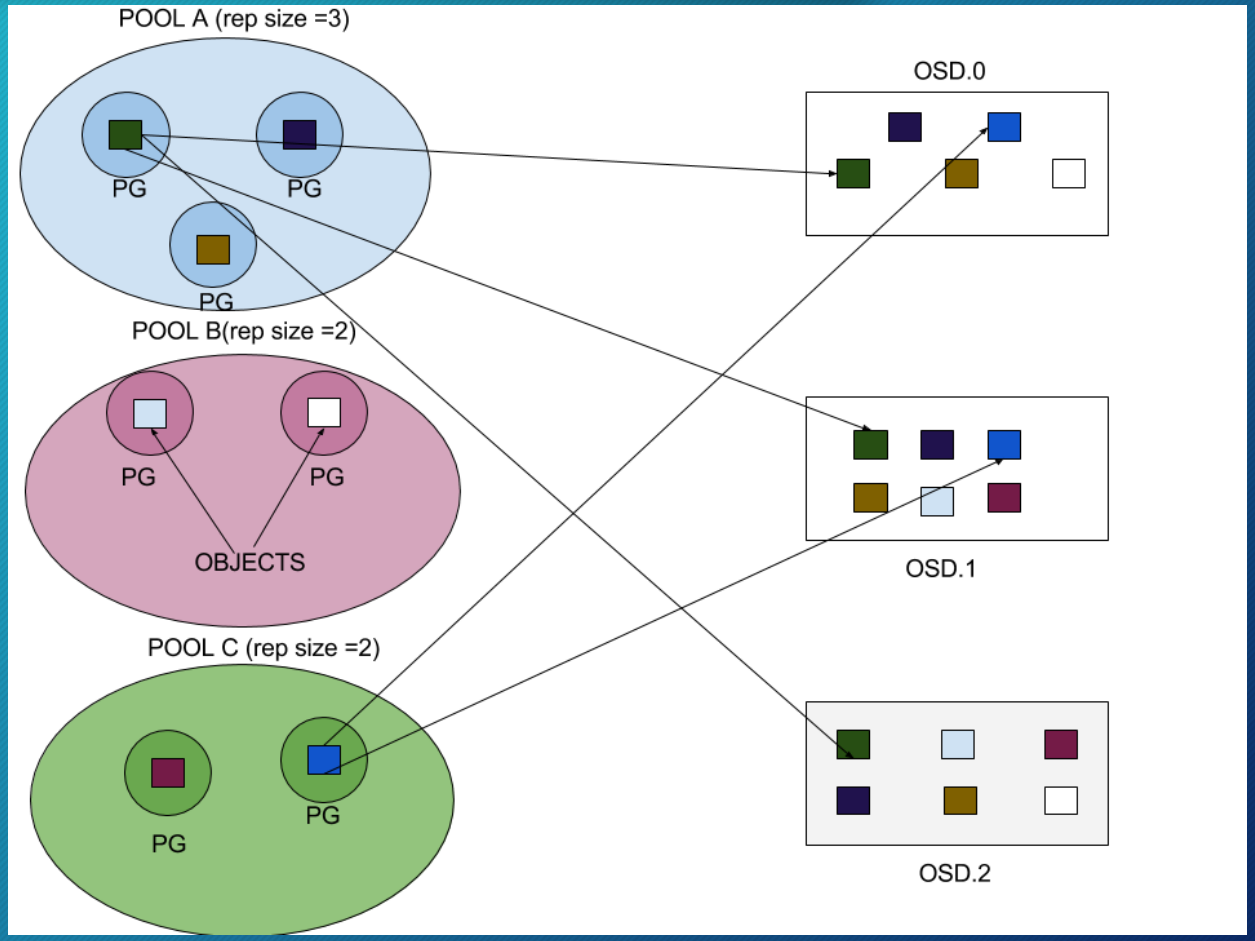
# 总体结构

- 同时提供块存储、对象存储、文件存储
  - RadosGW: 对象存储服务，支持S3和Swift接口
  - RBD: 块存储服务，可作为虚拟机的虚拟硬盘
  - CephFS: 分布式文件系统
- 也可以使用LibRados编程访问集群存储
- 服务器端
  - MON集群监控服务。保存有集群中的关键信息（供客户端和OSD使用）。多个MON实例通过Paxos协议协调，保持高可用。
  - OSD存储节点。OSD彼此会监控对方的运行状态，并汇报给MON。
  - MDS为CephFS提供元数据服务。目前只对Active+Standby模式提供产品级支持。



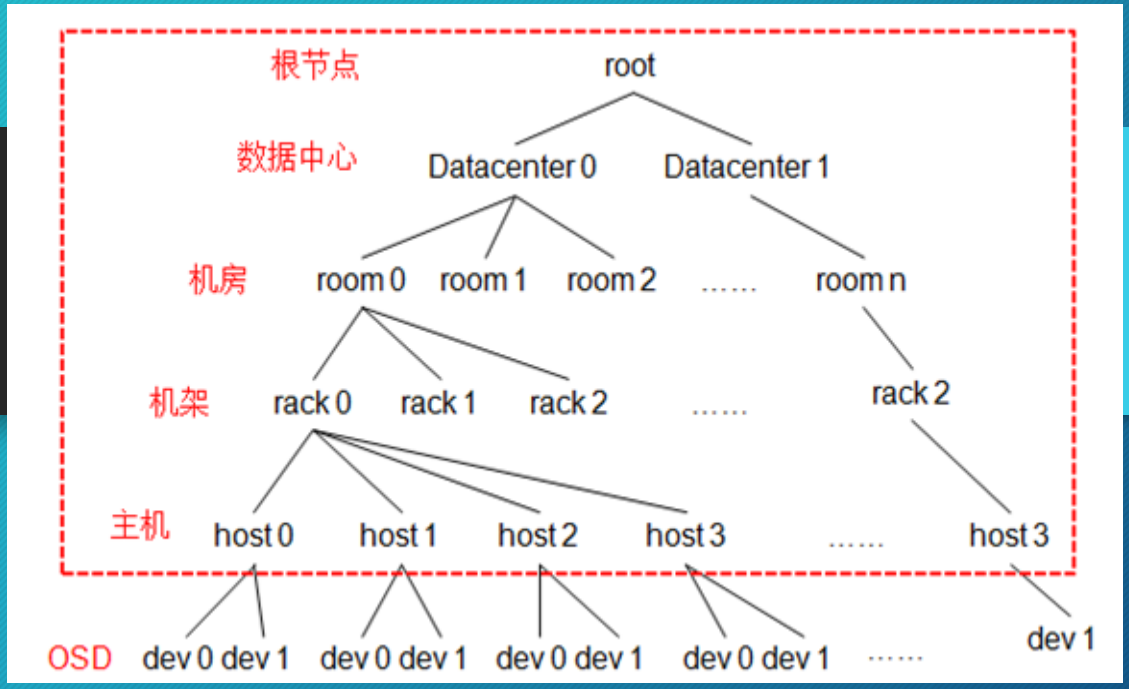
# 关键概念

- Pool
  - 存储一组Object，它设定了一些属性，如副本的数量，PG的数量，数据分布规则等。
- Placement Group (PG)
  - 是一个逻辑概念，用于将一组object分布在一组OSD(由副本数决定)上。它也是数据迁移的最小单位。
- Object
  - 最小的存储单元，所有的数据都以object存储，缺省大小为4MB。
- Object Storage Device (OSD)
  - 用于存储Object的物理设备（分区或目录）
  - 一个主机上可以有多个OSD



# 数据分布方法 - 定义规则

- 设备列表
  - 一个设备可对应一个ceph-osd进程。
- 设备分组(bucket)类型，如机房、机架、主机等。后续将使用这些类型定义存储节点的组织结构。
- 定义存储节点的组织结构。这通常会是一个树形结构。
  - 每个非叶子节点都可以定义其下每个子节点的权重，以及选择子节点时所用的算法。
- 定义选择存储节点的规则(ruleset)。规则主要包含3个关键操作：
  - Take：在设备树种选择查找的起点
  - Choose：选择一定数量的指定类型的设备
  - Emit: 输出查找结果

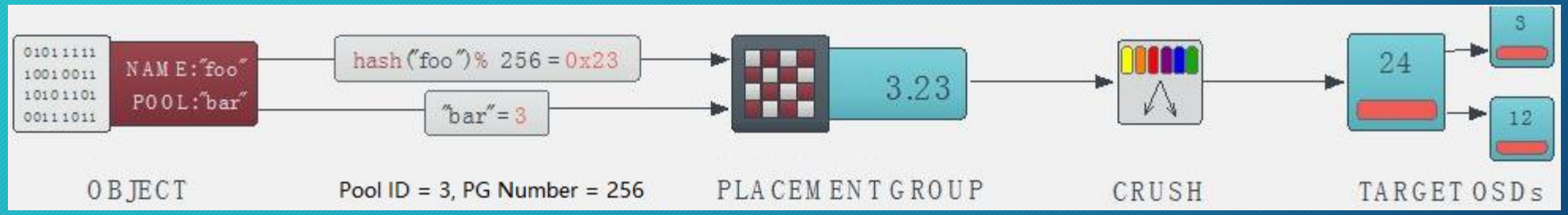



```
root default {
        id -1           # do not change unnecessarily
        # weight 2.998
        alg straw
        hash 0  # rjenkins1
        item wanzh02-ceph-mon weight 0.999
        item wanzh02-ceph-osd2 weight 0.999
        item wanzh02-ceph-osd3 weight 0.999
}

# rules
rule replicated_ruleset {
        ruleset 0
        type replicated
        min_size 1
        max_size 10
        step take default
        step chooseleaf firstn 0 type host
        step emit
}
```

# 数据分布方法 - 选择OSD列表

- 当收到对象读写请求时：
  - 计算对象名字哈希值并与PG数量取模，然后将结果和Pool的ID组合获得PG的ID。
  - 使用PG ID，经过CRUSH算法获得保存数据的OSD列表。第一个OSD为Primary OSD。
    - CRUSH算法使用此前定义的规则在设备树中选择由pool size指定数量的OSD
  - 从OSD map中查找选定OSD对应的连接信息(如IP地址、端口等)
  - 在OSD上，数据存储于以PG ID命名的目录下。

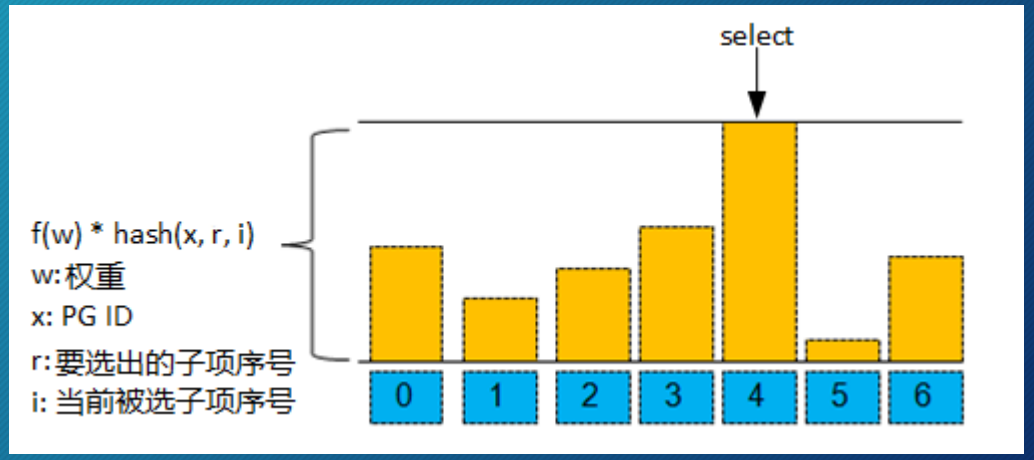


# 数据分布方法 - CRUSH算法选择OSD

- 设备树中一个父节点下的子节点的选择（一个bucket下子项），CRUSH算法会：
  - 四种选择算法，各有不同适应场景
  - 缺省方法为Straw(抽签算法)
    - 以PG ID、当前要选择的子项序号r、当前被选择的子项编号i为输入，计算哈希值，得到一个伪随机数(摇签)
      - 如果选择失败（如OSD主机停机，或者被选出的OSD已经在结果集中），则r+=1重新选择
    - 计算基于权重的函数与哈希值之积，选择结果最大者(抽签)
    - 也就是说：在考虑权重的情况下，确保所有子项机会均等
- 设备树中不同层级的节点选择时，使用同上的选择方法，并通过Ruleset来将副本分布在不同的失效域中。上一级的选择结果作为下一级的输入，最终选择出与要求的副本数量相一致的OSD

| Action | Uniform | List | Tree | Straw |
|---|---|---|---|---|
| Speed | O(1) | O(n) | O(log n) | O(n) |
| Additions | poor | optimal | good | optimal |
| Removals | poor | poor | good | optimal |

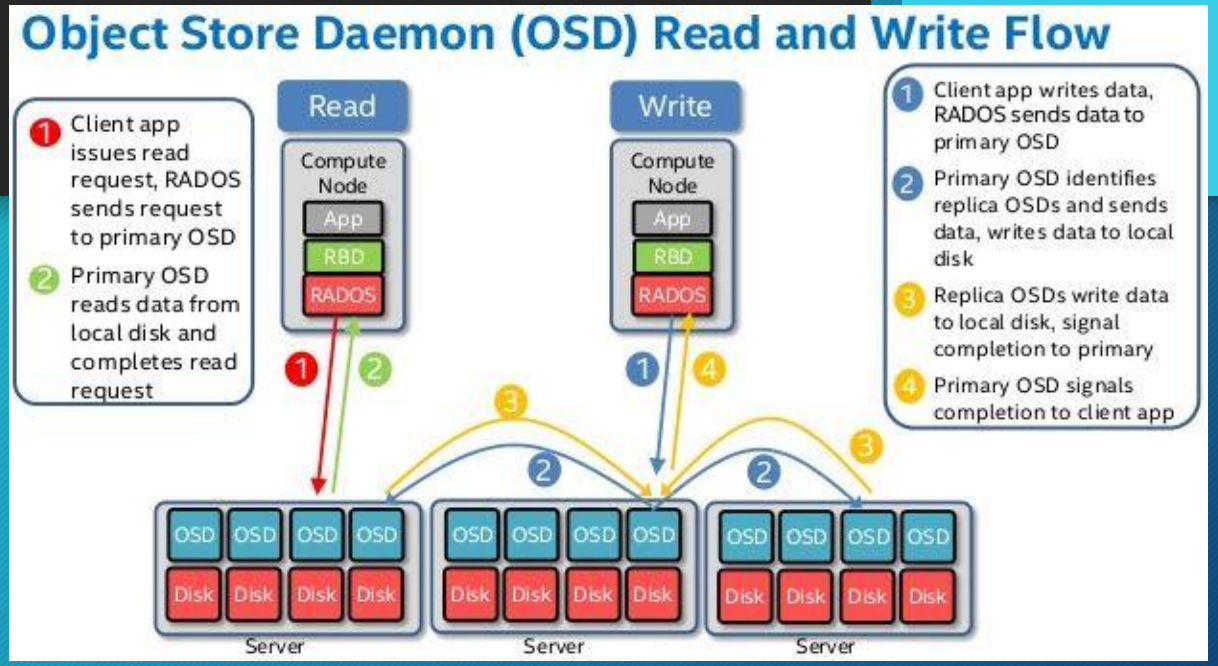
四种选择方法对比

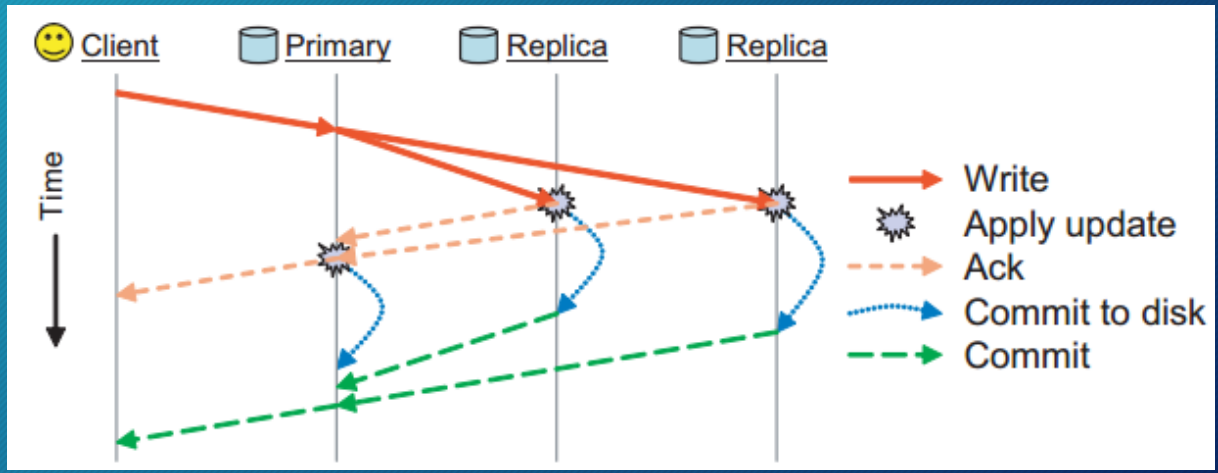



Straw方法示意图

# 读写主体流程

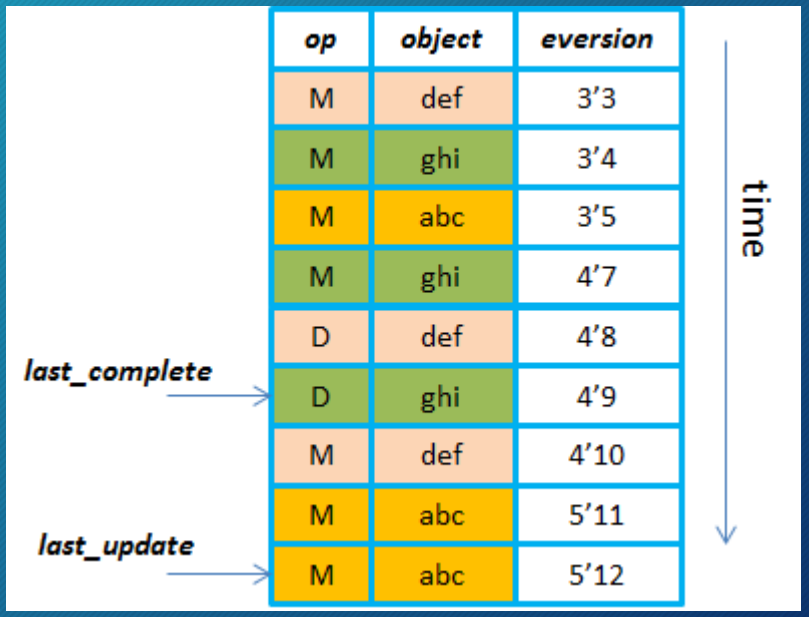
- 读操作
  - 客户端发送请求给主OSD
  - 主OSD从本地磁盘读取数据并返回给客户端
- 写操作
  - 客户端发送数据流到主OSD
  - 主OSD写本地日志，同时转发数据到从OSD
  - 从OSD写本地日志并通知主OSD
  - 主OSD通知客户端数据已保存
    - 客户端写函数此时返回(写完成)
  - 主OSD和从OSD同时提交数据到最终存储
  - 主OSD通知客户端数据提交完成(数据可读)



# 存储结点变动时副本之间的一致性

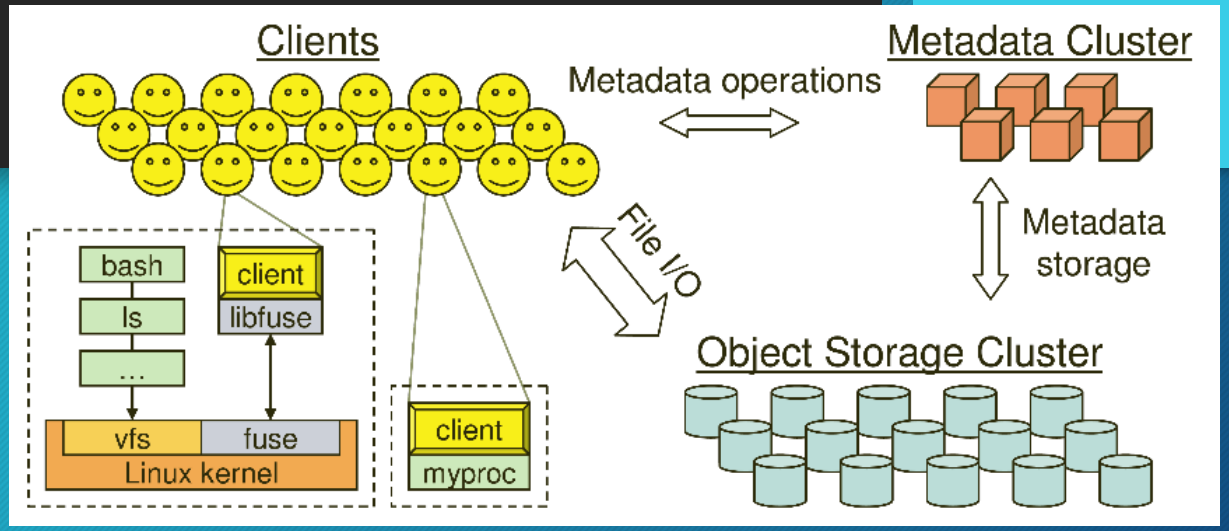
- 每个PG副本都维护PGLog用于记录近期修改操作
  - 只记录操作，无具体数据
  - 时间戳：eversion_t = epoch + version
    - Epoch: OSDMap的版本，每当有OSD状态发生变化时，都会加1
    - Version：PG更新的版本号，由Primary分配并且递增
- Primary故障后：
  - 一个replica OSD作为临时primary
  - 原primary重新启动后，检查各副本的PGLog，找出权威日志，根据权威日志确定数据缺失列表(peering过程)
  - 最后根据缺失列表进行数据恢复
- Replica故障后：
  - Replica OSD再次启动后会由Primary 发起peering过程，根据权威日志(即自身)构建数据缺失列表，然后对replica发起数据恢复过程。
- 注意：一个OSD既是一部分PG的Primary，同时也是另一部分PG的Replica



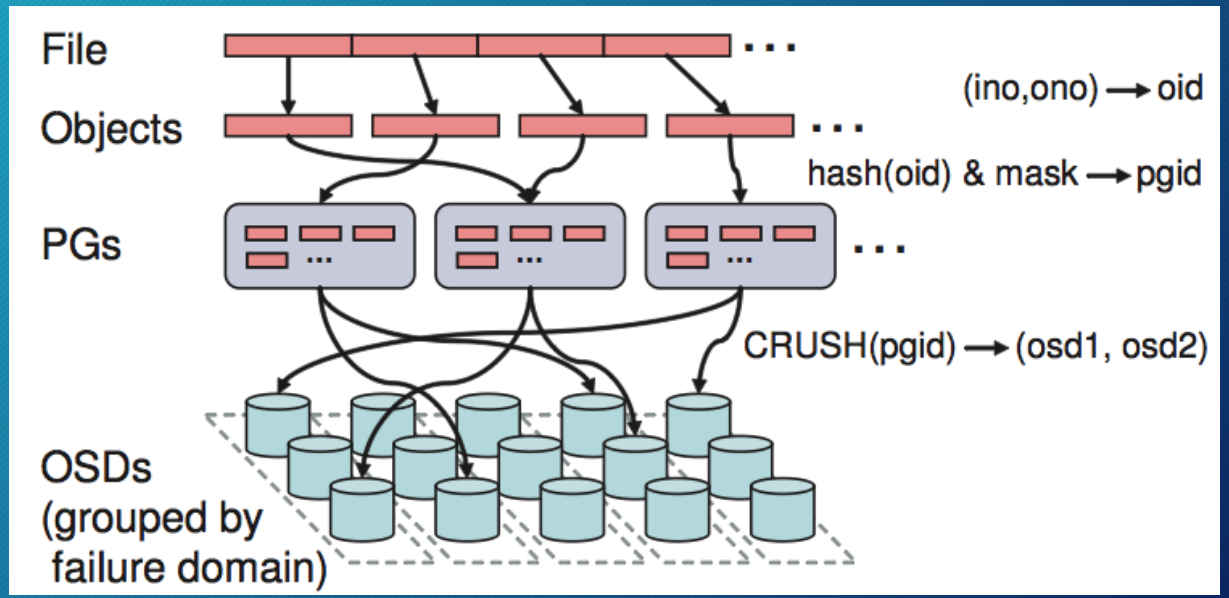

last_complete：最新的已完成全部副本复制的对象版本
last_update：PG中最新的对象版本

# CephFS

- Client
  - 通过CephFS驱动/FUSE程序/SDK访问文件系统功能。
  - 该文件系统接近于POSIX标准，但加入了数据一致性和性能之间的权衡（如支持O_LAZY, 和readdir cache等）
- OSD集群
  - 用于保存文件数据和元数据。
  - 文件内容数据是在切分后，直接保存在OSD上的，无需经过MDS节点。 Ceph对象文件名包含inode number 和 strip number.
  - 使用CRUSH算法，无需记录每个文件包含哪些数据块。
- MDS集群
  - 用于管理元数据，权限、同步等。
  - 可与OSD集群共享同一组物理节点。目前官方只支持Active-Standby模式。
  - 可根据元数据规模、访问频度，自动分裂合并元数据（Dynamic Subtree Partitioning）。




注：上图未画出通过CephFS驱动访问的方法。



# 特点分析

- 优势
  - 同一软件支持多种存储模式：块存储、对象存储、文件存储
    - 对象存储同时支持S3和Swift访问接口
  - 强一致性策略，易于使用
  - 新的bluestore可以将数据直接存储在块设备上，减少性能损耗
  - 已有很多较大规模部署案例
- 劣势
  - 基于算法的伪随机数据分布，不利于极端情况下的手工数据恢复
  - 对象存储在对象列表管理上仍存在性能问题

# 功能对比

# 存储服务与访问接口

| | HDFS (v2.8) | GlusterFS (v3.8) | OpenStack Swift (v2.13, Ocata) | Ceph (v12.0) |
|---|---|---|---|---|
| **存储服务：** | | | | |
| 文件存储 | Yes | Yes | No | Yes |
| 对象存储 | No | No | Yes | Yes |
| 块设备存储 | No | No | No | Yes |
| **访问接口：** | | | | |
| mount访问 (FUSE驱动) | No | Yes | No | Yes |
| mount访问 (内核驱动) | No | No | No | Yes |
| 通过SDK访问 | Yes | Yes | Yes | Yes |
| NFS访问接口 (用户态NFS集成SDK) | Yes | Yes | No | Yes |
| CIFS访问接口 (samba VFS集成SDK) | No | Yes | No | Yes (由samba团队提供) |

# 基本功能

| | HDFS (v2.8) | GlusterFS (v3.8) | OpenStack Swift (v2.13, Ocata) | Ceph (v12.0) |
|---|---|---|---|---|
| 文件随机写 | No | Yes | No | Yes |
| ACL | Yes | Yes | Yes (账号级和容器级) | Yes (CephFS) |
| 存储空间配额 | Yes (目录级) | Yes (卷级或目录级) | Yes (账号级和容器级) | Yes (CephFS:目录级) |
| 访问频度控制 | No | No | Yes | No |
| 数据加密 | Yes (客户端加密) | Yes (客户端加密) | Yes (服务器端加密) | Yes (rgw:服务器端加密) |
| 数据压缩 | No (MR支持压缩) | Yes (只用于数据传输) | No (有非官方支持) | Yes (rgw:服务器端压缩) |
| 快照 | Yes | Yes (依赖LVM，非自身实现) | No ("对象版本"功能有些类似) | Yes |
| 跨区域复制 | No | Yes | Yes | No |

# 高级功能

| | HDFS (v2.8) | GlusterFS (v3.8) | OpenStack Swift (v2.13, Ocata) | Ceph (v12.0) |
|---|---|---|---|---|
| 元数据管理方式 | 集中管理 | 一致性哈希 | 增强型一致性哈希 | CRUSH算法<br>(Ceph本身无元数据管理，但CephFS目前只对元数据集中管理模式提供产品级支持) |
| 多副本 | Yes | Yes<br>(非缺省配置) | Yes | Yes |
| EC编码 | No<br>(将于3.x支持) | Yes | Yes | Yes |
| 数据一致性模型<br>(当多副本情况下) | 强一致性 | 最终一致性<br>(也可设置强一致性) | 最终一致性 | 强一致性 |
| 数据访问本地优先<br>(data locality) | Yes | Yes<br>(通过glusterfs-hadoop-plugin实现) | No | No |
| RDMA传输 | No | Yes | No | Yes |
| 节点数量变化导致的数据迁移(rebalance) | 最优 | 最差 | 较好 | 优<br>(与CRUSH选择的算法相关) |

# 部署与监控

| | HDFS (v2.8) | GlusterFS (v3.8) | OpenStack Swift (v2.13, Ocata) | Ceph (v12.0) |
|---|---|---|---|---|
| 部署工具 | Yes (web界面部署工具) | No (小规模部署很简单，甚至无需操作任何配置文件) | No (部署复杂) | Yes (命令行部署工具) |
| 滚动升级 (rolling upgrade) | Yes | Yes (非推荐方式) | Yes | Yes |
| 运行状态自监控机制 | Yes (NN监控DN) | No (客户端访问时检测并处理) | No (客户端访问时检测并处理) | Yes (OSD报告给MON) |
| 图形界面监控管理工具 | Yes (web界面管理工具) | No | ? | 不完善 |

注 1：此处只考虑开源免费的管理工具。总体上，HDFS的部署和管理工具比较完善，易于使用。这些工具并非只为HDFS而设计的，而是为Hadoop生态系统设计的。OpenStack有一些第三方部署工具，但仍然很复杂。
注 2：对滚动升级的支持与升级前后的版本相关。

# 开发与扩展

| | HDFS (v2.8) | GlusterFS (v3.8) | OpenStack Swift (v2.13, Ocata) | Ceph (v12.0) |
|---|---|---|---|---|
| 客户端SDK支持： | | | | |
| 实现语言 | Java | C | Python | C++ |
| 支持平台 | 平台无关 | Linux | 平台无关 | Linux, Windows(非官方支持) |
| 授权协议 | Apache 2.0 | LGPL v3/GPL v2 | Apache 2.0 | LGPL v2.1 |
| 服务器端扩展： | | | | |
| 实现语言 | Java | C | Python | C++ |
| 授权协议 | Apache 2.0 | GPL v3 | Apache 2.0 | LGPL v2.1 |
| 插件式扩展 | 不支持 | 支持 | 支持 | 支持，但功能有限 |

# 参考资料

- 各开源项目文档和源码
- HDFS
  - http://hadoop.apache.org/docs/r2.8.0/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithQJM.html#Note:_Using_the_Quorum_Journal_Manager_or_Conventional_Shared_Storage
  - http://stackoverflow.com/questions/33311585/how-does-hadoop-namenode-failover-process-works
- GlusterFS
  - https://my.oschina.net/uvwxyz/blog/182224
  - http://blog.chinaunix.net/uid-22166872-id-4215856.html
  - http://blog.csdn.net/iesool/article/details/48919333
- OpenStack Swift
  - https://www.ibm.com/developerworks/cn/cloud/library/1310_zhanghua_openstackswift/
  - http://www.cnblogs.com/yuxc/archive/2012/07/04/2575536.html
- Ceph
  - http://www.cnblogs.com/chenxianpao/p/5568207.html
  - http://blog.csdn.net/skdkjzz/article/details/51488943
  - http://xuxiaopang.com/2016/11/08/easy-ceph-CRUSH/
  - http://www.sysnote.org/2015/12/18/ceph-pglog/

谢谢