華中科技大學

数据中心技术课程实验报告

院系：计算机科学与技术学院

班级：硕 2110

姓名：高丹

学号：M202173856

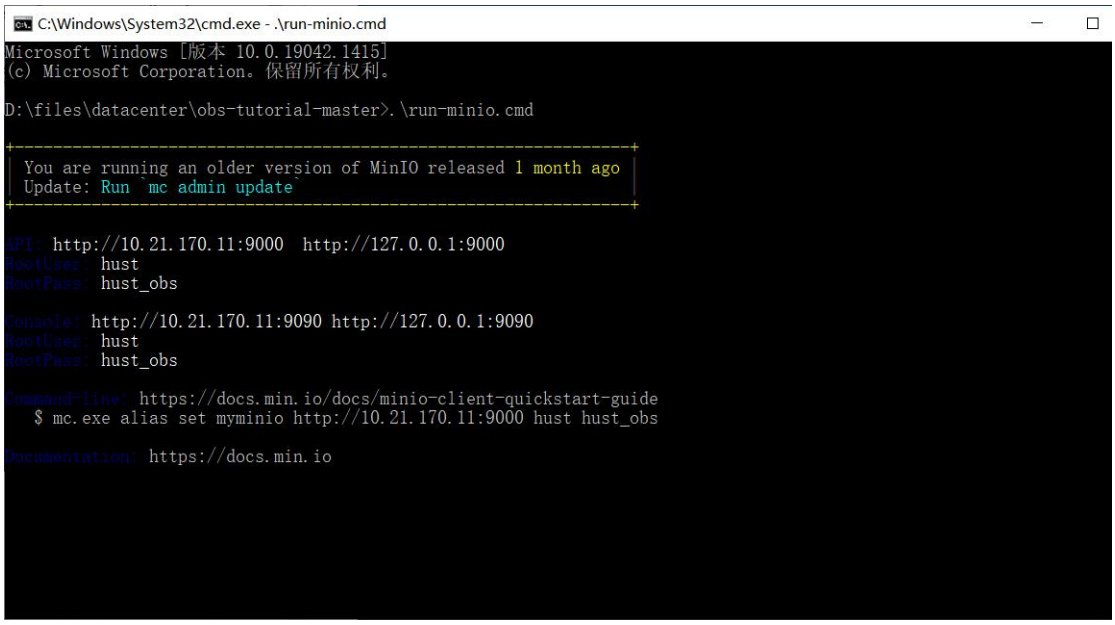
2021 年 1 月 2 日

实验一 系统搭建

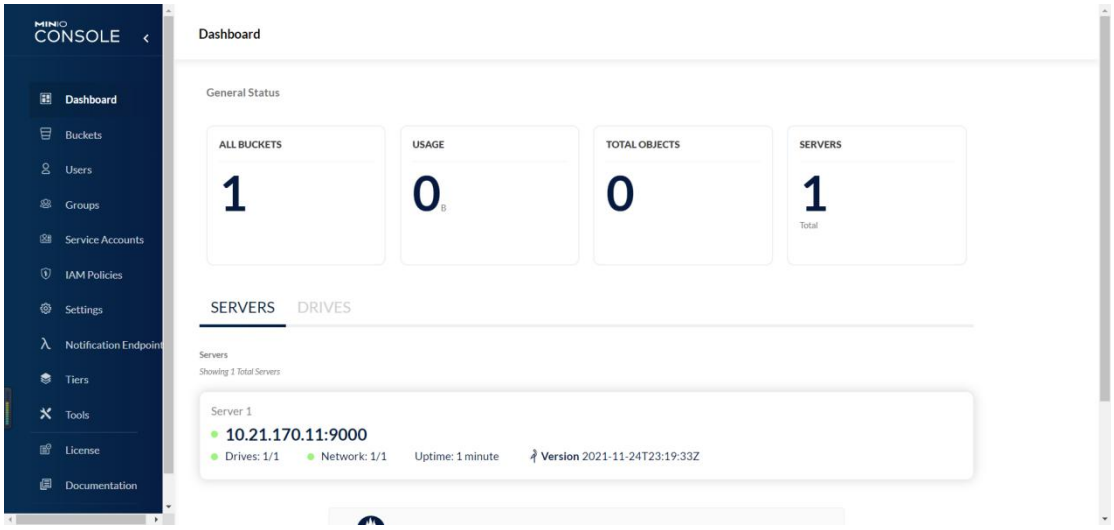
服务端：minio

测试工具：s3bench

首先下载 minio.exe 和 s3bench.exe，然后运行 run-minio.cmd 启动 minio 服务端。



在浏览器输入 127.0.0.1:9000 访问 minio 管理网站。

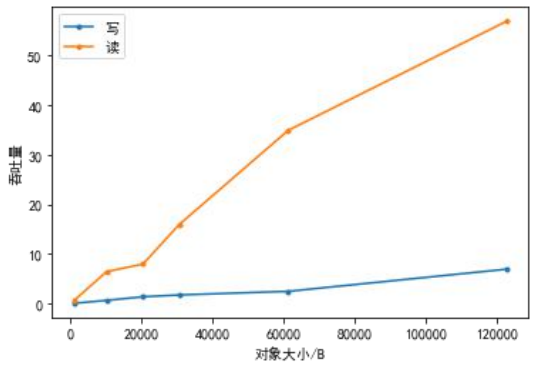


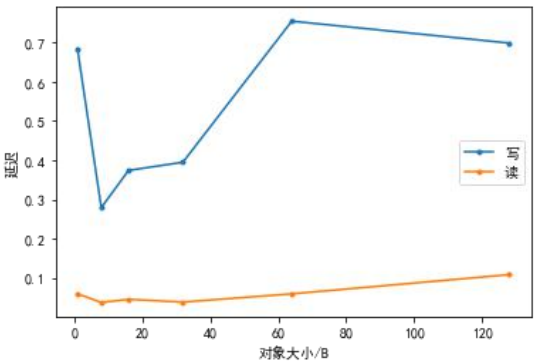
通过 s3bench 对 minio 进行测试，运行 run-s3bench.cmd。

```
C:\Windows\System32\cmd.exe - .\run-s3bench.cmd
Microsoft Windows [版本 10.0.19042.1415]
(c) Microsoft Corporation。保留所有权利。

D:\files\datacenter\obs-tutorial-master>.\run-s3bench.cmd

D:\files\datacenter\obs-tutorial-master>s3bench.exe     -accessKey=hust     -accessSecret=hust_obs     -bucket=exp     -
endpoint=http://127.0.0.1:9000     -numClients=8     -numSamples=256     -objectNamePrefix=exp     -objectSize=1024
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           exp
objectNamePrefix: exp
objectSize:       0.0010 MB
numClients:       8
numSamples:       256
verbose:      %!d(bool=false)


Generating in-memory sample data... Done (2.9919ms)

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           exp
objectNamePrefix: exp
objectSize:       0.0010 MB
numClients:       8
numSamples:       256
```

实验二 性能测试

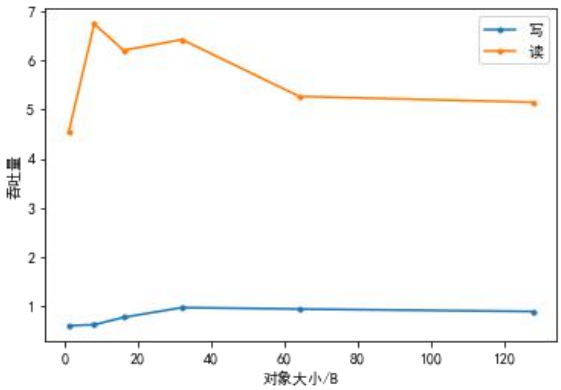
1. 改变对象大小，保持客户端数目为 8，对象数量为 256，对象大小依次为 1K，10K，20K，30K，60K，120K。读写吞吐量和延迟如下图所示。

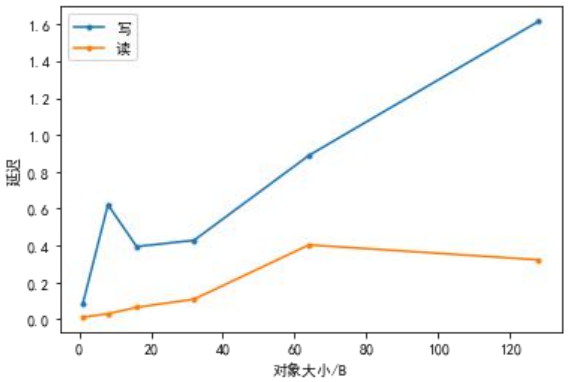




可以看到，随着对象大小的增加，吞吐量也在增加。随着对象大小的增加，读延迟有所增加，写延迟先减小后增大再减小，可以观察到当对象较小或者较大时写延迟较高。

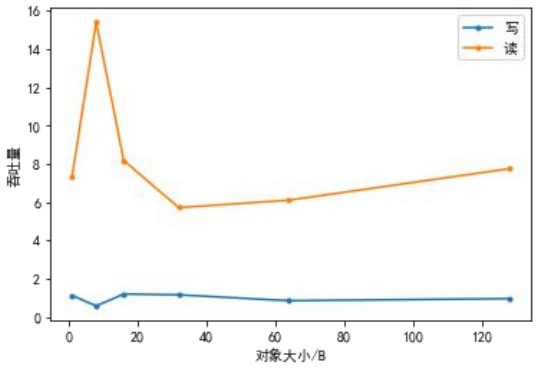
2. 改变客户端数目，保持对象数量为 256，对象大小为 10K，客户端数目依次为 1，8，16，32，64，128。

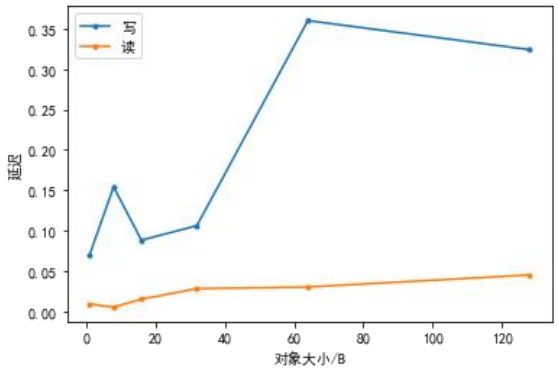




可以看到，随着客户端数量的增加，吞吐量先增大后减小，延迟呈增大的趋势。这意味着随着并发数据量增加，吞吐量不断上升；当系统资源达到高峰后，并发数量还在增加的话，服务器处理不过来了，吞吐量会逐渐下降。可以观察到并发数在 10-40 这个区间比较好。

3. 改变对象数量，保持客户端数量为 8，对象大小为 10K，对象数量依次为 8，16，32，64，128，256。





可以观察到随着对象数量的增加，读写延迟呈增长趋势，但是吞吐量先上升后下降再上升。

实验三 尾延迟挑战

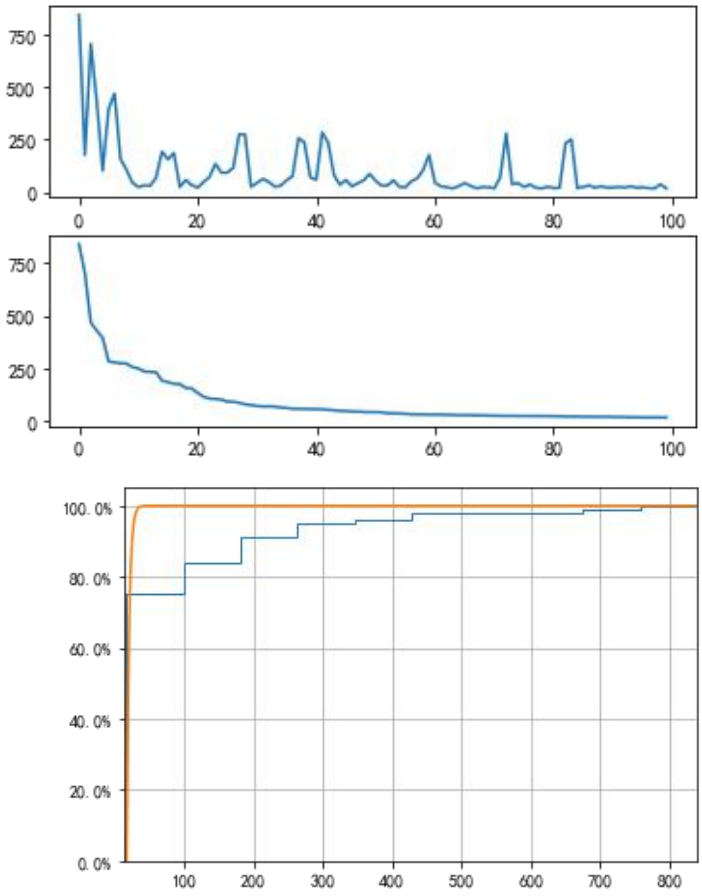
1. 尾延迟

首先连接本地服务，查看本地服务 ip 和端口并修改代码中的 local_s3 变量。

```python
import os
import time
from concurrent.futures import ThreadPoolExecutor, as_completed
from boto3.session import Session
import botocore
from tqdm import tqdm
import throttle

# 准备密钥
aws_access_key_id = 'hust'
aws_secret_access_key = 'hust_obs'

# 本地S3服务地址
local_s3 = 'http://10.21.170.11:9000'

# 建立会话
session = Session(aws_access_key_id=aws_access_key_id, aws_secret_access_key=aws_secret_access_key)

# 连接到服务
s3 = session.resource('s3', endpoint_url=local_s3)
```
✓ 0.4s

然后创建测试所用 bucket 'test100objs'，准备负载，记录每个请求完成的时间。

发起请求，预设 100 项上传任务，通过线程池进行执行，请求到达率设为无限制 arrival_rate_max，收集成功完成的请求的返回结果即 latency，将 latency 保存到 csv 文件中绘制 latency 结果图，清除实验环境。最终结果如下：

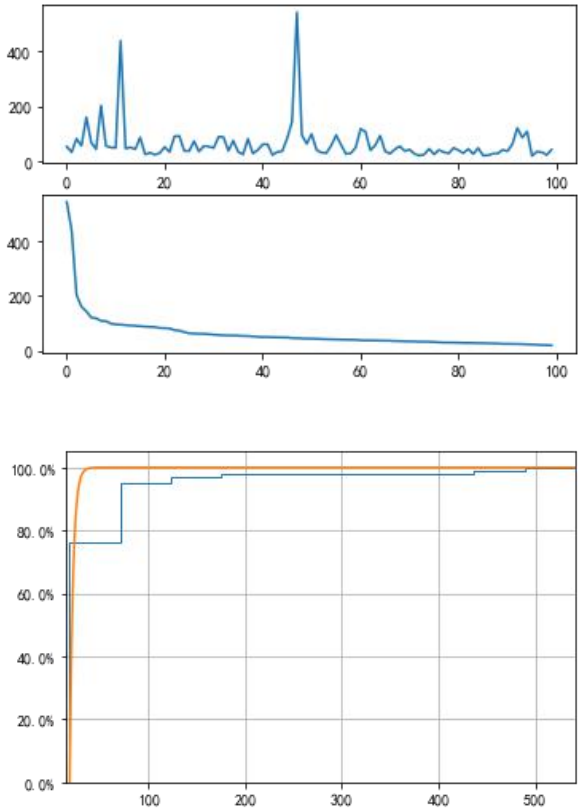


有一部分写请求的延迟远远高于其他请求，即出现了尾延迟现象。

2.对冲请求

观察到图中有 90%的请求延迟都在 250ms 以下，少数请求延迟远大于 250，因此针对延迟大于 250ms 的请求让其重试即发起对冲请求，对冲延时设置为

250ms，当请求在对冲延时内没有收到回复时立刻发起新的尝试。修改后结果如下：



可以看到修改之后虽然尾延迟现象还是存在但是改善了很多，修改前最高延迟达到 700 多，现在最多 400 多 ms。