Ozone设计文档

Oozie是类Amazon S3的对象存储，Key和Object的数据格式都是任意的比特数，Key的大小尽量小于1K，Value可以从几百Kb到几百MB，但是没有严格的限制。Key和Object存储在buckets中，bucket有唯一的key集合，bucket放置在Storage Volume中唯一标识。Storage Volume的名字全局唯一，而且有一定配额。在私有云中，管理员可以为用户、项目及租户创建Storage Volume，并且为volume设置配额。在公有云中，可以为每个云账户创建多个storage volumes。在Ozone实现中，Storage Container Manager进行NameSpace和Block Space的管理，具体的实现为Key Space Manager(KSM)。在Ozone中，几个关键模块如下：

1. Shared Container Layer

Datanode提供一个共享存储服务（Container Layser）,借鉴HDFS的经验，被设计成全分布式服务，无单点问题。Container Layer包括两个关键组件：

1. Storage Containers，副本单元
2. Storage Container Manager（SCM）, replicated Service，用于追踪每个container的副本。

每个Container都是一个Replication单元，每个Container都是一个key-value存储，每个container中keys唯一。

1. Ozone Namespace Management，

Ozone NameSpace Management由KSM管理，SCM管理Container。将KSM作为一个单独的服务有一下好处：

1. HDFS一直期望将namespace management从block space management分开，所以反方向的很难实现

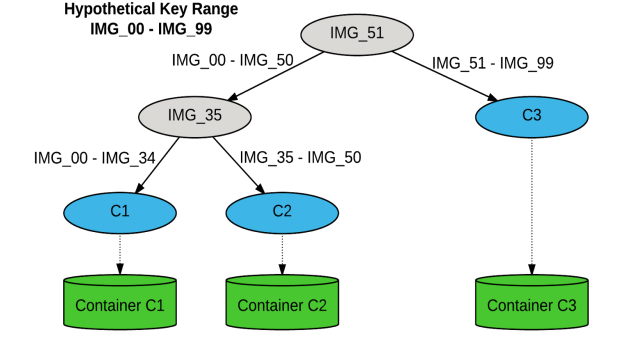
b）block space访问随机，但是namespace具有本地行，因此数据结构和最优化结构很难做。

c) 将这两功能分开，可以启动多个KSM，类似于hdfs federation，由于ozone不支持重名名和迁移，因此很容易将Ozone namespace分成多个KSM实例。

d) KSM的扩展性，可以使ozone cluster规模扩大，存储模块达到百万级别

**Buckets Spanning Containers**

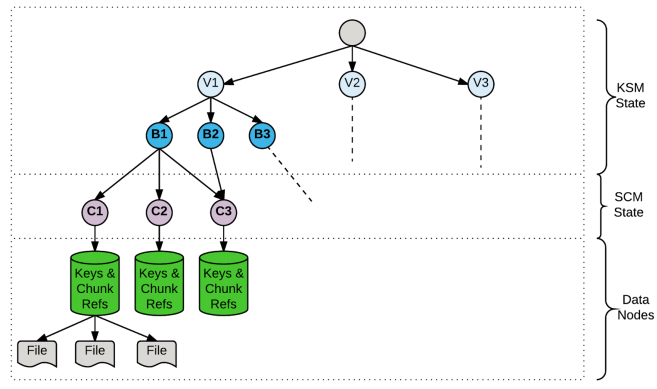
当container中的bucket有更多数据，KSM允许bucket可以扩展多个containers，可以从KSM中获取这些信息。因此，当查询ozone下的name，获取到的是container的路径，KSM可以看出name与container的映射服务。Bucket Spanning Containers的系统架构图如下所示：



在KSM中维护了namespace的信息树，而不是全局的namespace信息。KSM不自己维护keys，这部分由container来完成。在上图中，KSM也维护bucket中的split point的状态。KSM能够向Oozie Client发送准确的container位置。

下面举个不常见的例子，Bucket频繁的split，极端的例子是KSM中的实体和keys数目相同。以ETL的例子中，用户上传了大量的文件，并且删除这些文件。KSM在上传文件时会split bucket，但是当删除这些文件时并不会合并bucket，因为后面会根据key的name上传文件。这会增加KSM中的元数据，但是使用B+树不会让元数据增加。如果考虑的内存，将KSM spin up，并且以federation没使用。

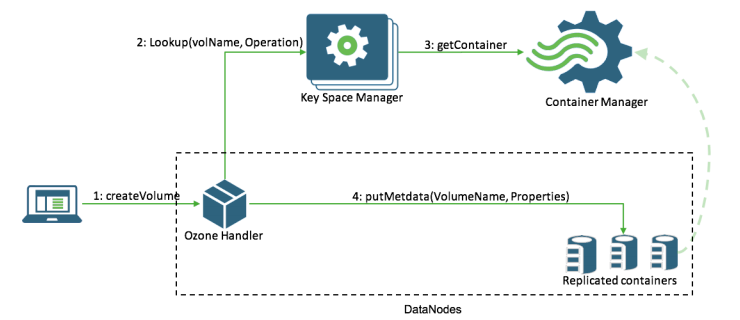
下面的DAG图展示使用KSM的Ozone Cluster的分布式状态：



在上图中，KSM维护namespace，大部分是volume信息及bucket信息。V1,V2,V3对应ozone中的Volumes，在ozone namespace中为根对象，如果不使用多KSM，SCM还是有相同的树结构，但是KSM就比较类似于NameNode。

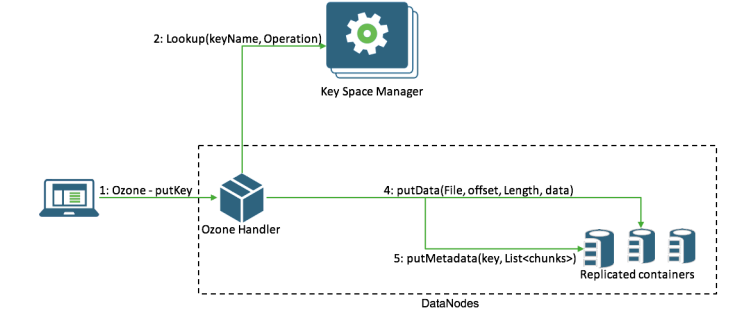
**KSM Workflows**

SCM管理Container生命周期，KSM从SCM中获取container位置。SCM获取container所在的datanode。这些主机可能是随机的，可能是使用复杂的算法如copysets。使用copysets可以减少数据丢失。下表是Ozone中KSM的执行流程：



1. 客户端向Ozone 前端发送createVolume
2. Ozone Handler或者REST前端，向KSM请求，然后请求转发到SCM，获取container的位置。
3. SCM向KSM返回container location，这是因为datanode与scm交互，通过心跳将container report传送给SCM
4. 基于container reports，包括container size,keys和操作数目，SCM选择一个KSM可以使用的container
5. 当volume create结束后，KSM更新期维护的元数据树结构，同时将信息写到container中，当出现KSM或者SCM数据丢失时，基于这些信息构建ozone的集群状态，

当创建bucket后，会执行相同的work flow。对于ozone的key写操作，有一个附加的操作，Ozone Handler会写入chunks，并更新key，如下图所示：



1. Ozone Client 插入key
2. Ozone handlesr从KSM中获取包含该key的container
3. Ozone handler写入chunks集合，数据通过简化的data pipeline写入多副本
4. Ozone handler写元数据，通过RAFT获取副本，当更新后，在Ozone namespace中可以查到key

KSM负责管理Quotas，key space与container的映射关系，是Ozone的关键组件,Ozone的操作及高可用性都是通过KSM来实现。

**Quotas under Ozone**

Ozone支持voloume level的quotas，因为volume是跨containe。KMS并不涉及到写，进告诉客户端向哪个container中写入数据。Blocks的分配在container level上完成。

但是由于缺少类似namenode的集中化namespace manager，实现像hdfs的集中化quotas控制很难。在Ozone中，SCM获取container reports，获取每个container中的数据量。KSM周期的获取这些信息，并实现volume级别聚合。因此当quota violation变化，KSM会有一定程度的滞后。

**Container Splits**

当bucket中的keys较多，container中存储的数据量会超过配额，因此会触发container splits，一般基于key range。典型例子是用户上传大量的image文件到bucket中，默认的container 大小为5G，超过5G, KSM进行container的split。当container执行split后，会更新SCM中的信息，然后KSM从SCM中获取信息并更新KSM状态。Container Split是由Container自己触发的。

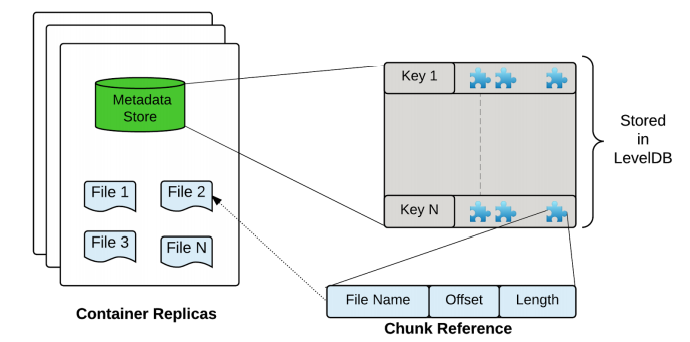
**Container Replication Pipeline**

Container负责元数据和数据的存储，数据存储使用的pipeline相对于HDFS replication为简化的pipeline。对于container framework，replication pipeline有两个条件：

1. Chunks immutable
2. Keys提供序列号，可以通过RAFT来实现

**Ozone Key Write**

首先看下不使用RAFT算法怎么写入Keys，然后使用RAFT来实现一致向。在Container中Key(ey 1)对应chuncks引用列表（ChunkInfo），指向data的block：



为了写一个OzoneKey，将数据写入，然后更新key值，这些key指向chunks的列表，该key可以在container namespace中可以使user data可见。例如：

1. 向一个chunk中写入100byte数据，在offset 0处写入block1文件，长度为100bytes。但是没有改变name space，这是因为Metadata没有改变
2. 使用下面的信息来更新key值，.putKey(OzoneKey -> {{filename:block1, offset:0,length:100}. })。这个步骤是完成OzoneKey写操作，同时数据也写入了chuck中，使用KeyEntry更新数据库，KeyEntry标识datablocks
3. OzoneKey可见

如果其他客户端想要读取OzoneKey，读取OzoneKey的元数据，记录哪些chunks可以读。这也是单节点ozone向container中读写数据。下面会介绍怎么通过一致性协议来实现，使用RAFT和Simplified，逻辑如下：

1. 向chunk中写入100bytes
2. 通过pipeline向另外两个节点复制数据
3. 如果成功，执行putKey操作
4. 通过RAFT协议执行putKey，元数据复制到state machine中
5. 当RAFT状态机调用新提交key的metadata,Container查询data block。如果data block没有找到或者checksum和复制过来的chunck的checksum不同。Container Client写入key也是可能。

**Ozone Key Read**

客户端从container中read key会返回chunk引用列表。使用这些chunks，可以从实际的chunk文件中读取数据。RAFT的改进是客户端从RAFT leader中获取key-chunks映射。

**Ozone Key Overrite**

在这个模型中声明chunks的不可变。一段时间内不支持

**Append to keys**

Append是override的特殊例子，在metadata链尾添加blocks。步骤如下：

1. OzoneKyes -> {{filename:block1,offset:0,length:100}}
2. 写入100bytes，然后写入another puKey，但是time length不同

OzoneKey -> {{filename:block1,offset:0,length:200}}

1. 将100bytes写入chunk，通过putKey update

OzoneKye-> {{filename:block1,offset:0,length:300}}

PutKey的频度由写数据来决定。

**Failures and recovery under Simplilied Pipeline Model**

在Simplified Pipeline Chunks只有两种状态，存在或者不存在，只有在完全写成功后，数据才不可变且可见。Completely formed的定义在元数据中的key定义。在经典的pipeline中，block在不同的状态转变，包括RBW(Replica Being Writern),Finalized Replica,Replicat waiting to be Recovered,Replicat Under Recovery等。比较难处理的是错误恢复。

**Recovery From Disk Failure**

Datanode中有检测disk失败，需要考虑两种情况：

1）Disk Loss造成LevelDB运行失败，通过RAFT replicated

2）丢失data chunks

第一种情况下，会丢失container的元数据，可以作为complete replica loss。第二种情况是丢失数据块，但是没有元数据丢失，可以从其他节点复制数据，但是磁盘满的问题，很难解决。Data node向SCM汇报副本的丢失，SCM会生成新的副本，所有的container data会重建。

**Recovery From Node failure**

两种不同的机制实现节点恢复

1. 通过RAFT Leader检测机制发现一个Container RAFT Leader loss，重新选择leader
2. SCM检测到几个主机丢失，datanode通过心跳与SCM交互，SCM会通知RAFT Leader，leader重新获取一个新主机

**Over Replication**

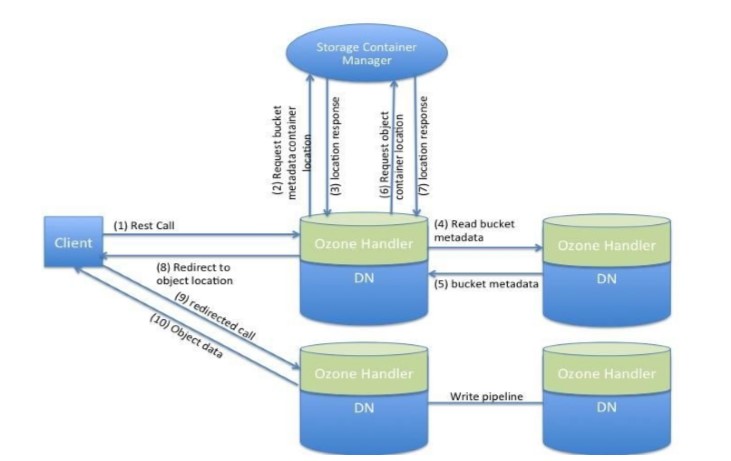
Ozone不支持Over-replication

**Health of a Container**

使用Raft Protocol, container获取本身的health情况，当出现node loss，container会自动恢复。Container汇报的信息仅是让SCM知道containers中有多少数据。对于订阅service，可以发送key reports。

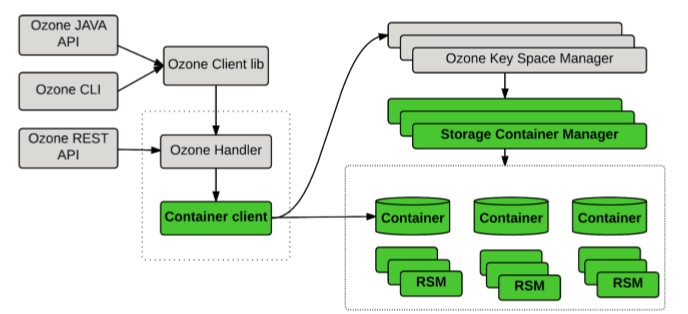
**Metadata Management**

当KSM和SCM分开，下面对比原框架和新框架的区别：



原推荐框架

新框架如下图所示：



上面的两个图不同，不同点是多了KSM服务模块。还有一个是RSM(replicated State Machine),