基于Golang语言实现的云储存服务

基于Golang语言实现的云储存服务
- 简介
- 设计概述
- 实现过程
  - 云服务用户API
    - RESTful风格的http接口
    - 命令行接口工具
  - 接口服务器
  - 数据服务器
  - 元数据服务器
  - 消息队列机制
  - 部署和测试
- 一致性、可用性以及可靠性性分析
- 不足与进一步
- 总结
- 参考文献

简介

我实现了一个基于Golang语言实现的云存储服务，是一个使用对象存储的云存储服务，以对象的方式来管理数据，一个对象包含三部分：对象的数据、对象的元数据以及一个全局唯一的标识符（即对象的ID）。这个云存储服务具有可伸缩、版本控制、数据校验、数据去重、数据冗余、数据修复、断点续传、数据压缩、数据维护的功能。

该云存储服务使用RESTful风格的API接口，同时提供了一个简单易用的交互命令行工具。在本文档中，我将描述本云存储服务的设计、接口以及进行基本的功能展示。

本云存储服务的设计目标是：

1. 性能，包括存储空间利用效率、数据传输效率等等。
2. 用户体验。
3. 可伸缩性。
4. 一致性。
5. 可靠性。
6. 可用性。

该云存储服务的有一些基本设计如下：

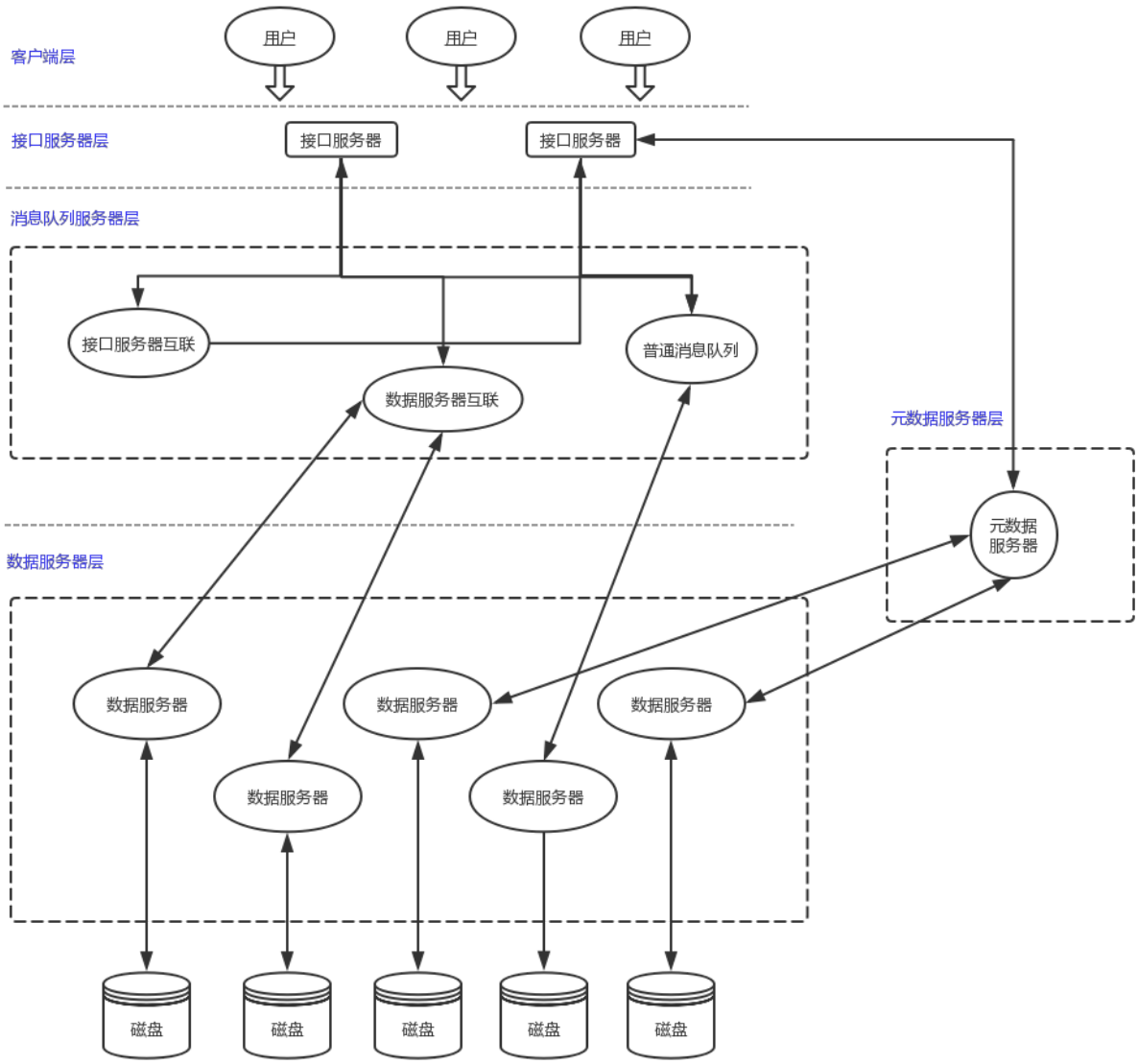
首先，该云存储服务的服务对象是个人用户，假定个人用户的文件都不大，这样以对象的方式来存储数据才有意义。

其次，该云存储服务只考虑了服务器的存储出错，而不考虑服务器的运行出错，即假设服务器不宕机。

再次，给云存储服务的一致性保证通过开源搜索框架ElasticSearch来保证，设计为强一致性。

设计概述

在设计该云存储服务时，我将其作为一个分布式对象存储系统来设计，整体设计如下图：



下面分别简述这些层次的设计：

- 客户端层：用户通过RESTful风格的http接口访问接口服务器，或者通过设计好的接口命令行工具访问。
- 接口服务器层：负责与数据服务器以及元数据服务器交互，也是各种做出各种决策的位置。
- 消息队列服务器层：使用RabbitMQ组件来处理服务器之间的通信。
- 元数据服务器层：使用开源搜索框架ElasticSearch来处理对象的元数据，该系统的一致性和版本控制也由元数据服务器来保证。
- 数据服务器层：对象的数据存储在数据服务器层，同时该层进行数据的修复、压缩、维护等。

实现过程

云服务用户API

用户的接口功能有上传、下载、版本查询、文件删除、快速搜索等。

RESTful风格的http接口

- 上传

```
curl -XPUT API_SERVER_IP:API_SERVER_PORT/objects/OBJECT_NAME -T OBJECT_NAME -H "Digest: SHA-256=SHA256_BASE64_HASH"
```

- 下载

```
curl -XGET API_SERVER_IP:API_SERVER_PORT/objects/OBJECT_NAME -o TARGET_NAME
```

- 版本查询

```
curl API_SERVER_IP:API_SERVER_PORT/objects/OBJECT_NAME/versions/OBJECT_NAME
```

- 文件删除

```
curl -XDELETE API_SERVER_IP:API_SERVER_PORT/objects/OBJECT_NAME
```

- 快速搜索

```
curl API_SERVER_IP:API_SERVER_PORT/objects/OBJECT_NAME/lookup/
```

命令行接口工具

使用一个开源的Go语言命令行交互工具Survey实现了一个命令行接口工具，提供的功能与上述方式一样，还提供了文件夹压缩上传功能，以提高上传效率。详细情况见测试部分。

接口服务器

接口服务器除了直接和用户交互，还要在与数据服务器以及元数据服务器交互的过程中承担更多的责任：

1. 数据校验：验证客户端提供的散列值和接口服务器根据对象数据计算出来的对象散列值是否一致。同时因为数据以流的形式传输，需要在数据服务器上建立一个临时对象，当数据传输完之后，散列值也计算完毕，此时再决定是否将临时对象转成正式对象。
2. 数据去重：当用户上传一个对象时，先在用该对象的散列值在数据服务器上进行定位，确定是否已经存在这样一个对象，若存在，则直接生成该对象的新版本元数据；否则按正常流程上传。
3. 数据冗余中的分片与恢复：为了减少数据服务器的存储出现错误带来的影响，本设计中使用Reed Solomon纠删码，同时选择4个数据片和两个校验片。也就是说会把一个完整的对象平均分成 6 个分片对象，其中包括 4 个数据片对象，每个对象的大小是原始对象大小的 25%，另外还有两个校验片，其大小和数据片一样。这 6 个分片对象被接口服务存储在 6 个不同的数据服务节点上，只需要其中任意 4 个就可以恢复出完整的对象。

   对象PUT流程：

   - 用户调用上传数据接口。
   - 使用RS编码将数据分为4+2个分片。
   - 将该6个分片对象分别上传到6个数据服务器。

   对象GET流程：

    - 用户调用下载数据接口。
    - 将6个分片（可能有缺失）下载到接口服务器。
    - 若6个分片都完整，则将0~3的分片组合成对象；否则若剩余大于等于4个分片，使用RS纠删码的中间矩阵来恢复数据；否则返回错误。
    - 返回该对象。
4. 断点续传：
    - 断点下载：对象客户端在请求时设置Range头部来告诉接口服务器从什么位置开始输出。
    - 断点上传：接口服务器返回一个标识数据，该数据包含当前已上传数据量等信息。用户端根据该标志数据来继续上传数据。

数据服务器

数据服务器除了承担写入和读取数据的基本功能之外，还要承担数据压缩、数据维护功能。

1. 数据压缩：本设计中使用gzip压缩和解压对象。
2. 数据维护：
    - 对象版本预留：使用数量限定策略，保留每个对象最后的5个版本。
    - 删除无元数据引用的数据，但是这样可能产生竞争，即在删除时有用户上传相同散列值的对象。于是删除对象时将其移进另一个存储空间，一段时间后再去真正删除，若在这段时间内有用户上传了相同散列值的对象，再将其重新放入对象存储空间。
3. 数据维护：定期检查数据是否发生顺坏，并即时进行修复，减少数据不可恢复（损失大于三个数据分片）。

元数据服务器

元数据指的是对象的描述信息，比如对象的名字、版本、大小以及散列值（本设计的散列函数使用SHA-256，并使用标准的Base64编码，即HTTP协议中采用的的Digest头部值）。

本设计中使用的是开源搜索框架ElasticSearch来维护元数据。Elasticsearch 是一个分布式、可扩展、实时的搜索与数据分析引擎。它是一个提供了基于RESTful的web接口，能够达到实时，稳定，可靠，快速的搜索引擎，每个字段都能被索引并可被搜索。

元数据服务器还在本设计中承担版本控制、强一致性保证的任务。

元数据服务器流程如下：

- 对象PUT流程：
    1. 用户调用上传文件接口。
    2. 根据用户上传的文件名来寻找最新的元数据。
    3. 添加新版本的元数据。
    4. 返回到接口服务。
- 对象GET流程：
    1. 用户调用下载文件接口。
    2. 从元数据中获取该对象最新版本的哈希值。
    3. 返回到接口服务。

通过ElasticSearch自带的一致性访问机制，所有用户访问到的元数据都是最新版本，所以所有用户访问到的都是最新版本的对象。

消息队列机制

数据服务器需要将自身的存在发送给所有的接口服务器，这就个广播过程通过消息队列机制完成：每个接口服务节点在启动后都会创建自己的消息队列并绑定至消息队列服务器，每个数据服务节点在启动后每隔 5s 就会发送一条消息给接口服务器。每个数据服务节点在启动时都必须创建自己的消息队列并绑定至消息队列服务器 。当接口服务需要定位时，会创建一个临时消息队列，然后发送一条消息给原先绑定好的消息队列 ，定位成功的数据服务节点需要将反馈消息发送给这个临时队列，临时消息队列会在一定时间后关闭。如果在关闭前没有收到任何反馈则该对象定位失败，接口服务节点就会知道该对象不存在于数据服务器层。

部署和测试

为了设计报告的简洁性，只展示部分重要功能的测试结果。

一致性、可用性以及可靠性性分析

不足与进一步

总结

参考文献

- Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." (2003).
- Tanenbaum, Andrew S., and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.
- Erl, Thomas, Ricardo Puttini, and Zaigham Mahmood. *Cloud computing: concepts, technology & architecture*. Pearson Education, 2013.
- 胡士杰. "分布式对象存储."(2018).
- Everything的原理猜想与实现. https://github.com/Artwalk/Fake-Everything.
- 一个Golang语言实现的简单文件系统. https://github.com/sjqzhang/go-fastdfs.
- 一个Golang语言实现的命令行交互框架. https://github.com/AlecAivazis/survey