

2019 级

《物联网数据存储与管理》课程

实 验 报 告

姓 名 李启鑫

学 号 U201915165

班 号 计算机 1908 班

日 期 2022.04.14

目 录

一、实验目的.....	1
二、实验背景.....	1
三、实验环境.....	1
四、实验内容.....	1
4.1 对象存储技术实践.....	1
4.2 对象存储性能分析.....	2
五、实验过程.....	2
5.1 服务端程序安装.....	2
minio.....	2
mock-s3.....	2
5.2 客户端程序安装.....	3
minio.....	3
mock-s3.....	4
5.3 性能评测.....	4
修改对象尺寸.....	5
修改客户端数.....	7
六、实验总结.....	8
参考文献.....	8

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

对象存储，也叫做基于对象的存储，是用来描述解决和处理离散单元的方法的通用术语，这些离散单元被称作为对象。其对应存储和保护大量非结构化的数据所带来的挑战提供了一个直接的响应。

对象存储综合了 NAS 和 SAN 的优点，同时具有 SAN 的高速直接访问和 NAS 的分布式数据共享等优势，提供了具有高性能、高可靠性、跨平台以及安全的数据共享的存储体系结构，非常适用于现在的海量数据的场景。

Mock-S3，是用 Python 重写 fake-S3 实现的，沙盒环境中测试非常有用，无需实际调用 Amazon，其目标是最小化运行时依赖关系，并更像是一个开发工具来测试代码中的 S3 调用。

Object Store Manipulator，对象存储操纵器，用于云存储服务的 curl。osm 可以为 AWS S3、AWS S3 兼容的其他存储服务（即 Minio）、DigitalOcean Spaces、谷歌云存储、Microsoft Azure 存储和 OpenStack Swift 创建和删除存储桶，并从存储桶上载、下载和删除文件。

评测工具 S3 Bench，此工具提供了针对 S3 兼容端点运行非常基本的吞吐量基准测试的能力。它执行一系列的 put 操作，然后执行一系列的 get 操作，并显示相应的统计信息

三、实验环境

表 1 实验环境

CPU	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
内存	16GB
操作系统	Windows 11 家庭中文版
Python	Python 3.8.4
服务端	Mock-s3
客户端	OSM
评测工具	S3 Bench

四、实验内容

4.1 对象存储技术实践

1. 熟悉代码管理和仓库；
2. 在 Linux 和 window 下分别配置服务端和客户端的开发环境。
3. 下载实验任务书中的 mock-s3 链接以及 osm 的 window 编译文件，安装配置。
4. 客户端通过 osm 连接服务端，并尝试对象存储基本命令操作。

4.2 对象存储性能分析

1. 选择 s3-bench 作为评测工具
2. 编写 cmd 脚本文件，运行 s3-bench，改变不同参数，多次循环
3. 观察实验结果，导出实验数据

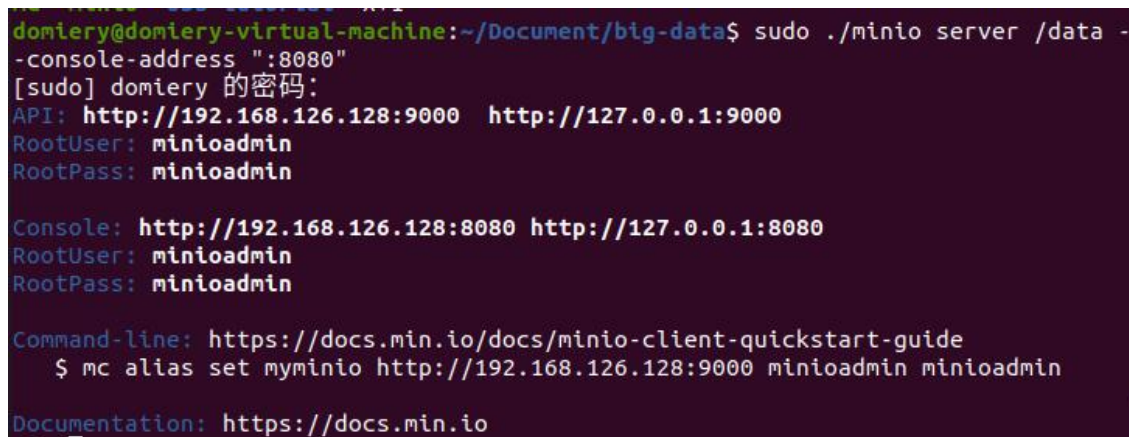
五、实验过程

实验时先尝试了 minio 服务端、客户端搭建，之后使用了 mock-s3 以及 osm 进行替换，下面将叙述这两种方式的实验过程。

5.1 服务端程序安装

minio

1. 进入虚拟机命令行，输入下述命令下载 minio
wget https://dl.min.io/server/minio/release/linux-amd64/minio
2. 使用 chmod +x minio 改变 minio 文件权限
3. 输入 sudo ./minio server /data 启动服务



```
domiery@domiery-virtual-machine:~/Document/big-data$ sudo ./minio server /data -
-console-address ":8080"
[sudo] domiery 的密码:
API: http://192.168.126.128:9000 http://127.0.0.1:9000
RootUser: minioadmin
RootPass: minioadmin

Console: http://192.168.126.128:8080 http://127.0.0.1:8080
RootUser: minioadmin
RootPass: minioadmin

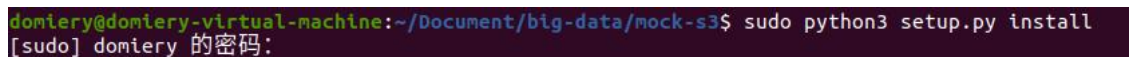
Command-line: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc alias set myminio http://192.168.126.128:9000 minioadmin minioadmin

Documentation: https://docs.min.io
```

图 1 minio 启动

mock-s3

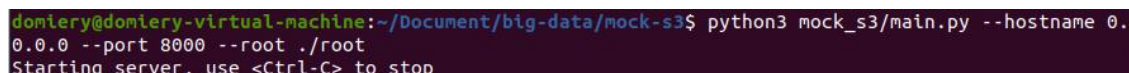
1. Unbutun 虚拟机自带 python 开发环境，无需下载。在 GitHub 上下载 mock-s3 源码，使用 install 命令安装。



```
domiery@domiery-virtual-machine:~/Document/big-data/mock-s3$ sudo python3 setup.py install
[sudo] domiery 的密码:
```

图 2 mock-s3 安装

2. 在 CMD 中输入对应命令以启动服务，为服务器指定端口、根目录参数。mock-s3 没有访问密钥；默认使用本地回环测试地址。



```
domiery@domiery-virtual-machine:~/Document/big-data/mock-s3$ python3 mock_s3/main.py --hostname 0.
0.0.0 --port 8000 --root ./root
Starting server, use <Ctrl-C> to stop
```

图 3 开启 mock-s3 服务

3. 访问 mock-s3 网页

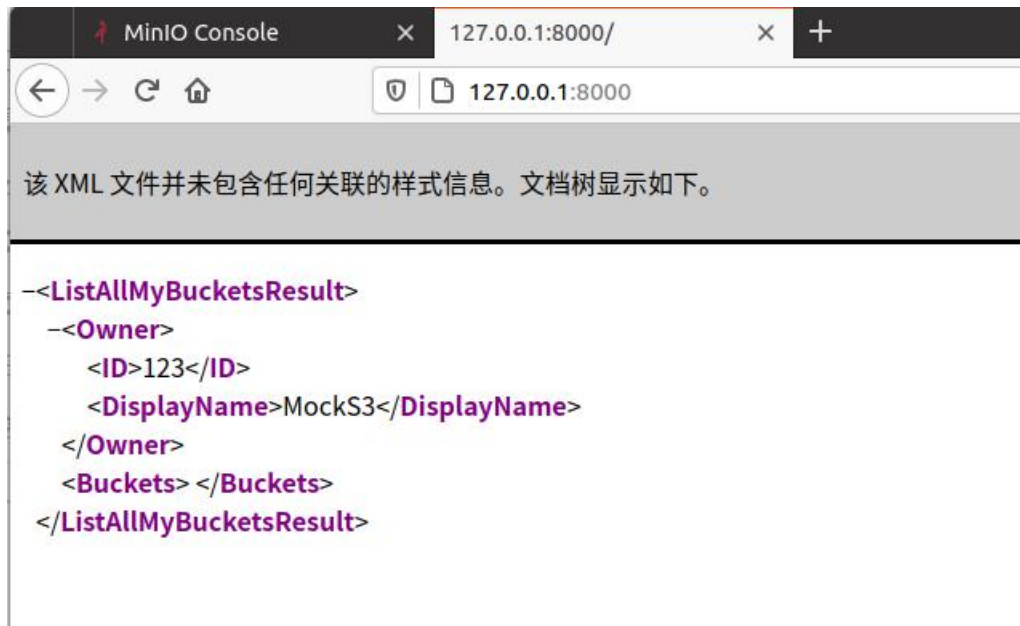


图 4 mock-s3 网页

5.2 客户端程序安装

minio

1. 从 minio 官网下载客户端程序 mc.exe
2. 设置 minio 客户端连接服务端的地址、密钥、公钥

```
C:\Users\Dominery\Desktop\大学课程\大三下\大数据存储\实验>mc.exe alias set myminio/ http://192.168.126.128:9000 minioadmin minioadmin
Added myminio successfully.
```

图 5 设置客户端

3. 简单命令尝试

```
C:\Users\Dominery\Desktop\大学课程\大三下\大数据存储\实验>mc.exe mb myminio/mybucket
Bucket created successfully myminio/mybucket .
```

图 6 创建 bucket

```
C:\Users\Dominery\Desktop\大学课程\大三下\大数据存储\实验>mc.exe ls myminio
[2022-03-31 20:00:16 CST] 0B bucket/
[2022-03-31 19:50:16 CST] 0B mybucket/
[2022-03-31 19:59:04 CST] 0B wasabi-benchmark/
```

图 7 查看 bucket

4. 返回 minio 服务器提供的网页，发现 bucket 创建成功

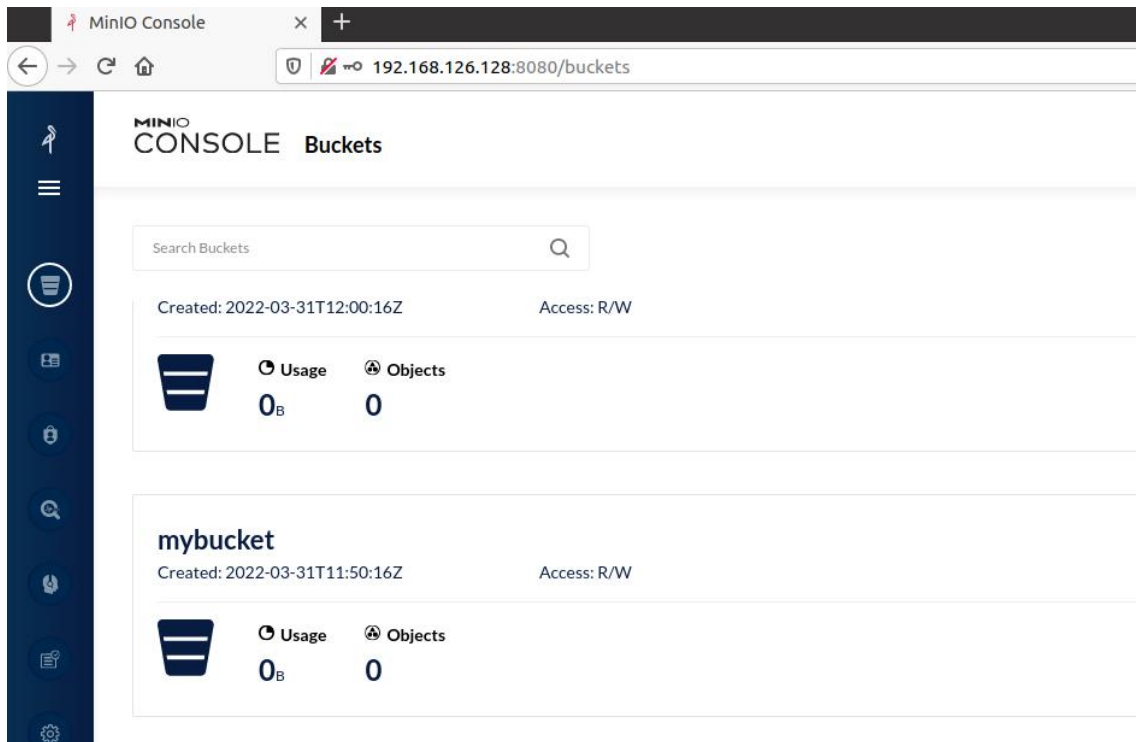


图 8 服务端网页

mock-s3

1. 从代码仓库下载 windows 的可执行文件 `osm.exe`
2. 下载 `config-osm.cmd` 文件，修改 ip 地址后，执行

```
C:\Users\Dominery\Desktop\大学课程\大三下\大数据存储\实验>config-osm.cmd
contexts:
- config:
  access_key_id: hust
  auth_type: accesskey
  disable_ssl: "false"
  endpoint: http://192.168.126.128:8000
  secret_key: hust_obs
  name: osm-minio
  provider: s3
current-context: osm-minio
```

图 9 执行 config-osm.cmd

3. 尝试基础命令

```
C:\Users\Dominery\Desktop\大学课程\大三下\大数据存储\实验>osm push -c U201915165 ./result1.txt result1
.txt
Successfully pushed item result1.txt

C:\Users\Dominery\Desktop\大学课程\大三下\大数据存储\实验>osm ls U201915165
result1.txt
Found 1 item in container U201915165
```

图 10 基础命令尝试

5.3 性能评测

1. 启动 mock-s3，开启对象存储服务

2. 修改 run-s3bench.cmd 脚本文件，在多次循环中改变请求参数
3. 运行脚本文件
4. 获取数据，观察修改的参数对读写吞吐率以及读写延迟的影响

```

Write times 90th %ile: 0.019 s
Write times 75th %ile: 0.015 s
Write times 50th %ile: 0.013 s
Write times 25th %ile: 0.012 s
Write times Min:      0.008 s

Results Summary for Read Operation(s)
Total Transferred: 0.249 MB
Total Throughput:  0.20 MB/s
Total Duration:    1.226 s
Number of Errors:  1
=====
Read times Max:      1.004 s
Read times 99th %ile: 0.229 s
Read times 90th %ile: 0.022 s
Read times 75th %ile: 0.014 s
Read times 50th %ile: 0.011 s
Read times 25th %ile: 0.009 s
Read times Min:      0.004 s

Cleaning up 256 objects...
Deleting a batch of 256 objects in range {0, 255}... Succeeded
Successfully deleted 256/256 objects in 57.9255ms

```

图 11 脚本文件运行结果

修改对象尺寸

对象尺寸	客户端数	样本数量	写吞吐率	写99延迟	写90延迟	读吞吐率	读99延迟	读90延迟
0.002	8	256	0.49	0.02	0.014	0.36	0.207	0.012
0.0039	8	256	0.55	0.058	0.038	2.17	0.218	0.014
0.0059	8	256	0.99	0.043	0.029	1.39	0.041	0.026
0.0078	8	256	1.84	0.045	0.032	2.89	0.207	0.029
0.0098	8	256	2.07	0.069	0.041	1.61	0.214	0.021
0.0117	8	256	2.77	0.055	0.031	2.81	0.043	0.025
0.0137	8	256	2.76	0.049	0.034	2.41	0.223	0.034
0.0156	8	256	2.13	0.519	0.058	5.52	0.206	0.023
0.0176	8	256	3.72	0.056	0.044	5.62	0.206	0.023
0.0195	8	256	4.24	0.051	0.036	3.42	0.206	0.024
0.0215	8	256	2.62	0.512	0.038	5.41	0.032	0.014
0.0234	8	256	4.49	0.043	0.034	8.84	0.032	0.024
0.0254	8	256	3.79	0.508	0.037	5.21	0.22	0.027
0.0273	8	256	3.42	0.099	0.052	8.34	0.209	0.027
0.0293	8	256	3.55	0.513	0.058	5.11	0.207	0.03
0.0312	8	256	3.77	1.013	0.042	5.39	0.209	0.037
0.0332	8	256	3.2	1.012	0.066	4.95	0.047	0.033
0.0352	8	256	3.4	1.023	0.07	6.23	0.21	0.045
0.0371	8	256	6.37	0.044	0.031	7.13	0.214	0.021
0.0391	8	256	9.18	0.088	0.056	8.04	0.08	0.044

图 12 对象尺寸改变后数据记录表

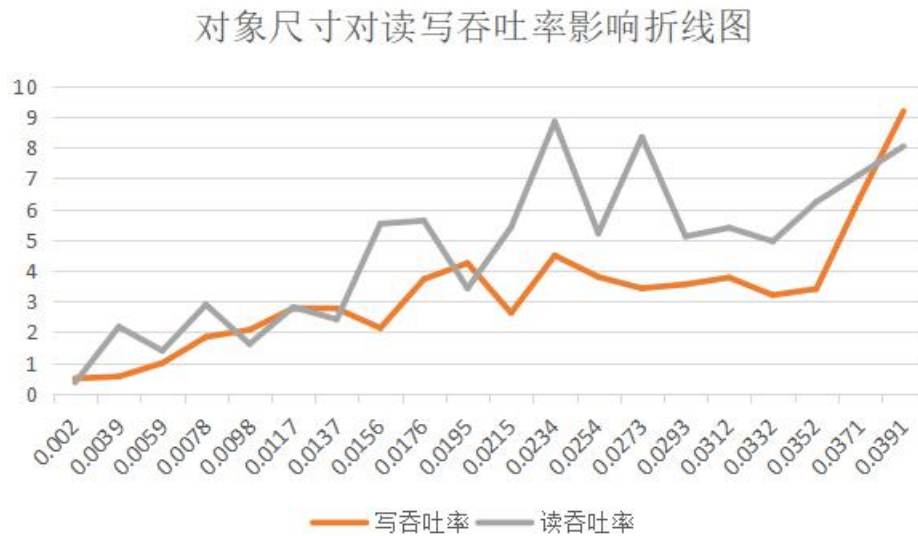


图 13 对象尺寸对读写吞吐率影响

分析：

对象尺寸增大，读写吞吐率随之增加，读写吞吐率与对象尺寸呈线性相关。

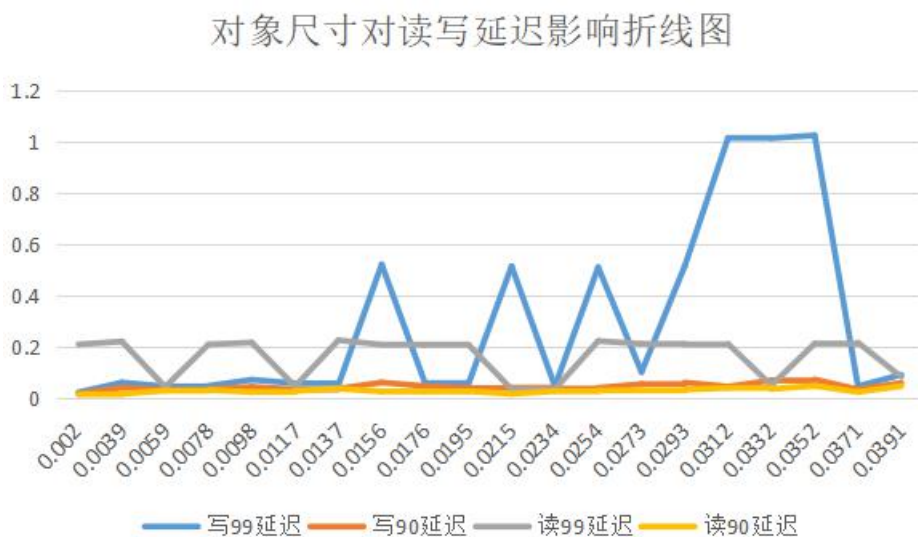


图 14 对象尺寸对读写延迟影响

分析：

对象尺寸增大，写 90 延迟、读 90 延迟几乎没有变化，读 99 延迟呈现低水平波动的规律，写 99 延迟波动随着对象尺寸增大而增大。

修改客户端数

对象尺寸	客户端数	样本数量	写吞吐率	写99延迟	写90延迟	读吞吐率	读99延迟	读90延迟
0.002	1	256	0.7	0.008	0.005	0.66	0.01	0.006
0.002	2	256	0.67	0.017	0.01	0.73	0.015	0.009
0.002	3	256	0.46	0.073	0.018	0.51	0.096	0.02
0.002	4	256	0.34	0.108	0.041	0.5	0.046	0.024
0.002	5	256	0.65	0.03	0.022	0.37	0.125	0.051
0.002	6	256	0.65	0.038	0.026	0.78	0.028	0.023
0.002	7	256	0.46	0.048	0.03	0.37	0.043	0.025
0.002	8	256	0.37	0.049	0.034	0.4	0.247	0.035
0.002	9	256	0.4	0.507	0.03	0.3	0.237	0.039
0.002	10	256	0.31	1.004	0.037	0.33	0.246	0.047
0.002	11	256	0.32	1.004	0.03	0.37	0.211	0.027
0.002	12	256	0.32	1.006	0.034	0.3	1.003	0.021
0.002	13	256	0.32	1.004	0.025	0.32	1.004	0.029
0.002	14	256	0.32	1.004	0.038	0.28	1.004	0.034
0.002	15	256	0.31	1.008	0.043	0.27	1.003	0.033
0.002	16	256	0.39	1.005	0.028	0.25	1.004	0.034
0.002	17	256	0.46	1.034	0.037	0.22	1.017	0.064
0.002	18	256	0.39	1.029	0.029	0.25	1.006	0.043
0.002	19	256	0.3	1.023	0.033	0.24	1.006	0.038
0.002	20	256	0.16	3.007	0.035	0.34	1.002	0.015

图 15 客户端数改变后数据记录表

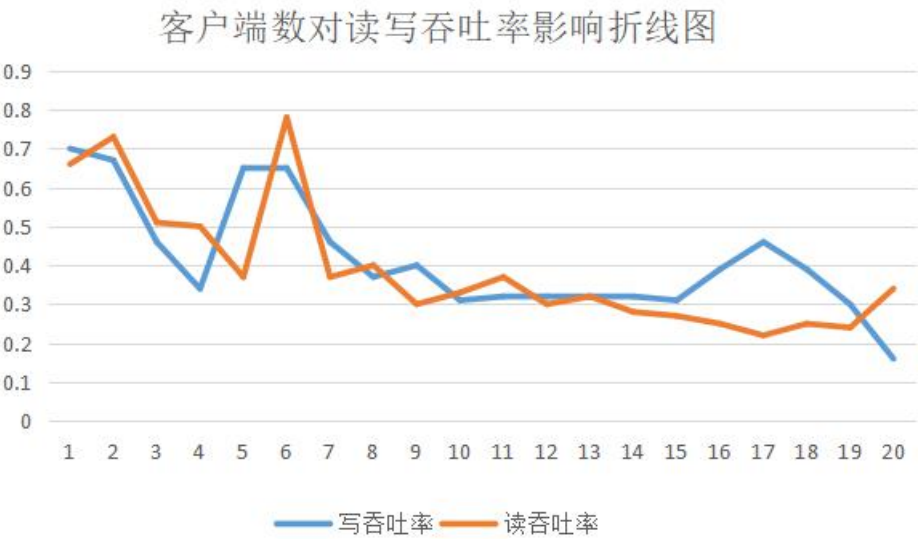


图 16 客户端数对读写吞吐率影响

分析：
客户端数增加，读写吞吐率降低

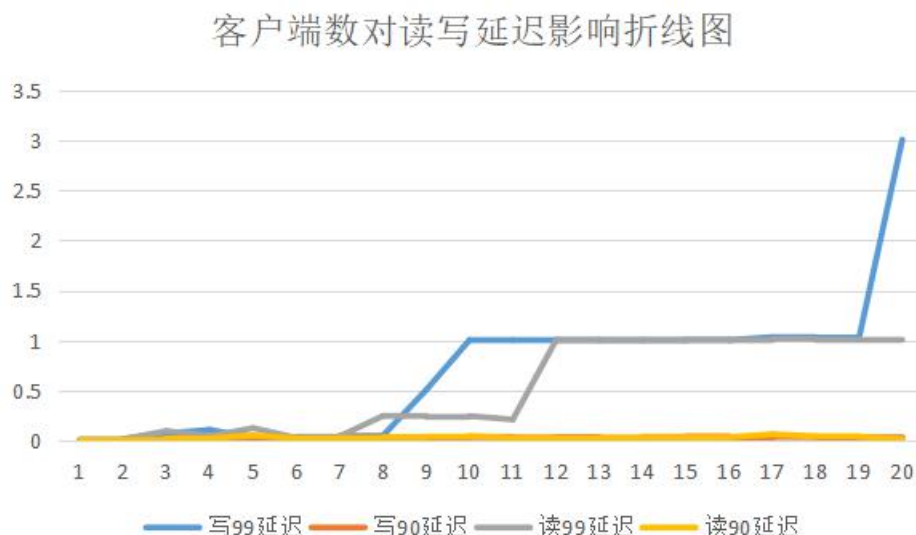


图 17 客户端数对读写延迟影响

分析:

客户端数增加对读写 90 延迟都没有影响, 读写 99 延迟有增加趋势。

六、实验总结

这次实验在最开始, 我使用 minio 搭建服务端、mc 搭建客户端, 成功运行对象存储服务后, 我转而使用了 osm、mock-s3。我一开始打算使用 s3-benchmark 性能测试工具, 但很快就遇上难以解决的问题, 之后我使用老师提供的 windows 执行程序能够正常执行。

这次实验让我了解了业内使用对象存储技术的多种方案, 能够手动搭建一个简单的对象存储服务端、客户端, 还能够分析对象存储方案的性能。

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.

(可以根据实际需要更新调整)