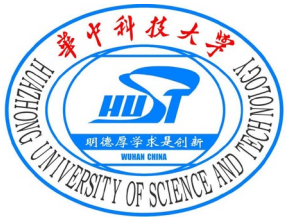


2019 级

《大数据存储系统与管理》课程

实 验 报 告

姓　名　张家荣

学　号　U201915084

班　号　ACM1901 班

日　期　2022.04.14

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

在万物互联的大数据时代的当下，数据的规模呈指数级扩张、内容结构日趋复杂，其中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，例如非结构化媒体数据、Web 内容、物联网设备产生的传感器数据等等。

对象存储技术在大数据时代已经成为存储、归档、备份和管理海量非结构化数据的首选方法。对象存储系统不是将文件分解为存储在文件系统中的磁盘上的块，而是将对象视为存储在结构扁平的数据环境中的离散数据单元。在对象存储系统中没有文件夹、目录等复杂的层次结构，每个对象都是一个简单的存储库，其中包括数据、元数据和唯一的标识 ID 号，这些信息使得应用程序能够定位和访问对象。对象存储系统中的对象通常存储在云服务器上，可以通过基于 HTTP 的 RESTful API 访问和操作。

与基于文件或基于块的传统存储技术相比，对象存储技术消除了具有文件夹和目录的分层文件系统带来的复杂性挑战，提高了存储系统在内容、规模和功能上的可扩展性。对象存储将存储位置抽象为 URL，以便能够以一种与底层存储机制无关的方式扩展规模，这使得对象存储成为构建大规模和高并发系统的理想方式。

对象存储系统由服务器端和客户端组成。对象存储服务器端、Amazon S3 协议、对象存储客户端三者之间的关系，类似于 HTTP 服务器、HTTP 协议、HTTP 客户端之间的关系。对象存储客户端向对象存储服务器端发出 GET、PUT、DELETE 等请求，对象存储服务器处理这些请求，把这些请求实现为对文件的操作，它们之间按照 Amazon S3 协议进行通信，Amazon S3 协议其实是 HTTP 协议的一个扩展。

根据本实验的指导教程，对象存储系统的服务器端可分为三类：第一类是面向初学者的 Minio，其带有图形化界面，操作简单；第二类是实验性模拟服务程序，即用 Python/Go/Java 等语言写的对象存储系统服务器端模拟器，如 mock-s3、s3proxy，通过

研读这些程序的代码，可以帮助我们理解对象存储系统的工作原理；第三类是企业级项目，如 Openstack Swift 和 Ceph，在实际生产中得到了广泛应用，上手门槛较高。

对象存储系统的客户端也可分为三类：第一类是面向初学者的 Minio Client，与 Minio 配套使用；第二类是已写好的命令行工具（CLI），如 osm、s3cmd、aws-shell，这一类客户端是功能固定的可执行程序，需要自己写 shell 脚本实现更复杂的批量测试操作；第三类是应用程序编程接口（API），如 awssdk、boto3，它们提供的是在程序中可调用的一系列函数，使用者可以在自己的程序中需要访问对象存储系统服务器端时调用这些函数，因此自由度高、更贴合实际生产需求。

为尽量贴合实际生产环境，本实验选用 Openstack Swift 搭建对象存储系统，并使用 Docker 简化部署。注意 Openstack Swift 的 API 与 Amazon S3 协议略有不同，因此只能使用它自己的客户端 python-swiftclient，该客户端同时提供了 CLI 和 API。

三．实验环境

为了保持实验系统环境的纯净性，我没有选择直接在 Windows 上搭建对象存储系统，而是在个人电脑上安装 Ubuntu 虚拟机，并在虚拟机中完成了对象存储系统服务器端、客户端搭建以及基准测试部分的实验。但当我探究对象存储系统的尾延迟时，却发现对冲请求和关联请求都不能改善系统的尾延迟，反而会显著增大请求的平均延迟。经询问老师得知，这是因为我的对象存储系统的服务器端负载较重，增大发送请求的数量反而会使服务器端负载进一步加重，每个请求的延迟都会显著变长。我尝试减少请求大小和并发数，但并没有得到明显改善。

考虑到我使用的是个人电脑上运行的虚拟机，还把客户端和服务器端在同一个机器上运行，性能自然稍弱。因此最终我决定把对象存储系统的服务器端搭建在性能更好的云服务器上，客户端直接搭建在个人电脑上，不再使用虚拟机，客户端和服务器端互不干扰，重新进行实验。改进配置后，系统的瓶颈将不再是网络带宽、硬盘 I/O 等，因此测量结果将可以更好地反映对象存储系统本身的性能。

对象存储系统服务器端的软硬件配置如表 1 所示，对象存储系统客户端的软硬件配置如表 2 所示。

表 1 对象存储系统服务器端软硬件环境

CPU	Intel Xeon(Ice Lake) Platinum 8358 2.6GHz|3.4GHz，8 核
内存	16GB
硬盘	企业型 SSD 本地盘，40GB，IOPS: 2000
公网带宽	50Mbps
操作系统	Ubuntu Server 18.04.6 LTS 64bit
对象存储服务器端	Openstack Swift (Docker 20.10.14 + docker-swift 2.27.0)

表 2 对象存储系统客户端软硬件环境

CPU	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz，6 核
内存	16 GB (8GB DDR4 2666MHz 内存×2，双通道)
硬盘	512GB 固态硬盘
操作系统	Microsoft Windows 10 家庭中文版 (Build: 18363.1556) (64 位)
对象存储客户端	python-swiftclient 3.13.1
集成开发环境	Spyder 5.0.5 (Python 3.8.10 64-bit)

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. 在云服务器上安装 docker，在 docker 中搭建 Openstack Swift 对象存储系统服务器端。
2. 在个人电脑上安装 python-swiftclient，搭建对象存储系统客户端。
3. 使用 Swift CLI 向对象存储系统服务器端发出简单的请求操作，以验证所搭建对象存储系统功能的正确性。

4.2 对象存储性能分析

4.2.1 程序编写

1. 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。该程序共有以下参数可以调整：对象尺寸 object_size、对象个数 num_samples、客户端个数（并发数）num_clients。
2. 借助 Swift API 编写对冲请求 python 程序，该程序可以采用对冲/关联/普通请求方式向对象存储系统服务器端发起 PUT（上传）请求，测量请求延迟情况。该程序共有以下参数可以调整：对象尺寸 object_size、对象个数 num_samples、客户端个数（并发数）num_clients、相邻两次对冲请求发起的时间间隔 wait_time、对冲请求总共发起的请求数 request_num。
3. 编写尾延迟分析 python 程序，该程序可以分析基准测试程序和尾延迟测试程序输出的 csv 结果文件，绘制尾延迟实际曲线和相应的排队论模型拟合曲线。

4.2.2 性能分析

1. 用默认参数运行基准测试程序，初步了解对象存储系统的性能和延迟分布情况，体会 PUT、GET、DELETE 三种类型请求之间的异同。
2. 通过运行基准测试程序，在保持 num_samples 和 object_size 的乘积以及其他各参数恒定的前提下，探究对象尺寸对对象存储系统性能的影响，比较“大量小尺寸对象”和“少量大尺寸对象”哪一个可获得更好的性能。
3. 通过运行基准测试程序，在保持其他各参数恒定的前提下，探究客户端请求并发数对对象存储系统性能的影响。
4. 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 request_num 对缓解尾延迟效果的影响。
5. 通过运行对冲请求程序，在保持其他各参数恒定的前提下，探究关联请求策略中 request_num 对缓解尾延迟效果的影响。
6. 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 wait_time 对缓解尾延迟效果的影响。

五．实验过程

5.1 搭建对象存储服务器端

以下步骤均通过 MobaXterm 软件远程连接到云服务器，在云服务器中进行操作。

安装 docker。

```
ubuntu@i-ocdvrty9:~$ sudo curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun
[sudo] password for ubuntu:
# Executing docker install script, commit: 93d2499759296ac1f9c510605fef85052a2c32be
+ sudo -E sh -c 'apt-get update -qq >/dev/null'
+ sudo -E sh -c 'DEBIAN_FRONTEND=noninteractive apt-get install -y -qq apt-transport-https c
+ sudo -E sh -c 'curl -fsSL "https://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg" | gpg --
gpg'
gpg: WARNING: unsafe ownership on homedir '/home/ubuntu/.gnupg'
+ sudo -E sh -c 'echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.
c stable" > /etc/apt/sources.list.d/docker.list'
+ sudo -E sh -c 'apt-get update -qq >/dev/null'
+ sudo -E sh -c 'DEBIAN_FRONTEND=noninteractive apt-get install -y -qq --no-install-recommen
+ version_gte 20.10
+ '[' -z '' ']'
+ return 0
+ sudo -E sh -c 'DEBIAN_FRONTEND=noninteractive apt-get install -y -qq docker-ce-rootless-ex
+ sudo -E sh -c 'docker version'
Client: Docker Engine - Community
 Version:           20.10.14
 API version:       1.41
 Go version:        go1.16.15
 Git commit:        a224086
 Built:             Thu Mar 24 01:47:57 2022
 OS/Arch:           linux/amd64
 Context:           default
 Experimental:      true

Server: Docker Engine - Community
 Engine:
  Version:          20.10.14
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.16.15
  Git commit:       87a90dc
  Built:            Thu Mar 24 01:45:46 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.5.11
  GitCommit:        3df54a852345ae127d1fa3092b95168e4a88e2f8
 runc:
  Version:          1.0.3
  GitCommit:        v1.0.3-0-gf46b6ba
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0

================================================================================

To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:

    dockerd-rootless-setuptool.sh install

Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.


To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/

WARNING: Access to the remote API on a privileged Docker daemon is equivalent
         to root access on the host. Refer to the 'Docker daemon attack surface'
         documentation for details: https://docs.docker.com/go/attack-surface/

================================================================================

ubuntu@i-ocdvrty9:~$
```

测试 docker 安装情况。

```
ubuntu@i-ocdvrty9:~$ sudo docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:bfea6278a0a267fad2634554f4f0c6f31981eea41c553fdf5a83e95a41d40c38
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

ubuntu@i-ocdvrty9:~$
```

获取 dockerswiftaio/docker-swift 镜像。由于老师在文档中提供的 openstack-swift-docker 已经多年未维护，在最新版本下运行会报错，所以我在 docker hub 上搜索其他镜像，找到了 dockerswiftaio/docker-swift，该镜像由 NVIDIA 开发和维护，能够使我们在 docker 上方便快捷地部署 Openstack Swift 对象存储系统。

```
ubuntu@i-ocdvrty9:~$ sudo docker pull dockerswiftaio/docker-swift
Using default tag: latest
latest: Pulling from dockerswiftaio/docker-swift
345e3491a907: Pull complete
57671312ef6f: Pull complete
5e9250ddb7d0: Pull complete
02c2e228ae7c: Pull complete
cedecb3b1e61: Pull complete
64a9b8d722f7: Pull complete
d5754380bc91: Pull complete
1184b40a2a8d: Pull complete
c299513b421a: Pull complete
84e213d4119c: Pull complete
98bf35fc5965: Pull complete
Digest: sha256:3b5b5b7c7bc5f4bff5c8d84d50226dd07ac955301b1500e386cc1714900b7074
Status: Downloaded newer image for dockerswiftaio/docker-swift:latest
docker.io/dockerswiftaio/docker-swift:latest
ubuntu@i-ocdvrty9:~$
```

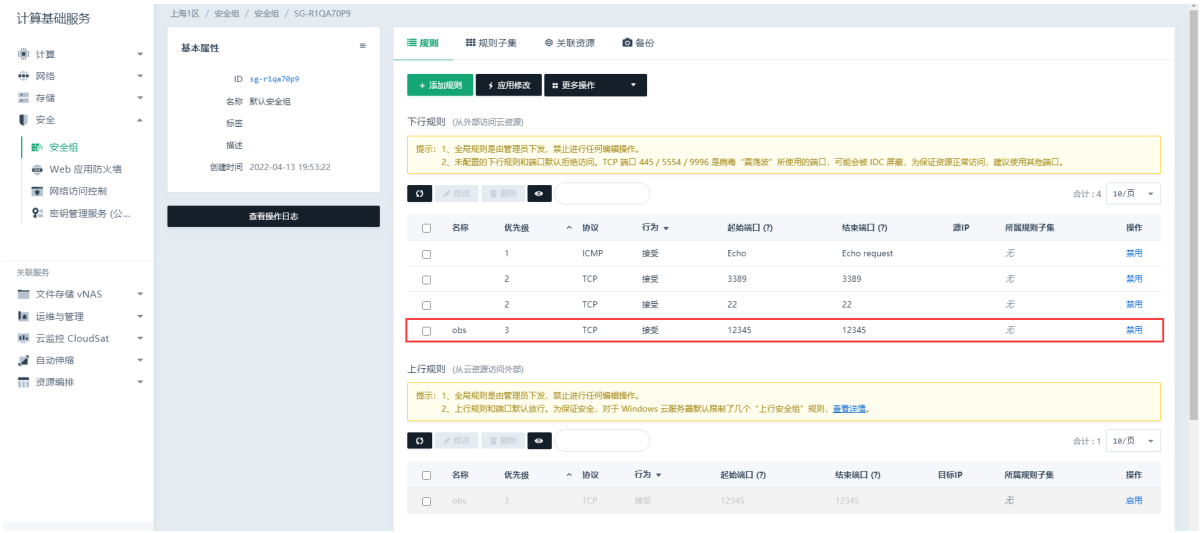
使用 dockerswiftaio/docker-swift 镜像启动一个容器。

```
ubuntu@i-ocdvrty9:~$ sudo docker run -d --name openstack-swift -p 12345:8080 dockerswiftaio/
docker-swift
3415b5181be56bd09e971864e2c80db94fcd996d11b87e8aaa8718fedd2fd9e2
ubuntu@i-ocdvrty9:~$
```

检查容器状态。确认容器正常运行，表明对象存储系统服务器端已搭建完成。

```
ubuntu@i-ocdvrty9:~$ sudo docker logs openstack-swift
 * Starting enhanced syslogd rsyslogd
   ...done.
 * Starting rsync daemon rsync
   ...done.
Starting memcached: memcached.
Device d0r1z1-127.0.0.1:6010R127.0.0.1:6010/sdb1_"" with 1.0 weight got id 0
Reassigned 1024 (100.00%) partitions. Balance is now 0.00.  Dispersion is now 0.00
Device d0r1z1-127.0.0.1:6011R127.0.0.1:6011/sdb1_"" with 1.0 weight got id 0
Reassigned 1024 (100.00%) partitions. Balance is now 0.00.  Dispersion is now 0.00
Device d0r1z1-127.0.0.1:6012R127.0.0.1:6012/sdb1_"" with 1.0 weight got id 0
Reassigned 1024 (100.00%) partitions. Balance is now 0.00.  Dispersion is now 0.00
Starting container-server...(/etc/swift/container-server/1.conf)
Starting proxy-server...(/etc/swift/proxy-server.conf)
Starting account-server...(/etc/swift/account-server/1.conf)
Starting object-server...(/etc/swift/object-server/1.conf)
Starting container-sharder...(/etc/swift/container-server/1.conf)
Starting object-auditor...(/etc/swift/object-server/1.conf)
Starting account-auditor...(/etc/swift/account-server/1.conf)
Starting container-replicator...(/etc/swift/container-server/1.conf)
Starting object-expirer...(/etc/swift/object-server/1.conf)
Starting object-replicator...(/etc/swift/object-server/1.conf)
Starting container-updater...(/etc/swift/container-server/1.conf)
Starting account-replicator...(/etc/swift/account-server/1.conf)
Starting container-sync...(/etc/swift/container-server/1.conf)
Starting object-reconstructor...(/etc/swift/object-server/1.conf)
Starting container-auditor...(/etc/swift/container-server/1.conf)
Starting container-reconciler...(/etc/swift/container-reconciler.conf)
Starting account-reaper...(/etc/swift/account-server/1.conf)
Starting object-updater...(/etc/swift/object-server/1.conf)
2022-04-13 12:38:12,501 INFO Set uid to user 0 succeeded
2022-04-13 12:38:12,503 INFO RPC interface 'supervisor' initialized
2022-04-13 12:38:12,503 INFO supervisord started with pid 189
ubuntu@i-ocdvrty9:~$ sudo docker ps -a
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS
               PORTS                                                   NAMES
3415b5181be5   dockerswiftaio/docker-swift    "/bin/bash /swift/bi…"   18 hours ago    Up 15 minu
tes            0.0.0.0:12345->8080/tcp, :::12345->8080/tcp             openstack-swift
bfe99d5802fc   hello-world                    "/hello"                 18 hours ago    Exited (0)
 18 hours ago                                                          elegant_torvalds
```

在云服务器控制台中编辑安全组规则，在下行规则中添加 12345 端口，以便从外部可以访问在云服务器的 12345 端口上运行的对象存储系统服务器端。



5.2 搭建对象存储客户端

以下操作均在个人 Windows 电脑上完成。安装 python-swiftclient，其中包含了命令行工具 Swift CLI 和供 python 编程使用的 Swift API。

```
(base) PS C:\WINDOWS\system32> pip install python-swiftclient
WARNING: Ignoring invalid distribution -oupsieve (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -arso (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -oupsieve (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -arso (c:\programdata\anaconda3\lib\site-packages)
Collecting python-swiftclient
  Downloading python_swiftclient-3.13.1-py2.py3-none-any.whl (87 kB)
     |████████████████████████████████| 87 kB 422 kB/s
Requirement already satisfied: six>=1.9.0 in c:\programdata\anaconda3\lib\site-packages (from py
thon-swiftclient) (1.16.0)
Requirement already satisfied: requests>=1.1.0 in c:\programdata\anaconda3\lib\site-packages (fr
om python-swiftclient) (2.26.0)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\programdata\anaconda3\lib\site-pa
ckages (from requests>=1.1.0->python-swiftclient) (2.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\programdata\anaconda3\lib\site-packag
es (from requests>=1.1.0->python-swiftclient) (1.26.7)
Requirement already satisfied: idna<4,>=2.5 in c:\programdata\anaconda3\lib\site-packages (from
requests>=1.1.0->python-swiftclient) (3.2)
Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anaconda3\lib\site-packages
(from requests>=1.1.0->python-swiftclient) (2021.10.8)
WARNING: Ignoring invalid distribution -oupsieve (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -arso (c:\programdata\anaconda3\lib\site-packages)
Installing collected packages: python-swiftclient
WARNING: Ignoring invalid distribution -oupsieve (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -arso (c:\programdata\anaconda3\lib\site-packages)
Successfully installed python-swiftclient-3.13.1
WARNING: Ignoring invalid distribution -oupsieve (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -arso (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -oupsieve (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -arso (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -oupsieve (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\programdata\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -arso (c:\programdata\anaconda3\lib\site-packages)
(base) PS C:\WINDOWS\system32>
```

使用 Swift CLI 操纵对象存储系统，以验证 Swift CLI 的功能和对象存储系统服务器端搭建的正确性。查看对象存储系统状态信息。

```
(base) PS C:\WINDOWS\system32> swift -A http://139.198.179.26:12345/auth/v1.0 -U test:tester -K
testing stat
                        Account: AUTH_test
                     Containers: 1
                        Objects: 400
                          Bytes: 4096
Containers in policy "policy-0": 1
   Objects in policy "policy-0": 400
     Bytes in policy "policy-0": 4096
                   Content-Type: text/plain; charset=utf-8
                    X-Timestamp: 1649853851.81251
                  Accept-Ranges: bytes
                           Vary: Accept
                     X-Trans-Id: txc8177593e7994665902cf-006257c4b8
         X-Openstack-Request-Id: txc8177593e7994665902cf-006257c4b8
(base) PS C:\WINDOWS\system32>
```

列出所有容器。

```
(base) PS C:\WINDOWS\system32> swift -A http://139.198.179.26:12345/auth/v1.0/ -U test:tester -K testing list
(base) PS C:\WINDOWS\system32>
```

创建新容器 TestContainer。

```
(base) PS C:\WINDOWS\system32> swift -A http://139.198.179.26:12345/auth/v1.0/ -U test:tester -K testing post TestContainer
(base) PS C:\WINDOWS\system32> swift -A http://139.198.179.26:12345/auth/v1.0/ -U test:tester -K testing list
TestContainer
(base) PS C:\WINDOWS\system32>
```

向指定容器 TestContainer 中上传对象，对象文件在本地的路径为 D:\bigdata-storage-

lab\light.png，对象在容器中命名为 test.png。

```
(base) PS C:\WINDOWS\system32> swift -A http://139.198.179.26:12345/auth/v1.0/ -U test:tester -K testing upload TestContainer
 D:\bigdata-storage-lab\light.png --object-name test.png
test.png
(base) PS C:\WINDOWS\system32>
```

列出指定容器 TestContainer 中的所有对象。

```
(base) PS C:\WINDOWS\system32> swift -A http://139.198.179.26:12345/auth/v1.0/ -U test:tester -K testing list TestContainer
test.png
(base) PS C:\WINDOWS\system32>
```

下载指定容器 TestContainer 中的指定对象 test.png。

```
(base) PS C:\WINDOWS\system32> swift -A http://139.198.179.26:12345/auth/v1.0/ -U test:tester -K testing download TestContain
er test.png
test.png [auth 0.079s, headers 0.129s, total 14.250s, 1.511 MB/s]
(base) PS C:\WINDOWS\system32>
```

删除指定容器 TestContainer 中的指定对象 test.png。

```
(base) PS C:\WINDOWS\system32> swift -A http://139.198.179.26:12345/auth/v1.0/ -U test:tester -K testing delete TestContainer
 test.png
test.png
(base) PS C:\WINDOWS\system32>
```

删除指定容器 TestContainer。

```
(base) PS C:\WINDOWS\system32> swift -A http://139.198.179.26:12345/auth/v1.0/ -U test:tester -K testing delete TestContainer

TestContainer
(base) PS C:\WINDOWS\system32>
```

以上实验的结果均符合预期，表明对象存储系统客户端和服务器端的功能无误，搭建正确。

5.3 对象存储性能分析

5.3.1 程序编写

参考 s3-bench、swift-bench 等已有对象存储系统基准测试程序，将欲编写的基准测试程序的流程设计如下：

1. 创建一个测试用容器，生成一个大小为 object_size 的负载文件。
2. num_clients 个客户端线程总共发起 num_samples 个 PUT 请求，每个请求都是上传在步骤 1 中生成的大小为 object_size 的负载文件。所有请求都收到响应后，打印总延迟、吞吐率、平均延迟等测试结果，将每个请求的详细延迟数据保存至 csv 文件中。
3. num_clients 个客户端线程总共发起 num_samples 个 GET 请求，每个请求是下载步骤 2 中上传的每个对象。所有请求都收到响应后，打印总延迟、吞吐率、

平均延迟等测试结果，将每个请求的详细延迟数据保存至 csv 文件中。

4. num_clients 个客户端线程总共发起 num_samples 个 DELETE 请求，每个请求是删除步骤 2 中上传的每个对象。所有请求都收到响应后，打印总延迟、吞吐率、平均延迟等测试结果，将每个请求的详细延迟数据保存至 csv 文件中。
5. 删除在步骤 1 中创建的测试用容器和负载文件。

“num_clients 个客户端线程总共发起 num_samples 个 PUT 请求”这一操作可用 ThreadPoolExecutor 模块实现，使用该模块维护一个大小为 num_clients 的线程池，向线程池中提交 num_samples 个任务，系统将自动调度线程池中的线程执行这些任务，并能在每个任务执行结束时执行处理返回结果等操作。这里，一个“任务”就是一个函数 bench_put，该函数将发起 1 个 PUT 请求、记录并返回延迟。基准测试程序的具体实现代码参见 lab2.py。

在对冲请求程序中，我们只考虑和实现 PUT 请求。对冲请求程序仍然是 num_clients 个客户端线程总共发起 num_samples 个 PUT 请求，但与基准测试程序的不同之处在于，一个请求可能需要发出多次请求。因此，在 bench_put 函数中，需要创建多个线程，每个线程负责发出一次请求。

实现对冲请求的难点在于，如何实现当任意一个线程发出的请求收到响应时，立即取消其它所有线程。为此我在网上查找了大量资料，尝试过多种方法。

第一种方法是：在线程请求函数的 PUT 操作的下一句执行结束其它所有线程的操作，但是 Python 中并未直接提供结束线程的功能，需要借助其他库和函数自行实现，且不安全。

第二种方法是：在 bench_put 函数中创建一个子进程，让该子进程创建多个线程执行发出请求的任务，在线程请求函数的 PUT 操作的下一句执行结束当前进程的操作，这样子进程将被结束，在子进程中创建的所有线程也随之被结束，在 bench_put 中可用 join 函数阻塞等待子进程结束，子进程从创建到结束之间经历的时间即为延迟时间。这种方法似乎可以完美实现对冲请求，但我按此方法实现之后测试的过程中，发现 PUT 请求时不时会报 499 Client Disconnect 错误，有时还会漏传文件，最终也没能解决这一问题，只好放弃，具体代码参见 lab3_method2.py，注意该代码使用 SIGTERM 信号结束进程，因此只能在 Linux 系统下运行才能实现正确功能。推测问题的原因可能是因为结束进程太过突然，PUT 操作可能当执行到一半时被打断，引起错误。

本实验中采用了第三种方法，即通过全局的标志变量 flag 来协调各线程。flag 初值为 True，第一个收到响应的线程请求函数将 flag 置为 False，其它尚未收到响应的线程请求函数在**执行完当前语句后**（不会中途被打断），会因 flag==False 而退出。因为没有粗暴的结束进程或线程的操作，所以这种方法不会引起错误。

尾延迟分析程序参考了 obs-tutorial 仓库中的 latency-plot.ipynb 文件，对其中图像的展示方式做了一些修改，参见 latency-plot.py。

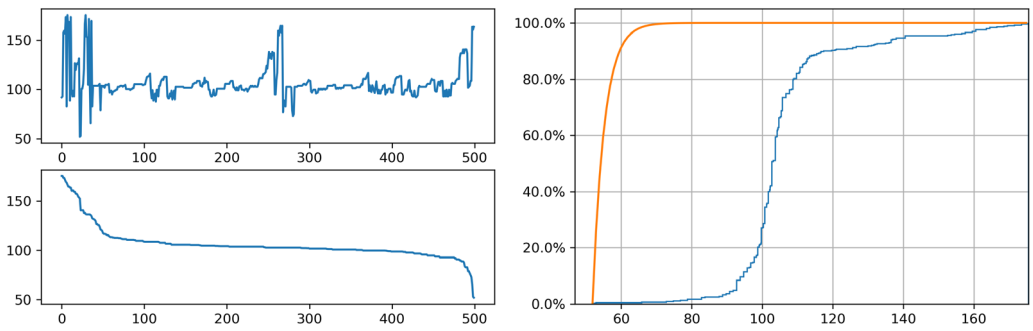
5.3.2 基准测试

设定基准测试的参数为 object_size=4kb、num_clients=10、num_samples=500，运行自己编写的基准测试程序，得到测试结果如表 3 和图 1 所示。

表 3 基准测试程序测试结果（性能指标）

请求类型	PUT	GET	DELETE
总延迟(ms)	5264.55164	4329.138279	4985.770941
吞吐率(KB/s)	379.8993983	461.985705	401.1415734
平均延迟(ms)	106.3329404	87.70697461	96.3898418
99%延迟(ms)	169.5463867	136.6345215	109.7087402
95%延迟(ms)	140.5375977	113.2075195	105.7182617
90%延迟(ms)	117.4372559	108.7089844	104.7199707
85%延迟(ms)	111.4572754	104.71875	103.7255859

PUT



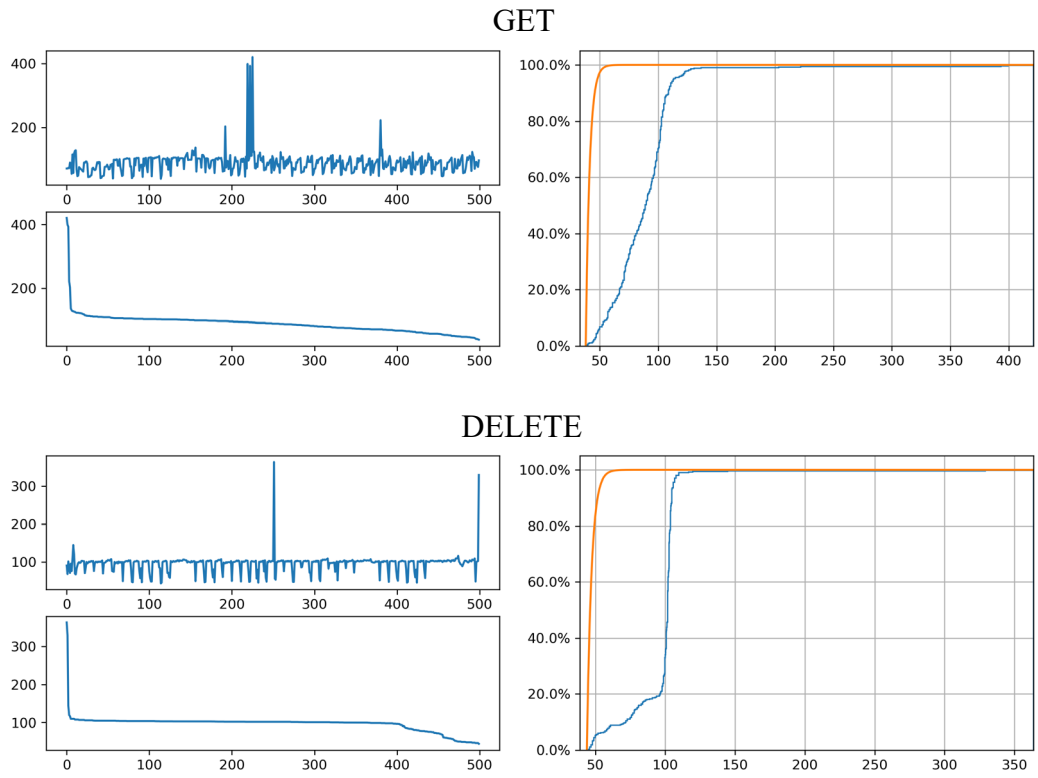


图 1 基准测试程序测试结果（延迟分布）

分析测试结果可知，在相同的参数下，PUT、GET、DELETE 三种类型请求的性能指标差异不大。GET 请求的延迟略小于 PUT 请求，猜测可能是由于实验误差，或者是由于客户端网络的下行速率大于上行速率。

观察图 1 中的延迟分布可以看到尾延迟现象，即所有请求的延迟在均值附近上下波动，偶尔还会出现剧烈波动。右图中蓝色曲线绘制的是测试结果的实际尾延迟曲线，橙色曲线是用排队论模型拟合测试结果得到的预测尾延迟曲线，比较图 1 以及后文中各图的这两条曲线可知，排队论模型把各百分位延迟都预测的偏小，模型预测准确率有待提升。

PUT 请求和 GET 请求相似，制约因素有客户端 I/O 速率、网络带宽、服务器端 I/O 速率等。DELETE 请求较为特殊，其制约因素只有服务器端 I/O 速率，与客户端和网络环境基本无关。后续实验我们将以 PUT 类型请求作为代表进行探究。

5.3.3 探究对象尺寸对性能的影响

设定参数 num_clients=10，保持 num_samples 和 object_size 的乘积恒定，改变 object_size 的大小，运行基准测试程序进行多组实验，仅取 PUT 请求的实验结果，如

表 4 和图 2 所示。

表 4 探究对象尺寸对性能的影响实验结果（性能指标）

object_size	num_clients	num_samples	总延迟(s)	吞吐率(KB/s)	平均延迟(ms)
1KB	10	2048	21.40177011	95.69301927	106.5350499
2KB	10	1024	10.40815401	196.7688024	102.9273007
4KB	10	512	5.206701756	393.3392186	103.4932604
8KB	10	256	2.602053642	787.0706302	102.3718548
16KB	10	128	1.344125986	1523.666696	105.177145
32KB	10	64	0.744123936	2752.229705	113.1536102
64KB	10	32	0.400296926	5116.20216	122.485405
128KB	10	16	0.484062433	4230.859202	236.9937134
256KB	10	8	0.379075527	5402.617297	349.5238342
1024KB	10	2	0.460682154	4445.58137	452.7906494

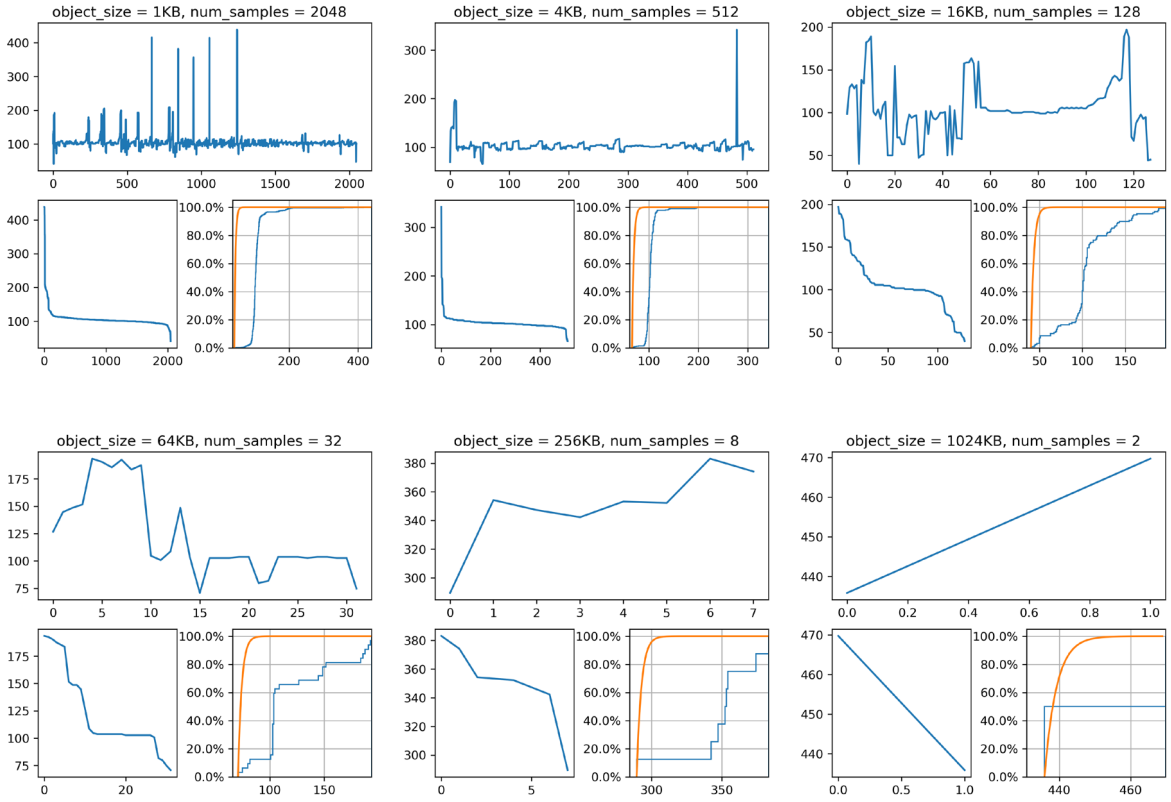


图 2 探究对象尺寸对性能的影响实验结果（延迟分布）（部分）

由表 4 可知，在对象尺寸较小时，随着对象尺寸的增加，总延迟呈反比减小，吞吐率呈正比增加，平均延迟基本不变；当对象尺寸超过 64KB 时，继续增大对象尺寸，吞吐率不再增加，总延迟不再减小，平均延迟开始与对象尺寸呈正比增加。

由表 1 知，对象存储系统服务器端的公网带宽为 50Mbps，因此表 4 中最后几组实验中，带宽已达到上限并成为系统的瓶颈，导致吞吐率不再增加，由于数据总量恒定，

所以总延迟也不在减小；由于单个对象的的尺寸增大，所以单个对象的平均延迟增大。

当对象尺寸较小时，带宽未达上限，在数据总量恒定的情况下，增大对象尺寸、减小对象数目，可以有效增大吞吐率、降低总延迟，推测其原因可能是降低了 I/O 开销，且与并发数较少有关。

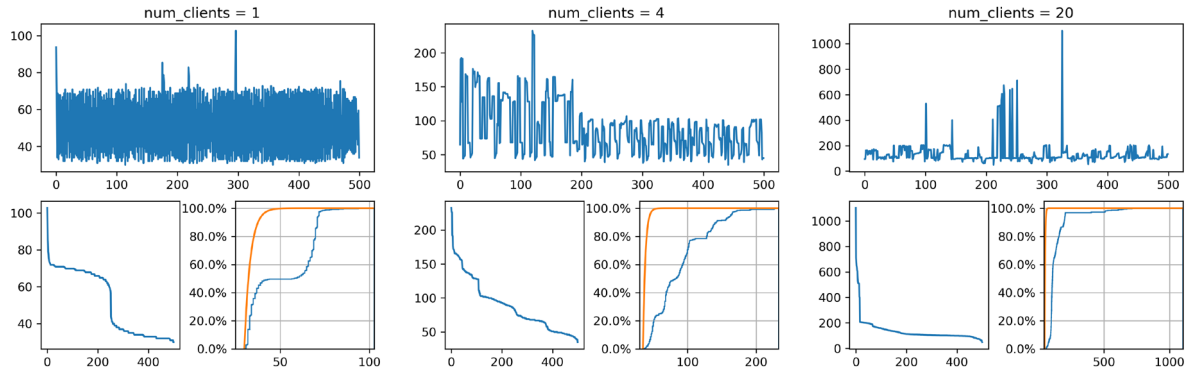
综上，应在不超过带宽上限的前提下，尽量把要传输的数据打包成较大尺寸的对象，可获得更高的吞吐率和更低的总延迟。

5.3.4 探究并发数对性能的影响

设定参数 object_size=4KB、num_samples=500，改变 num_clients 的大小，运行基准测试程序进行多组实验，仅取 PUT 请求的实验结果，如表 5 和图 3 所示。

表 5 探究并发数对性能的影响实验结果（性能指标）

object_size	num_clients	num_samples	总延迟(s)	吞吐率(KB/s)	平均延迟(ms)
4KB	1	500	25.19524789	79.38004853	51.54203906
4KB	2	500	16.20187306	123.4425175	66.2073291
4KB	4	500	10.8381052	184.534101	88.65008154
4KB	10	500	5.306123972	376.9229687	106.3329404
4KB	20	500	3.399389029	588.3410175	136.2508242
4KB	40	500	2.21363306	903.4921079	174.0384355
4KB	60	500	2.088909864	957.4371944	237.3332241
4KB	80	500	1.890930891	1057.680114	256.5314692
4KB	90	500	1.581104517	1264.938515	256.0460835
4KB	110	500	1.598968506	1250.806375	303.0448091
4KB	140	500	1.718657494	1163.698996	374.3952207
4KB	200	500	1.606169462	1245.198621	334.466082
4KB	300	500	1.58356905	1262.969872	315.2699668
4KB	400	500	1.699317455	1176.943127	359.1254121
4KB	500	500	1.66113019	1203.999549	349.985752



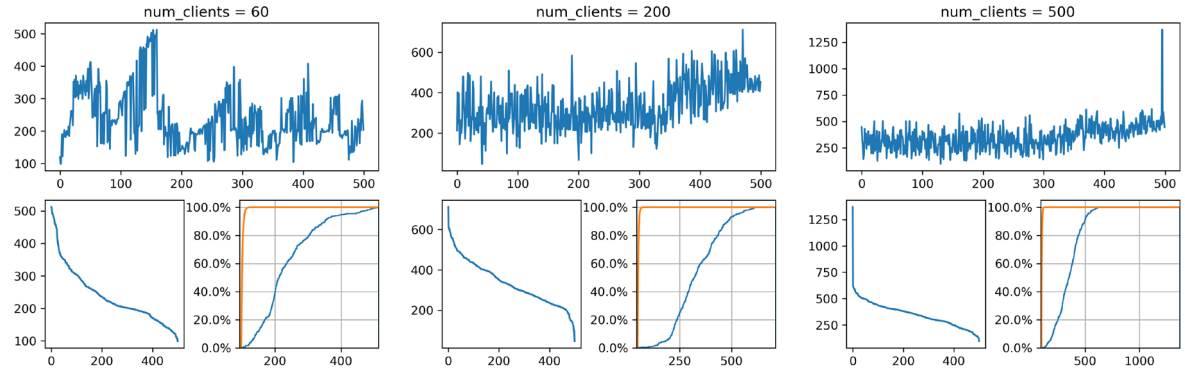


图 3 探究并发数对性能的影响实验结果（延迟分布）（部分）

由表 5 可知，在并发数较小时，随着并发数的增加，总延迟减小，吞吐率增加，这是并发操作带来的自然结果。实验结果还表明，随着并发数增加，平均延迟也增加，推测其原因可能是因为 I/O 操作远慢于计算操作，因此线程越多阻塞越严重。

当并发数超过 110 时，继续增大并发数，总延迟不再减小，吞吐率不再增加，平均延迟也不再增加，但由图 3 可知，系统延迟波动的剧烈程度增加，尾延迟增加。这表明对象存储系统服务器端遇到了瓶颈，由吞吐率数据仅为 1200KB/s 可知该瓶颈并非由带宽造成，推测是由服务器的 I/O 或 CPU 造成。

综上，为保障服务质量，应当对客户端并发数作出限制，即不应超过 110，否则可能会造成系统不稳定。

5.3.5 用对冲请求和关联请求方法应对尾延迟

对冲请求应对尾延迟的基本思想是：对于某个客户端的某个请求，如果它在发出第一次请求后 wait_time 时间内没有收到服务器端的响应，则立即发出第二次相同请求，如果在发出第二次请求后 wait_time 时间内仍没有收到任何响应，则发出第三次相同请求……依此类推，一共至多发送 request_num 次请求。在此过程中，一旦任何一次请求收到了响应，则立即取消其它所有请求，请求结束。

对冲请求程序有两个关键参数，wait_time 指相邻两次对冲请求发起的时间间隔，request_num 指对冲请求总共发送请求数的上限值。当 request_num=1 时，程序退化为前文采用的普通请求方式；当 wait_time=0 时，多路请求同时发出，即为关联请求方式。

本节所有实验均采用以下共同参数：object_size=4KB、num_clients=5、num_samples=200。首先设定 wait_time=0、request_num=1，运行对冲请求程序，得到

不采用对冲请求时的延迟分布情况，如图 4 所示，以了解应当如何设置对冲请求的时间间隔 wait_time。

总延迟(ms)	17.39002442
吞吐率(KB/s)	46.00338565
平均延迟(ms)	422.2212122
99%延迟(ms)	723.0664063
95%延迟(ms)	612.3618164
90%延迟(ms)	548.5334473
75%延迟(ms)	491.6850586
50%延迟(ms)	409.9038086
25%延迟(ms)	345.0776367

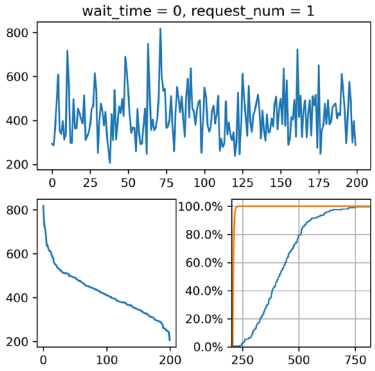
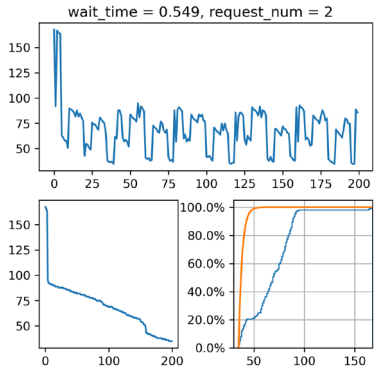


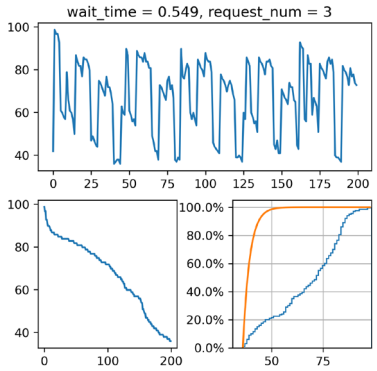
图 4 不采用对冲请求时的延迟分布情况

根据图 4，选定 wait_time=0.549s，改变 request_num，以探究对冲请求策略中请求数对缓解尾延迟效果的影响，运行对冲请求程序，得到实验结果如表 6 和图 5 所示。

表 6 探究对冲请求策略中请求数对缓解尾延迟效果的影响实验结果（性能指标）

wait_time	request_num	总延迟(s)	吞吐率(KB/s)	平均延迟(ms)
0.549	2	21.511814	37.18886727	68.47701294
0.549	3	21.514739	37.18381105	67.36516724
0.549	4	21.513762	37.18549933	67.70939697
0.549	5	21.510843	37.1905457	67.95859009





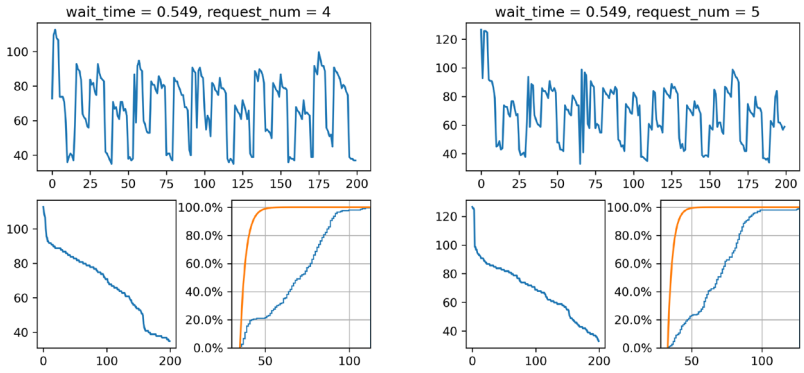


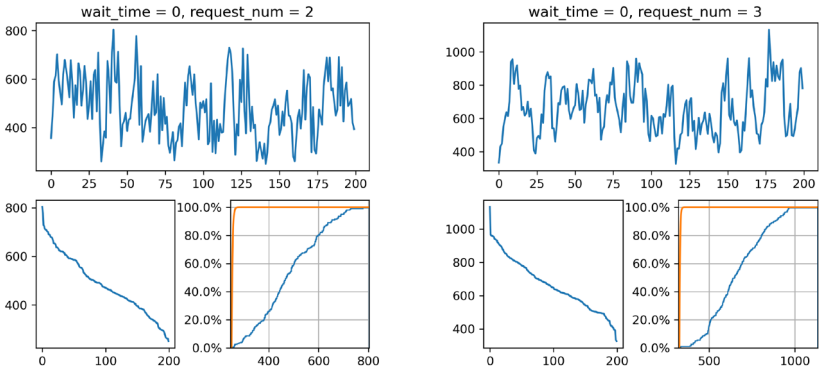
图 5 探究对冲请求策略中请求数对缓解尾延迟效果的影响实验结果（延迟分布）

由表 6 和图 5 可知，在对冲请求策略（wait_time＞0，request_num≥2）中，如果 wait_time 设置的是较高百分位延迟时间，那么请求数对缓解尾延迟效果影响不大，这是因为大部分请求只需要对冲一次就能等到服务器端的响应，需要对冲更多次的概率很小。

针对关联请求策略，wait_time 固定为 0，只有 request_num 一个变量。改变 request_num，以探究关联请求策略中请求数对缓解尾延迟效果的影响，如表 7 和图 6 所示。

表 7 探究关联请求策略中请求数对缓解尾延迟效果的影响实验结果（性能指标）

wait_time	request_num	总延迟(s)	吞吐率(KB/s)	平均延迟(ms)
0	2	19.736288	40.53447069	484.2251917
0	3	26.839365	29.80696397	663.1915283
0	4	39.909885	20.04515915	993.9272534
0	5	47.882706	16.70749337	1197.1838



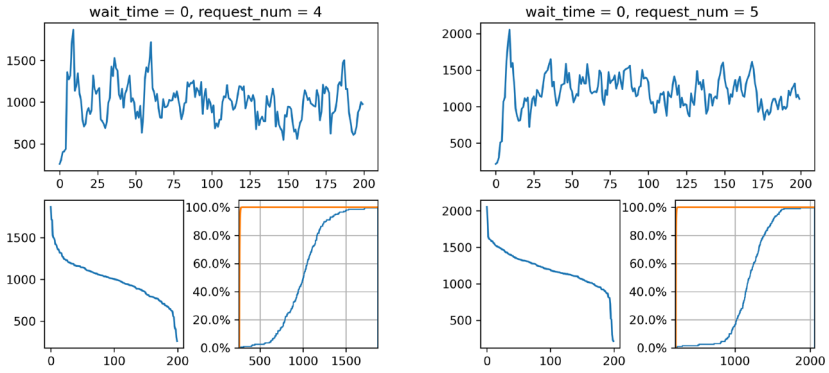


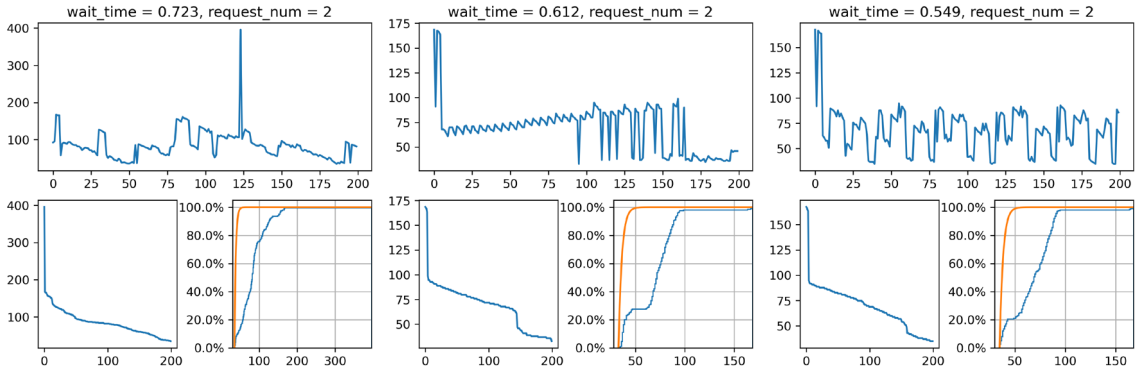
图 6 探究关联请求策略中请求数对缓解尾延迟效果的影响实验结果（延迟分布）

由表 7 和图 6 可知，使用关联请求策略时，尾延迟现象并未得到改善，且随着请求数增加，总延迟、平均延迟都呈正比增加，吞吐率呈反比减小，尾延迟现象更加严重。实验现象与理论分析不符，推测出现这种情况的原因可能是服务器端过载导致的。

根据上文分析，接下来固定 request_num=2，以图 4 结果作为指导，改变 wait_time 为不同百分位延迟时间，以探究对冲间隔时间对缓解尾延迟效果的影响，运行对冲请求程序，得到实验结果如表 8 和图 7 所示。

表 8 探究对冲间隔时间对缓解尾延迟效果的影响实验结果（性能指标）

wait_time	request_num	总延迟(s)	吞吐率(KB/s)	平均延迟(ms)
0.723	2	28.294463	28.27408348	84.59450317
0.612	2	23.96911	33.37629138	68.64169067
0.549	2	21.511814	37.18886727	68.47701294
0.492	2	19.2834	41.48645922	66.75712769
0.41	2	16.066417	49.79330347	73.58337036
0.345	2	13.531201	59.12261644	67.37002441
0.2	2	7.8773727	101.5567025	66.90647217
0.1	2	4.0594602	197.0705384	58.22974609
0.02	2	10.099947	79.20833289	247.1292566
0	2	19.736288	40.53447069	484.2251917



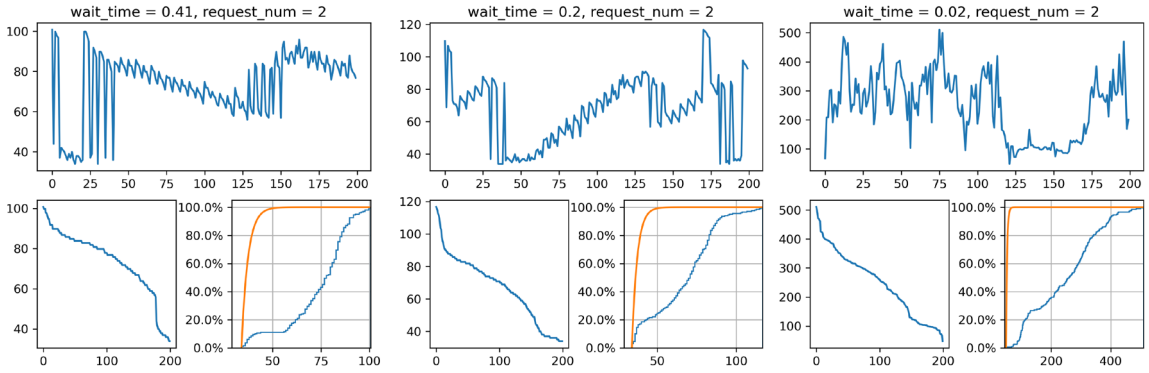


图 7 探究对冲间隔时间对缓解尾延迟效果的影响实验结果（延迟分布）（部分）

表 8 和图 7 的实验结果表明，对冲间隔时间不应设置为过高的百分位延迟时间，否则将不会引发对冲；对冲间隔时间也不应设置为过低的百分位延迟时间，否则情况和关联请求类似，可能会导致服务器端过载；其余的百分位延迟时间都能显著缓解尾延迟现象，且缓解效果大致相同，因此为了减少对冲请求的次数，应设置为尽量高百分位（如 95%、90%）的延迟时间。

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，掌握了云服务研发所需的一系列基础工具环境，包括 Git 版本管理、Python 后端语言、Docker 容器技术、云服务器的创建和使用等。

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，在编写基准测试程序的过程中熟练掌握了 Swift API 的使用。借助自行编写的基准测试程序，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，极大地培养了我自主设计实验、开展测量、分析数据的能力。在分析实验结果的过程中，我对对象存储系统性能的影响因素有了更深刻的认识。

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、多进程编程、阻塞、异步等概念有了初步了解，提升了编程能力。对冲请求的算法看上去非常简单，但真正实现起来才知道并不容易，我经过多次尝试

和失败，与同学不断讨论，锻炼了使用多种思路解决问题的能力。

由于时间原因，实验中还有很多地方值得我进一步学习和探究。例如尝试将对象存储系统的服务器数量和规模进行横向扩展，搭建对象存储集群，从而进一步探究服务器规模对系统性能的影响。又例如研读 mock-s3 等实验性模拟服务程序的源代码，学习对象存储系统在处理 PUT、GET、DELETE 等请求时是如何转化为对文件的各种操作的，深入理解对象存储系统的内部实现，以更准确地找出对象存储系统的瓶颈所在。

实验过程中，施展老师和同学们解答了我的很多困惑，给了我很多启发，衷心感谢老师和同学们的耐心指导和帮助！

参考文献

[1] 施展. 对象存储入门实践[EB/OL]. https://github.com/cs-course/obs-tutorial, 2021.

[2] 施展. 对象存储系统尾延迟问题[EB/OL]. https://shi_zhan.gitlab.io/data-center-course/data-center-2021-obs, 2021.

[3] OpenStack, LLC. SwiftClient Documentation[EB/OL]. https://docs.openstack.org/python-swiftclient/, 2022.

[4] Arnold J. OpenStack Swift: Using, Administering, and Developing for Swift Object Storage [M]. O'Reilly Media, 2014.

[5] Kapadia A, Rajana K, Varma S. OpenStack Object Storage (Swift) Essentials[M]. Packt Publishing Ltd, 2015.

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

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

[8] Ricardo Linck. Hedged requests — Tackling tail latency[EB/OL]. https://medium.com/swlh/hedged-requests-tackling-tail-latency-9cea0a05f577, 2020.