DANA GateWay设计方案

## 目录

[DANA GateWay设计方案 1](#_Toc440536858)

[1. 需求 3](#_Toc440536859)

[1.1. 需求概述 3](#_Toc440536860)

[1.2. 需求分析 3](#_Toc440536861)

[1.3. 项目目的 3](#_Toc440536862)

[2. 设计方案 4](#_Toc440536863)

[2.1. DANA GateWay整体架构 4](#_Toc440536864)

[2.2. DANA GateWay模块设计方案 5](#_Toc440536865)

[2.2.1. Object common 5](#_Toc440536866)

[2.2.2. Object管理模块 10](#_Toc440536867)

[2.2.3. User和Bucket管理 13](#_Toc440536868)

[2.2.4. Acl权限管理 15](#_Toc440536869)

## 需求

### 需求概述

需求明确要求必须放弃文件系统中的元数据和中心节点的数据组织方式，但是也需要同时具备某些文件系统的特性，即不能保留元数据服务器又需支持文件随机访问、随机读写和随机修改，从而避免元数据节点故障导致业务数据丢失的严重事故。

### 需求分析

通过需求分析，确定使用对象存储来满足系统的数据存储需求，需求要点如下：

* 支持s3/restful接口
* 支持随机覆盖写
* 支持媒体文件边写边读
* 支持对象扁平化管理
* 支持并发高性能读写

原有存储系统已支持基本对象的读写，创建删除等基本操作，但没有可以对外提供restful/s3接口访问，并支持对象随机覆盖写、随机读的对象存储网关。

### 项目目的

为满足非编业务的具体业务场景，需要一种无元数据，支持s3/restful接口，支持随机读，支持随机覆盖写和支持边写边读的对象存储网关。

## 设计方案

### DANA GateWay整体架构



图 1 系统整体概图

DANA GateWay在系统中用于沟通用户客户端和存储集群，它负责接收处理来自客户端的restful/s3请求，转化为存储识别的对象访问请求，最后返回给客户端存储集群的数据。



图 2 DANA GateWay整体架构

DANA GateWay内部结构如图2，主要模块包括s3/restful模块，协议解析模块，业务service模块，user管理模块，bucket管理模块，object管理模块，acl管理模块和object Common模块。

* S3/restful模块：负责封装对外的s3/restful的接口，并接收客户端的s3/restful请求，转发下层模块处理。
* 协议解析模块：负责解析接收报文中的语义，转化为相关参数传递给业务模块。
* 业务service模块：负责执行具体业务操作，统一数据的访问接口。
* user管理模块：负责用户数据的管理，包含用户创建，查询和删除。
* bucket管理模块：负责bucket数据管理，包含bucket创建，遍历和删除。
* object管理模块：负责用户上传object管理，包含object访问，创建，删除，修改和元数据的管理。
* acl管理模块：负责用户权限管理，包含对指定用户授予bucket读和修改权限，授予object读和修改权限，共享bucket和object对所有用户可见。
* object common管理模块：负责对象底层io操作封装，包含对象分片，对象基本的读，写和删除，对象的属性管理。

网关内部处理一般流程为（以创建对象为例）：用户请求通过网络到达网关后，首先是被s3/restful模块接收，然后传递给协议解析模块，经过解析得到具体的操作对象，然后调用业务service中的创建object方法，再执行object模块中具体创建方法，通过object common模块将数据发送给存储集群。

### DANA GateWay模块设计方案

### Object管理模块

#### 大文件分片上传

场景：为支持客户端大文件分段上传，可以并发上传各个分段，提高上传大文件整体性能。



图 9 分片上传过程

* 1. 客户端先调用init muti part upload接口，验证bucket是否合法，生成upload id，创建元数据对象，记录上传的分片大小（由客户端确定大小，默认是5M）。
  2. 客户端根据分片大小把待上传的对象拆分成小片，然后调用upload object part，先验证upload id是否有效，再根据分片编号，用户名称和bucket名称生成分片对象名称，最后上传分片文件。
  3. 客户端把各个分片上传完毕后，可以调用list upload part查看当前已经上传了哪些分片，如果有分片没完成上传，可以再上传一次，最后调用complete upload完成上传过程，并更新bucket列表。
  4. 客户端可以终止对象上传，调用abort part upload，释放已经上传的对象文件，并且释放upload id。



图 10 分片上传后对象在存储内部的分布

1. Init upload执行后在存储中生成了bucket\_a.txt文件，按照普通对象分片规则分布。
2. Upload object part完成在存储中生成shadow\_bucket\_a.txt\_0、shadow\_bucket\_a.txt\_1和shadow\_bucket\_a.txt\_2，按照普通对象分片规则分布。

#### 对象上传共享读

场景：媒体文件刚开始上传，其他非编客户端就可以读取播放已经上传到存储的文件数据。

约束：Cache只能支持单实例网关的场景。



图 11 a.mkv上传共享读过程



图 12 共享读存储内部对象分布

共享读流程：

* 1. 写操作到达后，先对这个写请求做有效性验证，然后获得对象写锁后，开始写对象。
  2. 更新对象所在bucket的列表，使该对象可以被列举出来。这时读请求可以看到正在创建的对象。
  3. 写操作会先创建目标对象的首对象，然后再创建分片数据对象。构造的对象首先会写到cache中，读请求访问该对象，会返回cache中已经写入的数据，并且把在cache中的对象引用计数加一，读分片完成后则计数减一，如果cache中没查找到则直接从存储中获得。读请求也可能会返回错误，因为当前访问的分片并未在cache或存储中，这个分片在正在写cache。
  4. 写完cache后就直接返回，cache数据异步刷入存储中。数据同步服务会周期性，刷入分片数据到存储。刷入前会检查该分片对象的引用计数，如果不为0则不操作，如果为0则写入到存储中，并且把该分片对象从cache中删去。后来的读请求直接访问存储中数据。



图 13 cache数据结构

### User和Bucket管理

为支持s3协议，具备对象存储特性，需要有用户和bucket的概念。

#### 创建用户

场景：用户是对象存储的执行者，任何对象操作都是在用户之下的，需要先创建用户，才能使用存储。



图 14 创建用户Tester

创建用户就是在存储中创建一个对象，对象名称就是用户名称。对象内容存放的bucket列表，这个用户下创建的bucket。

#### 创建bucket

场景：bucket为用户下存放对象的一个命名空间，类似于目录。要使用对象，必须先要找到bucket。



图 15 创建bucket1

创建bucket是在存储中创建一个对象，bucket名称为用户名+bucket组合。bucket内容存放的是object列表。



图 16 用户bucket对象在存储内分布

#### 删除用户

描述：删除用户先需要检查用户下的bucket是否为空，不为空则不能删除。删除则是将存储中的用户对象删除。

#### 删除bucket

描述：删除bucket前需要检查bucket内是否为空，不为空则不能删除。删除bucket后，需要更新用户bucket列表。

### Acl权限管理

场景：需要指定某个对象或bucket共享给所有用户或者只是某个指定用户。

描述：acl权限列表存放到一个对象中，用于保存和维护用户的权限列表。acl权限列表是一个hash表。hash表定义为：<user\_name, {orig\_user, bucket\_name, object\_list}>。user\_name为被赋予权限的用户名，orig\_user为赋予bucket或object的创建者名称，bucket\_name为共享的bucket名称，object\_list为共享的对象列表，可以为空。



图 17 hash表数据结构

#### bucket共享给指定用户

场景：用户tester拥有bucket B1，用户tester2拥有bucket B2，将B2共享给tester。



图 18 B2共享给tester

1. 共享完成后，向.acl\_grant\_list中添加一组记录<Tester, {Tester2,B2,\*}>，用户Tester可以列举出B2并且可以访问B2中所有的object。
2. Tester访问B2中的对象，首先读出acl列表中Tester对应的内容，找到对象的创建者名称，所在bucket名称以及object对象名称。如果是object列表为空，则表示该bucket中的所有object都可以访问，先列举bucket中的所有object，根据选择的object，加上bucket和user名称，构造对象名称后再访问对象本身。
3. 删除bucket B2后，需要更新Tester的acl列表。

#### object共享给指定用户

场景：用户tester拥有bucket B1，用户tester2拥有bucket B2，将B2中的b.txt共享给tester。



图 19 b.txt共享给Tester

1. 共享完成后，向.acl\_grant\_list中添加一组记录<Tester, {Tester2,B2,b.txt}>，用户Tester可以列举出B2中的b.txt并且可以访问。
2. Tester访问B2中的b.txt，首先读出acl列表中Tester对应的内容，找到对象的创建者名称，所在bucket名称以及object对象名称。构造好对象名称后再访问对象本身。
3. 删除bucket B2中的b.txt后，需要更新Tester的acl列表。

#### bucket共享给所有用户

场景：将指定的bucket共享给所有用户，每个用户都可以该bucket中的所有对象。

描述：将Tester2的bucket B2共享给全部用户。



图 20 共享B2给所有用户

1. 共享完成后，向.acl\_grant\_list中添加一组记录<public, {Tester2,B2,\* }>，其他用户可以列举出B2中的所有对象并且可以访问。
2. 其他用户访问B2中的对象，首先读出acl列表中public对应的内容，找到对象的创建者名称，所在bucket名称以及object对象名称。如果是object列表为空，则表示该bucket中的所有object都可以访问，先列举bucket中的所有object，根据选择的object，加上bucket和user名称，构造对象名称后再访问对象本身。
3. 删除bucket B2后，需要更新public的acl列表。

#### object共享给所有用户

场景：将指定的object共享给所有用户，每个用户都可以该object对象。

描述：将Tester2的bucket B2中的b.txt共享给全部用户。



图 21 b.txt共享给所有用户

1. 共享完成后，向.acl\_grant\_list中添加一组记录<public, {Tester2,B2,b.txt}>，所有用户可以列举出B2中的b.txt并且可以访问。
2. 其他用户访问B2中的b.txt对象，首先读出acl列表中public对应的内容，找到对象的创建者名称，所在bucket名称以及object对象名称。构造好对象名称后再访问对象本身。
3. 删除bucket B2中的b.txt后，需要更新public的acl列表。

### Object common

#### 普通对象分片

场景：上传文件大小大于存储管理最小chunk大小，则需要分片，反之则不分片。

分片原因：最大化利用存储资源，提高并发访问效率。

**不分片场景：**

1. 上传过程：
2. 比较文件大小和chunk大小。
3. 判断不需分片，在对象附加属性中设置文件本身大小、分片标识和bucket。
4. 直接将当前文件写入存储，本身作为meta对象。
5. 下载过程：
6. 获得meta对象。
7. 检查分片标识。
8. 没分片则返回当前对象（meta对象即为数据对象本身）。



图 3 不分片场景

**分片场景：**

1. 上传过程：
2. 比较文件大小和chunk大小。
3. 根据文件大小及chunk大小算出分片个数，结果向上取整。
4. 生成meta对象，对象名称为文件名本身，文件内容为空，在对象附加属性中设置文件本身大小、分片标识和bucket。
5. 分片文件名称规则为mutipart+文件名称+分片编号（从0开始），得到分片文件名后，根据chunk大小分别写入到存储中。
6. 下载过程：
7. 获得meta对象。
8. 检查分片标识。
9. 分片后，则读取meta对象中文件大小属性。
10. 通过文件大小计算出分片个数，则得到分片文件名称。
11. 通过名称从存储中获得每个分片文件。
12. 最后串行将各个小文件按顺序返回给请求方。



图 4 分片场景

#### 对象随机覆盖写

场景：文件本身，某些数据可以被覆盖修改。

**不分片场景：**

1. 获得meta对象。
2. 检查分片标识。
3. 根据写内容长度判断是否需要跨分片。
4. 根据偏移，查找属于第几个分片。
5. 确定分片后，把传入的数据，根据长度覆盖写对象本身。



图 5 不分片覆盖写

**分片场景：**

1. 获得meta对象。
2. 检查分片标识。
3. 判断起始偏移属于哪个分片。
4. 通过待覆盖长度判断写入是在分片内完成还是需要跨分片。
5. 根据文件偏移算出需要跨分片数量。
6. 算出片内偏移再将待覆盖的数据分片分别写入到各个分片中。
7. 完成覆盖后再返回。



图 6 分片覆盖写

#### 对象随机读

场景：支持访问对象任意位置，并且读出指定起始位置之后的数据。

**不分片场景：**

1. 获得meta对象。
2. 检查分片标识。
3. 根据写内容长度判断是否需要跨分片。
4. 根据偏移，查找属于第几个分片。
5. 确定具体分片后，根据读取长度返回读取数据。
6. 读取完毕后返回。



图 7 不分片随机读

**分片场景：**

1. 获得meta对象。
2. 检查分片标识。
3. 判断起始偏移属于哪个分片。
4. 通过待覆盖长度判断写入是在分片内完成还是需要跨分片。
5. 根据文件偏移算出需要跨分片数量。
6. 算出片内偏移，再根据读取数据长度算出，当前分片中需要读取的数据长度，读出当前分片的数据。
7. 完成各个分片数据读取后再返回。



图 8 分片随机读