Design and Implementation of Distributed Transaction for Ceph Distributed Storage

Li Wang

li.wang@kylin-cloud.com

What is Ceph

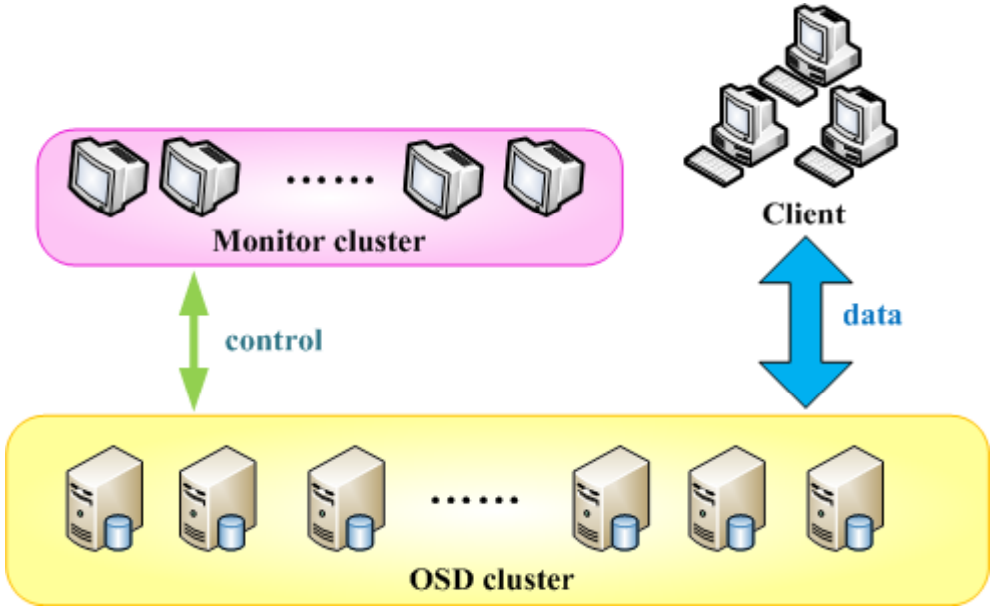
- 开源大规模分布式存储系统
- 提供多种访问接口
  - 块 , 文件, s3, swift, rados
- In Linux Kernel since 2.6.34
- Openstack support since Folsom
  - OpenStack中应用最多的存储后端
- History
  - Sage Weil 在UCSC 的博士课题
  - Inktank
  - Red Hat

What is special

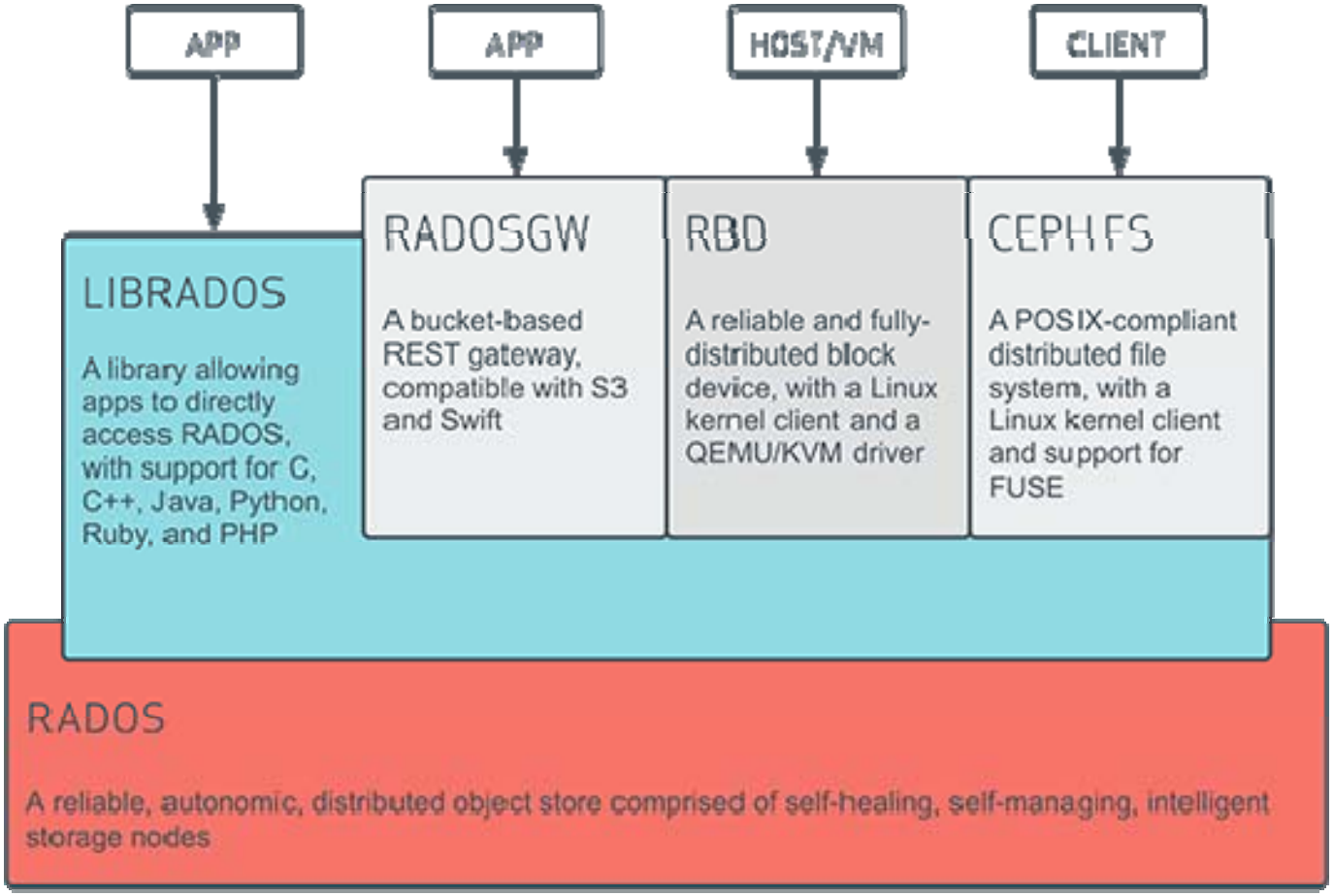
- Scale out
  - No metadata service
  - Object storage
  - Flat namespace
- Software defined
  - Commodity hardware
  - Self healing
  - Self managing
  - No single point of failure
- Object, block and file storage in a single system

集群架构

- Client
  - 提供标准块，文件接口的访问能力
- Monitor
  - 监视和维护整个存储系统的状态和拓扑结构
- OSD
  - 存储数据和元数据



软件架构



http://ceph.com/community/more-than-an-object-store/

软件架构

- 自下向上，可以将Ceph系统分为四个层次：
  - 底层存储系统RADOS
  - 存储访问库LIBRADOS
  - 高层应用接口：RADOSGW、RBD、CephFS
- RADOS
  - Ceph的核心组件
  - 提供高可靠、高可扩展的分布式对象存储架构
  - 利用本地文件系统存储对象
- LIBRADOS
  - 封装对RADOS的访问接口

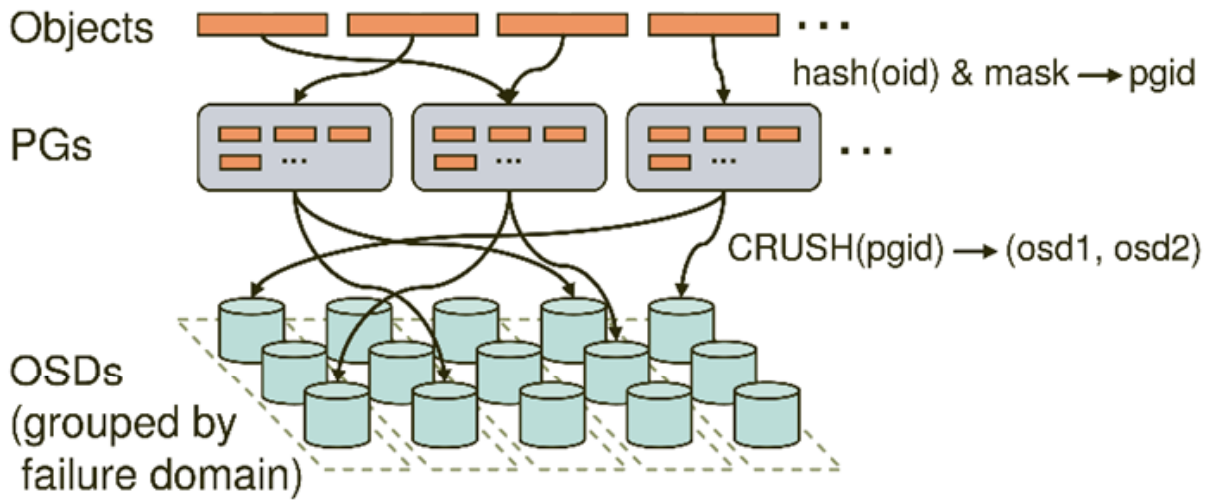
librados对象写操作接口

- 支持对单对象的一系列写操作原子完成, 并在副本OSD之间同步
  - 示例
    - 在对象“sss”的偏移为0的位置，写入“ceph”
    - 在对象“sss”的偏移为16的位置，写入“storage”

```
...
bufferlist b1,b2;
b1.append("ceph");
b2.apeend("storage");
ObjectWriteOperation op;
op.write(0, b1);
op.write(16, b2);
ioctx.operate("sss", &op));
...
```

对象定位

- 两级映射
  - Object 通过Hash映射到PG
    - PG(Placement Group)的引入避免了object与OSD的直接映射，减小了海量对象管理的复杂性
    - RADOS的许多操作是以PG为单位进行
  - PG通过CRUSH映射到一组OSD，其中一个为primary，其他为replica



Journal and PGLog

- 写对象涉及操作
  - 写入对象数据，即文件数据
  - 写入对象元数据，即文件扩展属性或LevelDB文件
  - 写入PGLog，即LevelDB文件
- 涉及到多个系统调用, 多个文件的修改，但都在本地完成
- 基于journal利用write ahead logging来保证多个文件的原子修改
- 操作在replica OSD上复制，为了支持多副本之间的数据一致性, 每个OSD上有一个PGLog, OSD做写对象操作时同时在PGLog中写入一个entry, 包含操作类型和操作的对象名
- OSD故障恢复时, 根据各个副本OSD上PGLog的entry, 可确定最新数据的位置，使数据达到一致
- PGLog存储在kv数据库LevelDB

对象写入

- Client通过计算, 得到Object所在PG的primary OSD
- Client将写操作请求发送给primary OSD
- Primary OSD收到写操作请求，加上对Object元数据的操作以及PGLog操作，构造为一个transaction，将transaction发送给所有replica OSD
- Primary OSD将transaction写入journal, 提交transaction
- Replica OSD收到transaction, 写入journal, 给primary发送响应, 提交transaction
- Primary OSD收到所有副本写入journal响应，给client返回响应

多对象操作场景

- CephFS
  - create
    - 在父目录文件对应对象中加入一个entry
    - 新文件对应创建一个新的对象
  - rename
    - 在原目录文件对应对象中去掉一个entry
    - 在新目录文件对应对象中加入一个entry
- Radosgw
  - 在一个bucket下创建object
  - object versioning
- 需要原子完成， 否则可能影响名字空间一致性
- 上层有自己解决方案，如CephFS的Journal

Motivation

- 存储底层支持事务
  - Isotope: Transactional Isolation for Block Storage (FAST 2016)
    - Existing storage stacks are top-heavy and expect little from low level storage
    - High level system required to implement complex functionality
- 上层无需再各自实现复杂的事务保证机制，设计可以大大简化
  - CephFS
  - Radosgw
- 社区提出
  - Sage at Ceph Developer Summit for Infernalis

多对象写操作接口

- 数据结构
  - 指定master对象
  - 指定一系列slave对象
  - 分别指定各个对象上的一系列写操作

```
class MultiObjectWriteOperation
{
 protected:
  std::map<std::string, ObjectWriteOperation*> slaves;
  void prepare_operate();
  ...
 public:
  ObjectWriteOperation master;
  ObjectWriteOperation& slave(const std::string &oid);
  ...
};
```

多对象操作接口

- 示例
  - 两个对象“vvv”和“xxx”，其中“vvv”为master对象
  - 在“vvv”的偏移为0的位置，写入“abc”
  - 在“xxx”的偏移为0的位置，写入“def”

```
string b1("abc"), b2("def");
bufferlist bl1, bl2;
bl1.append(b1.c_str(), b1.size());
bl2.append(b2.c_str(), b2.size());

MultiObjectWriteOperation writes;
writes.master.write(0, bl1);
writes.slave("xxx").write(0, bl2);
ioctx.operate("vvv", &writes));
```

流程

- 操作涉及多个对象，可能在不同的OSD上，需要原子完成
- 约定
  - MASTER：master对象所在PG的primary OSD
  - SLAVE：slave对象所在PG的primary OSD
  - REPLICA: MASTER, SLAVE所在PG的replica OSD
- Client将整个操作发送给MASTER

Step 1 LOCK

- MASTER remember the transaction
  - MASTER创建一个meta object，记录slave object信息到meta object，meta object不记录任何数据信息
  - 向PGLog写入一个对master object的LOCK entry
  - 上述操作利用单对象操作接口原子完成, 并在MASTER的REPLICA之间同步
  - Why 先persist
    - 如果先把op发给slave, master restart将forget transaction, slave事务无法继续
- MASTER send ops to SLAVE
  - MASTER对每个slave对象构造一个LOCK request，包含该对象上的所有操作
  - 依次将request发送给对应的SLAVE
  - 等待SLAVE响应

Step 1 LOCK

- SLAVE remember the transaction
  - SLAVE收到LOCK request，检查操作的合法性
  - 如果操作合法，向PGLog写入一个对slave object的LOCK entry，创建一个meta object，将操作信息写入meta object
  - 上述操作利用单对象操作接口原子完成, 并在SLAVE的REPLICA之间同步
  - 给Master发送响应
- Why SLAVE不像单对象事务那样, 直接commit
  - 因为单对象的实现, primary先做合法性检查, 再发送操作给REPLICA, REPLICA的操作和数据跟PRIMARY相同, 因此一定是合法的, 不会roll back
  - 多对象的实现, 每个SLAVE操作不一样, 一个操作不合法就需要roll back

Step 2 COMMIT

- MASTER commit
  - MASTER收到所有SLAVE的响应，如果有没通过合法性检查的, 通知所有SLAVE roll back
  - 如果均通过合法性检查，执行在master object上的操作，同时在PGLog中插入一个对master object的COMMIT entry, 该操作利用单对象操作接口原子完成, 并在MASTER的REPLICA之间同步
- MASTER ask SLAVE to commit
  - MASTER向SLAVE发送COMMIT request
  - 向client发送响应

Step 2 COMMIT

- SLAVE commit
  - 执行在slave object上的操作，向PGLog写入一个对于slave object的COMMIT entry
  - 向MASTER发送响应
- MASTER能否同时提交self与SLAVE
  - SLAVE开始提交后无法roll back
  - MASTER不确定能够提交成功，MASTER没有persist data, 可能提交失败

Step 3 UNLOCK

- SLAVE unlock
  - SLAVE commit成功之后，删除meta object，并在PGLog写入一个对于slave object的UNLOCK entry, 以上操作利用单对象操作接口原子完成
- MASTER unlock
  - MASER收到所有SLAVE commit响应， 删除meta object，并在PGLog写入一个对于master object的UNLOCK entry，以上操作利用单对象操作接口原子完成

Dead Lock Avoidance

- 假设有两个并发事务T1和T2, 都写对象{X, Y}
- T1对两个对象赋值为1，T2对两个对象赋值为2
- X和Y分别在OSD_X, OSD_Y上，在T1中，指定X为master object, 在T2中，指定Y为master object
- 根据事务的隔离性要求，只有两个结果是合法的，X=Y=1, 或者X=Y=2
- 放行
  - T1: lock x T2: lock y
  - T1: lock y T2: lock x
  - T1: x=1 T2: y=2
  - T1: y=1 T2: x=2
  - y=1, x=2

Dead LOCK Avoidance

- 等待
  - OSD_X收到T1, LOCK X, 然后发送LOCK Y请求给OSD_Y
  - OSD_Y收到T2, LOCK Y, 然后发送LOCK X请求给OSD_X
  - OSD_X收到T2 LOCK X请求, 发现有一个事务T1已经LOCK X，则令T2等待
  - OSD_Y收到LOCK Y请求, 发现T2已经LOCK Y，则令T1等待
- 算法
  - 当OSD收到对某个对象的多对象操作请求，如果该对象上有正在进行的多对象操作,返回EDEADLK, 除非以下情况
    - 在新来的操作请求中,如果该对象是master对象,则新来的请求可以等待
    - 在新来的操作请求中,如果该对象是slave对象,且正在进行的多对象操作已经进入了COMMIT阶段,则新来的请求可以等待

优化

- 两个对象应用场景最多
  - 两个并发操作T1和T2, 分别在同一个bucket下创建一个object, T1操作对象{X, Y}, T2操作对象{X, Z}
  - Master object只写一次, 从性能考虑, 最好设置Y,Z为master, X设置为slave
  - 在X上有两个并发多对象事务, 在上述死锁检测算法下, 第二个事务可能返回EDEADLK, 实际上不会死锁
- LOCK request给SLAVE传递一个整个事务操作的object数量, 如果数量为2, 如果新来的事务和正在进行的事务中该对象都是slave, 则等待

容错处理

- 出错重启时，OSD根据PGLOG entry和meta object信息在内存中重建事务状态
- MASTER restart
  - If the transaction is in LOCK state
    - (1) MASTER sends UNLOCK to SLAVE
    - (2) SLAVE receives UNLOCK, if the transaction does not exist or in UNLOCK state, goto (3) ; If the transaction is in LOCK state, SLAVE do UNLOCK process
    - (3) SLAVE sends ack to MASTER
    - (4) MASTER collects ack, and rolls back itself

容错处理

- MASTER restart
  - If the transaction is in COMMIT state
    - (1) MASTER sends COMMIT to SLAVE
    - (2) SLAVE receives COMMIT, if the transaction is not in LOCK state, goto (3); Otherwise, proceed as normal
    - (3) SLAVE sends ack to MASTER
    - (4) MASTER collects ack, proceed as normal

容错处理

- SLAVE restart
  - If the transaction is in LOCK state
    - Slave waits MASTER to re/send LOCK/UNLOCK/COMMIT
  - If the transaction is in COMMIT state
    - SLAVE does the UNLOCK process
    - If MASTER resends COMMIT, it directy sends ack to MASTER

其他

- 与PGLog操作的交互
  - Ceph会定期Trim对应操作已经完成且已在replica之间同步的PGLog entry
  - 不能trim还没有UNLOCK的transaction的LOCK, COMMIT entry, 因为这会forget transaction
  - 如果PGLog积累过多, 通过在末尾复制entry的方式允许trim之前的entry
- 与Tiering的交互
  - Ceph支持两层存储介质构成tiering, 上层做为cache
  - PGLog无法在上下层之间同步, 难以保证数据一致性
  - 写对象的时候, 如果上层对象不存在, 强制提升对象到上层
  - 多对象操作进行期间, 不允许其操作的对象从上层evict，meta object为临时对象，不flush到下层

Thanks!