2020 级

# 《物联网数据存储与管理》课程

# 实 验 报 告

姓　名　柳子淇

学　号　U202015628

班　号　本硕博 2001 班

日　期　2023.05.29

# 目　　录

# 一、实验目的

1. 熟悉对象存储技术，代表性系统及其特性；
2. 实践对象存储系统，部署实验环境，进行初步测试；
3. 基于对象存储系统，分析性能问题，架设应用实践。

# 二、实验背景

随着互联网行业的发展，数据的规模呈指数级增长，内容结构日趋复杂，例如电商行业，社交平台，短视频平台等等，这其中产生的数据大部分是非结构化数据，即不符合具有行和列的传统关系数据库的数据。

对象存储系统是一种用于存储大量非结构化数据的分布式存储技术。在传统的文件系统和块存储系统中，数据以文件或块的形式存储在物理设备上，而对象存储则将数据作为对象进行管理。

对象存储最初是由亚马逊在 2006 年推出的 S3（Simple Storage Service）服务引入的，它旨在为企业提供一个可扩展、高可用性和安全的云存储解决方案。随着数据的快速增长和企业对数据的要求不断提高，对象存储系统在近年来得到了广泛的应用。

相比于传统的文件系统和块存储系统，对象存储系统具有更好的可伸缩性、容错性和灵活性。它可以轻松地扩展存储容量，支持多副本备份和异地多活部署，同时还可以通过元数据管理和自定义策略实现更为灵活的数据管理。由此，对象存储系统已经成为许多企业和组织存储海量数据的首选技术。

# 三、实验环境

表格 1 实验环境

| 硬件环境 | |
|---|---|
| 处理器 | I5-1135G7 @ 2.4GHz x8 |
| 内存 | 16G |
| 硬盘 | 512G SSD |
| 操作系统 | Ubuntu20.04 |
| Golang 版本 | 12.5 |
| 服务器端 | Minio，Swift |
| 客户端 | Minio Client，Boto |

# 四、实验内容

实验内容分为两大部分：在对象存储技术实践中，主要掌握对象存储系统服务器端和客户端的搭建方法；在对象存储性能分析中，主要掌握对象存储系统的性能指标与基准测试方法，分析环境参数、负载特征对性能指标的影响，并尝试使用对冲请求、关联请求方法应对尾延迟问题。

## 4.1 对象存储技术实践

1．在个人电脑上安装 MinIO 服务端，服务端，进行初步尝试。
2．在服务器上用 docker 搭建 Openstack Swift 对象存储系统服务器端
3．在个人电脑上安装 python-swiftclient，编写对象存储系统客户端。

## 4.2 对象存储性能分析

### 4.2.1MinIO 测试

在搭建好 MinIO 的服务端和客户端后，下载 S3benche，编写批量测试脚本，进行基本测试。

### 4.2.2 Swift 程序编写

借助 Swift API 编写基准测试 python 程序，该程序可以向对象存储系统服务器端发起 PUT（上传）、GET（下载）、DELETE（删除）请求，测量请求延迟情况，计算吞吐率等性能指标。将不同参数下的测试结果保存到 csv 文件中。

根据测试结果作图并分析性能，探究对象尺寸对对象存储系统性能的影响。

### 4.2.3 应对尾延迟

通过运行对冲请求程序，在保持其他各参数恒定的前提下，探究对冲请求策略中参数对缓解尾延迟效果的影响。

# 五、实验过程

**1．安装本地 MinIO 服务端和客户端**

下载并运行服务端

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
chmod +x minio
MINIO_ROOT_USER=admin MINIO_ROOT_PASSWORD=password ./minio server ./root
--console-address ":9001"
```

```
# lzq @ ubuntuqaq in ~/Desktop/pros [15:22:24] C:130
$ sudo MINIO_ROOT_USER=admin MINIO_ROOT_PASSWORD=password ./minio server /mnt/data -
-console-address ":9001"
[sudo] lzq 的密码:
MinIO Object Storage Server
Copyright: 2015-2023 MinIO, Inc.
License: GNU AGPLv3 <https://www.gnu.org/licenses/agpl-3.0.html>
Version: RELEASE.2023-04-20T17-56-55Z (go1.20.3 linux/amd64)

Status:         1 Online, 0 Offline.
S3-API: http://10.12.61.110:9000  http://127.0.0.1:9000
RootUser: admin
RootPass: password

Console: http://10.12.61.110:9001 http://127.0.0.1:9001
RootUser: admin
RootPass: password

Command-line: https://min.io/docs/minio/linux/reference/minio-mc.html#quickstart
   $ mc alias set myminio http://10.12.61.110:9000 admin password

Documentation: https://min.io/docs/minio/linux/index.html
Warning: The standard parity is set to 0. This can lead to data loss.
```

图 2 MinIO 服务端启动

客户端

```
wget https://dl.min.io/client/mc/release/linux-amd64/mc
chmod +x mc
sudo ./mc alias set myminio/ http://10.12.50.206:9000 admin   password
```

```
# lzq @ ubuntuqaq in ~/Desktop/pros [15:27:24] C:1
$ ./mc alias set myminio http://10.12.61.110:9000 admin password
Added `myminio` successfully.
```

图 2 MinIO 客户端配置

浏览器访问 10.12.50.206:9001，输入账号 admin 密码 password，访问正常。

顺便创建一个 bucket：loadgen，用于后续的测试。

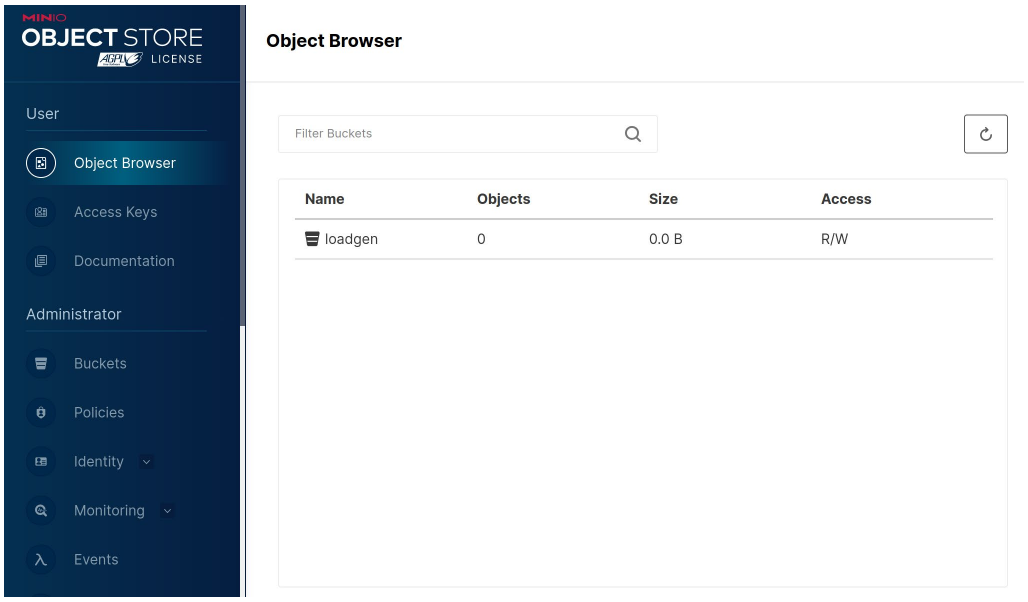



图 3MinIO 浏览器访问管理页面

### 2．S3bench 测试本地的 MinIO

下载标准测试程序 S3 Bench

需要:Go 环境  Go -version 1.20

```
go get -u github.com/igneous-systems/s3bench
```

调整了 S3bench 的输出（主要改动了格式，小数点保留位数，便于统计）重新编译。用 python 绘制散点图（myplot.py），但是特征并不是特别明显。

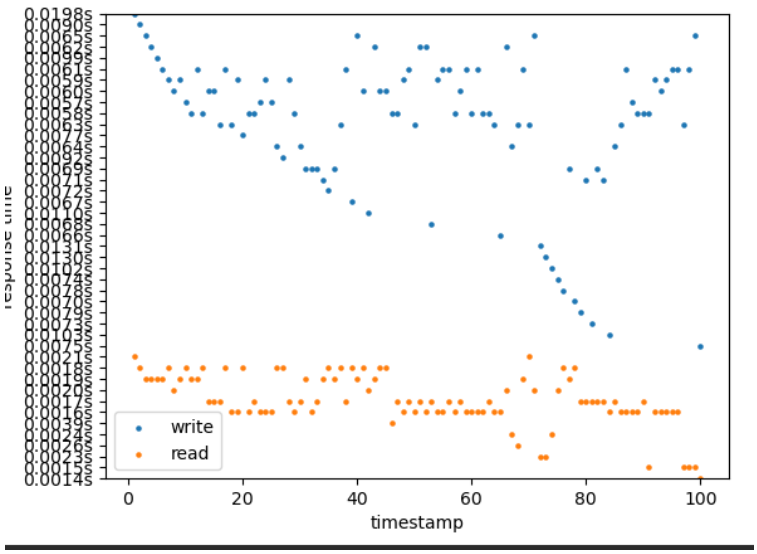
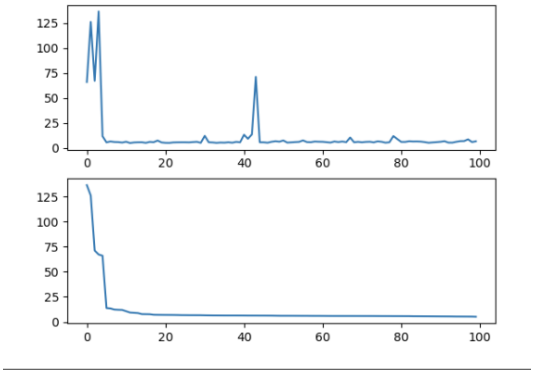


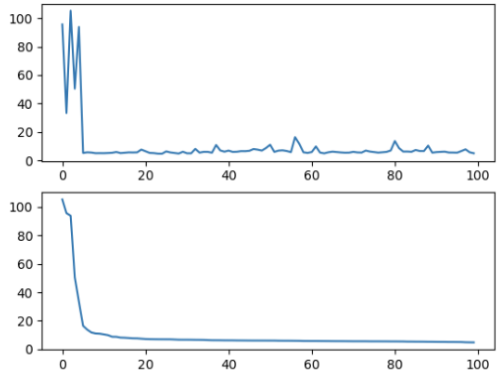

图 4 MinIO+S3benche 散点图（原数据未排序）

### 3．安装 pytho-boto3，用 BOTO 测试本地 MinIO

BOTO-本地 MinIO 测试脚本见  latenecy_test_MinIO.py。

选取部分测试图像（其他参数适当，改变单个元素大小）



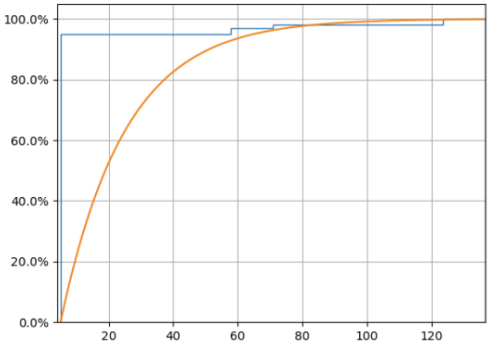




图 5 文件大小 4K

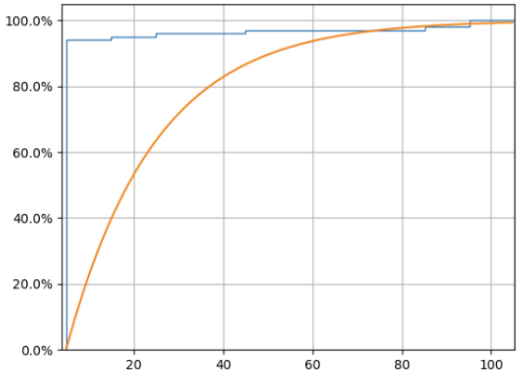



图 6 文件大小 16K

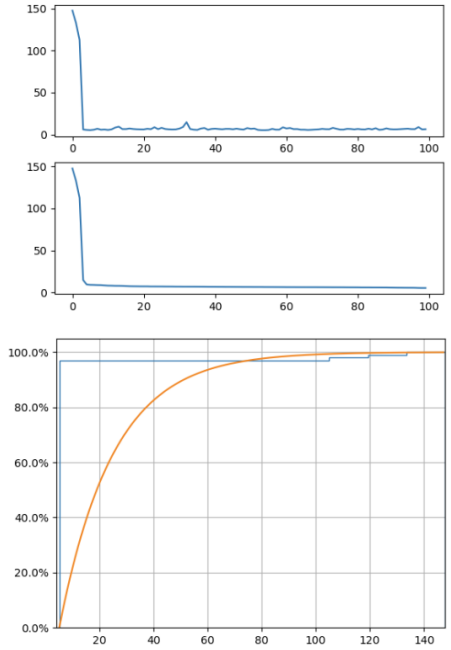



图 7 文件大小 32K

**4. 在远端搭建 Openstack Swift 容器版**

用 ssh 连接服务器 克隆 **Openstack Swift 的 docker 镜像并安装**。

```
liuziqi@liuziqi-MS-TZZ-H510M-K:~$ cd test
liuziqi@liuziqi-MS-TZZ-H510M-K:~/test$ ls
openstack-swift-docker
liuziqi@liuziqi-MS-TZZ-H510M-K:~/test$ cd  openstack-swift-docker/
liuziqi@liuziqi-MS-TZZ-H510M-K:~/test/openstack-swift-docker$ ls
Dockerfile  files  LICENSE  README.md
liuziqi@liuziqi-MS-TZZ-H510M-K:~/test/openstack-swift-docker$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED      STATUS      PORTS                                                  NAMES
924a1386cd80   openstack-swift-docker   "/bin/sh -c /usr/loc…"   2 days ago   Up 2 days   0.0.0.0:12345->8080/tcp, :::12345->8080/tcp   openstack-swift
liuziqi@liuziqi-MS-TZZ-H510M-K:~/test/openstack-swift-docker$
```

图 8 Swift 已在容器中运行

## 5 安装 python-swiftclient，编写客户端访问请求

(见 swift_client.py)

测试三种请求的响应情况，保存至 3 个 csv 文件，作图，图像就不一一列出了。

## 6 编写对冲请求应对尾延迟

进行测试，运行结果如下。

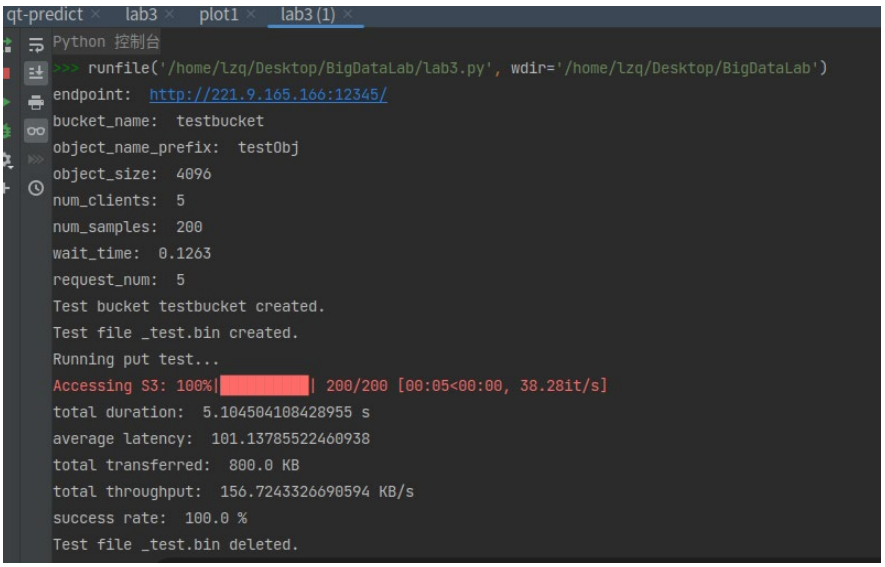



图 9 根据策略向服务器发送请求

经几次测试，参数设置如下：

```
bucket_name = 'testbucket'
object_name_prefix = 'testObj'
object_size = 4 * 1024  # 4KB
num_clients = 5
num_samples = 200
wait_time = 0.1263  # 相邻两次对冲请求发起的时间间隔，若 wait_time=0 即为关联请求
request_num = 3  # 对冲/关联请求总共发起的请求数
```

测得数据作图如下：

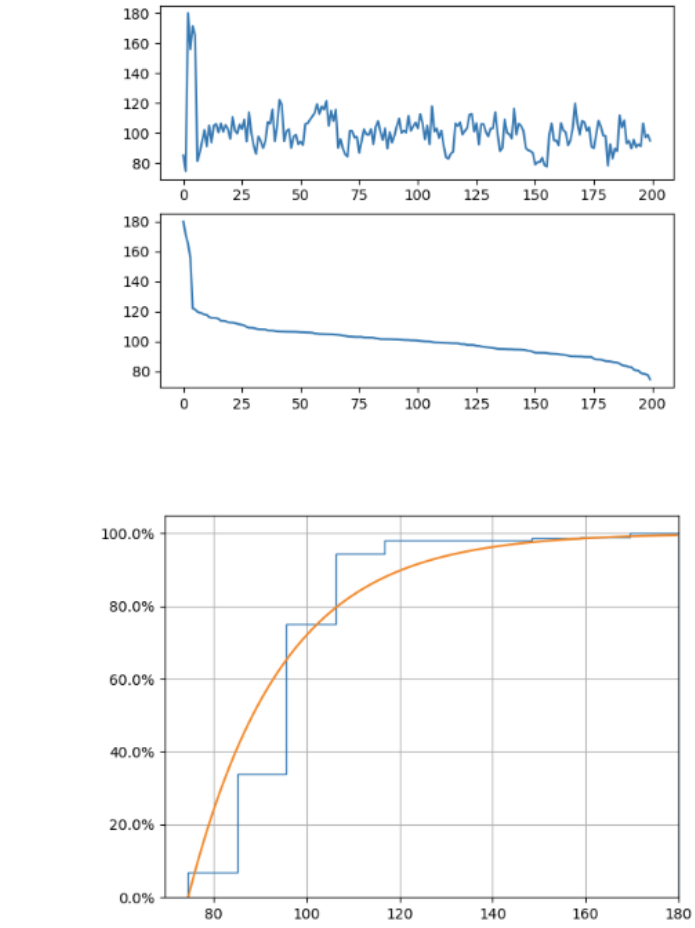



图 10 优化后(request_num=3)

将 request_num 增加至 5 结果如下：

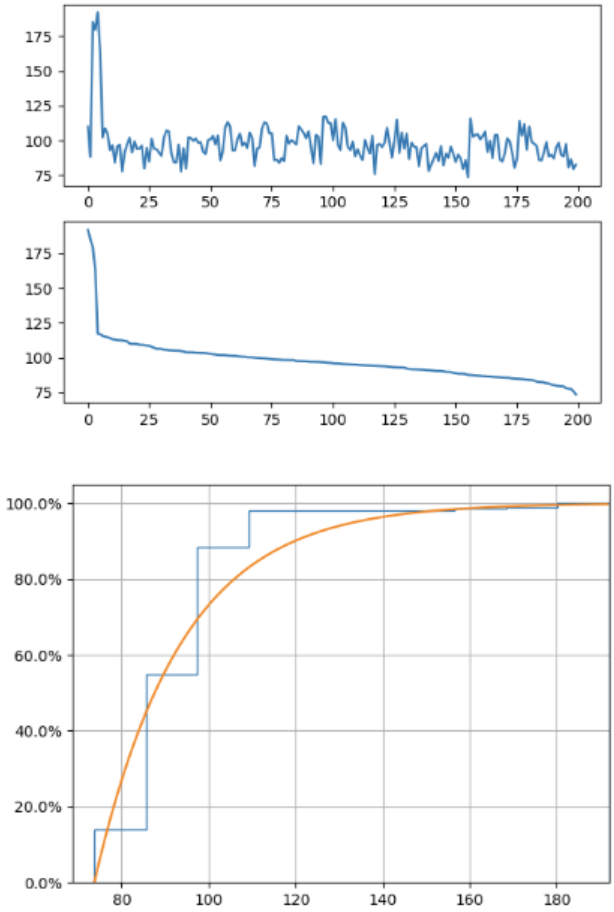



图 11 优化后(request_num=5)

可以观察到原始数据中，只有前几次请求时延迟会稍高，之后延迟稳定。尾延迟得到了一定程度的改善。下面是尾延迟百分位线对比图。

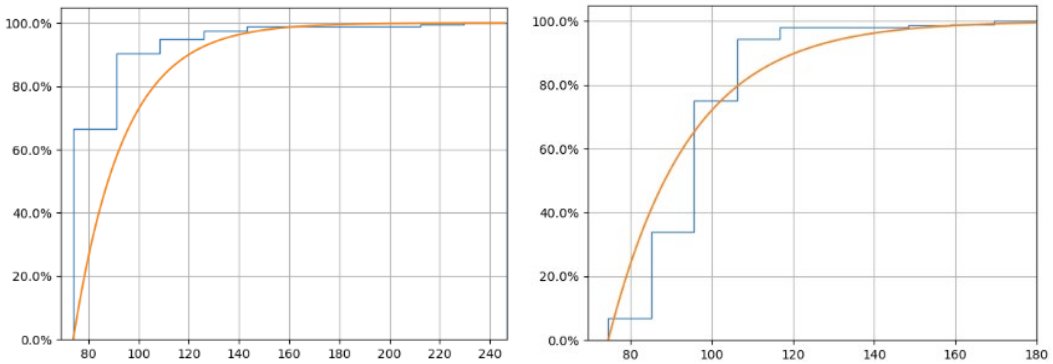



图 12 对比图

## 六、实验总结

通过实验一，我初步接触和了解了对象存储技术的特点，理解了对象存储系统的经典架构，实践了对象存储系统的服务器端和客户端的搭建方法，了解了

Python 后端，熟悉了 Docker 容器技术。

通过实验二，我在阅读 s3-bench 等基准测试程序源代码的过程中了解了基准测试程序的设计和编写方法，体验了 Swift API 的使用。我对对象存储系统的性能进行了观测，并通过设置不同参数对照实验的方式，探究了对象尺寸和并发数对对象存储系统性能的影响。

通过实验三，我在查阅文献理解对冲请求思想的过程中提高了文献阅读能力，在编程实现对冲请求的过程中进一步巩固了进程、线程等操作系统课程中所学的概念，对 Python 多线程有了初步了解，提升了编程能力。

本实验中的不足或可进一步探究的点：例如，无论在本地还是远程，我搭建的都是单机系统。可以进一步使用 docker 搭建 ceph 对象存储集群，从而进一步探究服务器规模对系统性能的影响。

实验过程中，施展老师和同学们解答了我的很多困惑，感谢老师和同学们的帮助!

## 参考文献

[1] ZHENG Q, CHEN H, WANG Y 等. COSBench: A Benchmark Tool for Cloud Object Storage Services[C]//2012 IEEE Fifth International Conference on Cloud Computing. 2012: 998－999.

[2] ARNOLD J. OpenStack Swift[M]. O'Reilly Media, 2014.

[3] WEIL S A, BRANDT S A, MILLER E L 等. Ceph: A Scalable, High-performance Distributed File System[C]//Proceedings of the 7th Symposium on Operating Systems Design and Implementation. Berkeley, CA, USA: USENIX Association, 2006: 307－320.

[4] Dean J, Barroso L A. Association for Computing Machinery, 2013. The Tail at Scale[J]. Commun. ACM, 2013, 56(2): 74－80.

[5] Delimitrou C, Kozyrakis C. Association for Computing Machinery, 2018. Amdahl's Law for Tail Latency[J]. Commun. ACM, 2018, 61(8): 65－72.