
2018 级

# 《物联网数据存储与管理》课程

# 实 验 报 告

姓　名　曹金羽

学　号　U201814597

班　号　物联网 1801 班

日　期　2021.06.21

# 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，架设实际应用，示范主要功能。

# 二、实验背景

随着互联网的不断扩张和云计算技术的进一步推广，海量的数据在个人、企业、研究机构等源源不断地产生，如何有效、快速、可靠地存取这些日益增长的海量数据成了关键的问题。

传统的存储解决方案面临越来越多的困难，比如数据量的指数级增长对不断扩容的存储空间提出要求、实时分析海量的数据对存储计算能力提出要求等。因此，需要能处理海量数据、高性能、易扩展、可伸缩、高可用的新型存储方案。

对象存储系统（Object-Based Storage System）是综合了 NAS 和 SAN 的优点，同时具有 SAN 的高速直接访问和 NAS 的数据共享等优势，提供了高可靠性、跨平台性以及安全的数据共享的存储体系结构。

对象存储架构由对象、对象存储设备、元数据服务器、对象存储系统的客户端四部分组成，其核心是将数据通路（数据读或写）和控制通路（元数据）分离，并且基于对象存储设备构建存储系统，每个对象存储设备具有一定的智能，能够自动管理其上的数据分布。

# 三、实验环境

实验环境如表 1 所示。

表 1 实验环境说明

| | |
|---|---|
| CPU | Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz |
| 内存 | 16GB |
| 操作系统 | Windows 10 家庭中文版 |
| Go | go version go1.14.3 windows/amd64 |
| Python | Python 3.7.10 |
| 客户端 | Osm |
| 服务端 | Mock-s3 |
| 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；配置 Python、Go 等开发、运行环境。
2. 依照实验任务指导，根据系统开发环境选择合适的对象存储系统服务端，并进行安装与配置。本次实验选用实验性模拟服务程序 mock-s3。
3. 选择合适的对象存储系统客户端，并进行安装与配置。本次实验选用 osm。

## 4.2 对象存储性能分析

1. 选择合适的对象存储评测工具，并进行安装与配置。本次实验选用 S3 Bench。
2. 设置评测参数（客户端数量、对象数量、对象大小等），运行评测工具，初步观察对象存储性能。
3. 在 Windows 环境下编写 cmd 脚本，设计循环结构实现批量测试，将结果重定向保存到文件用于后期分析。
4. 整理批量测试数据，对不同参数条件下的存储性能进行分析。

# 五、实验过程

## 5.1 搭建对象存储服务端

1. 实验提供的 mock-s3 依赖 Python 3 环境运行。使用 Anaconda 平台，为创建 Python 虚拟环境（Python 3.7.10）。
2. 从实验指导仓库下载 mock-s3 对应版本源码。按照程序使用说明，在对应目录下运行 setup.py 程序安装 mock-s3。
3. 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。在主机浏览器访问设置的地址与端口，可见服务器端文件结构，证明 mock-s3 服务启动成功。命令行输出与浏览器显示如图 1 所示。

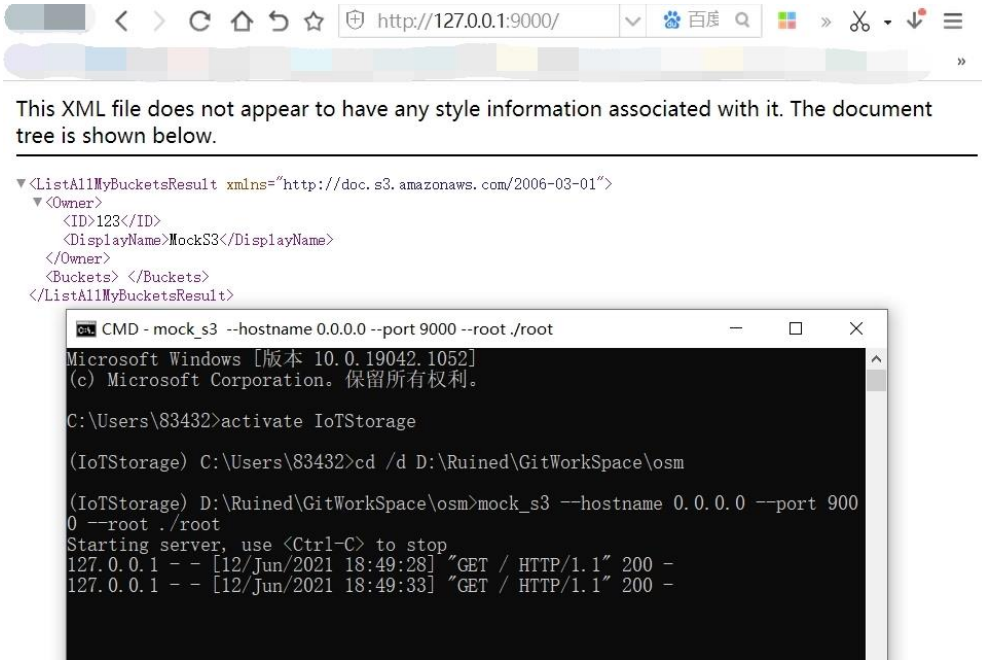


图 1 mock-s3 服务命令行输出与浏览器显示

## 5.2 搭建对象存储客户端

1. 从实验指导仓库下载 Windows 版 osm.exe 可执行文件。
2. 在 CMD 中对应目录下执行 osm 配置脚本 config-osm.cmd。

3. 在 CMD 中执行 osm -h 命令，可见相应输出结果，证明 osm 安装配置成功。如图 2 所示。

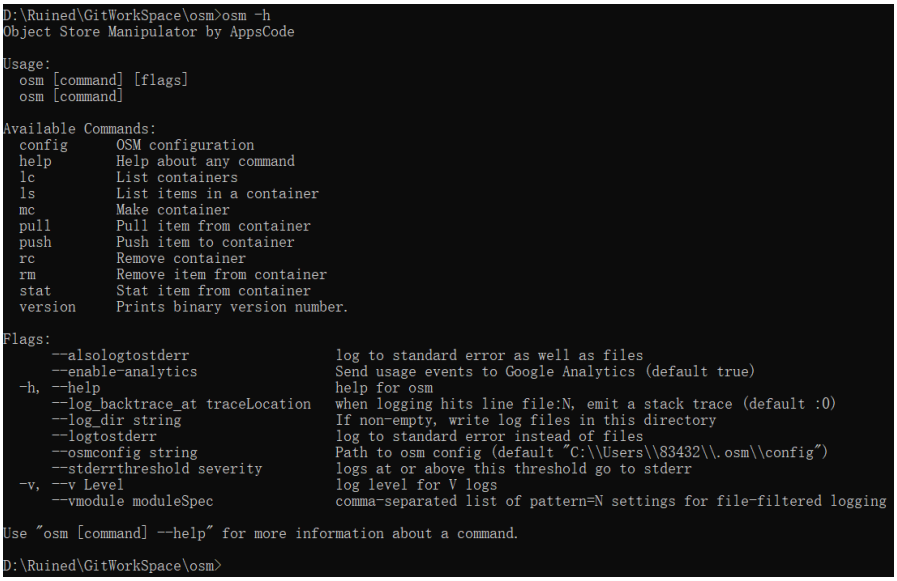


图 2 osm -h 命令行输出

## 5.3 初步测试对象存储功能

1. 在 CMD 中以相同参数启动 mock-s3 服务，保持服务运行。
2. 在另一 CMD 窗口中，执行 osm 的 Bucket 管理命令与存储管理操作：创建 bucket、上传测试文件。可见对象存储系统功能正常。osm 客户端操作与服务端输出如**错误!未找到引用源。**所示。

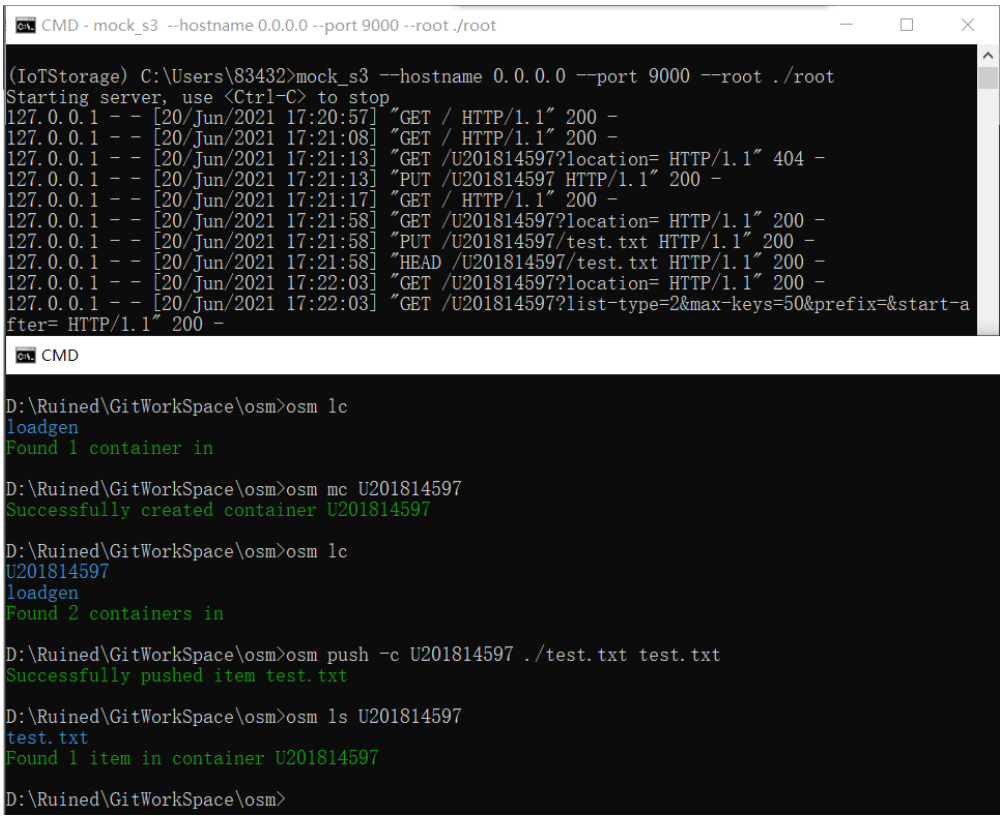


图 3 osm 客户端操作与 mock-s3 服务端输出

## 5.4 对象存储性能评测

1. 从实验指导仓库下载 Windows 版预编译可执行文件 s3bench.exe。
2. 在 CMD 中对应目录下运行范例脚本 run-s3bench.cmd，以默认配置启动 S3 Bench 进行存储性能评测。默认评测参数与评测结果如图 4 所示。可见 S3 Bench 成功安装与并配置，能正常完成对当前对象存储系统的评测。

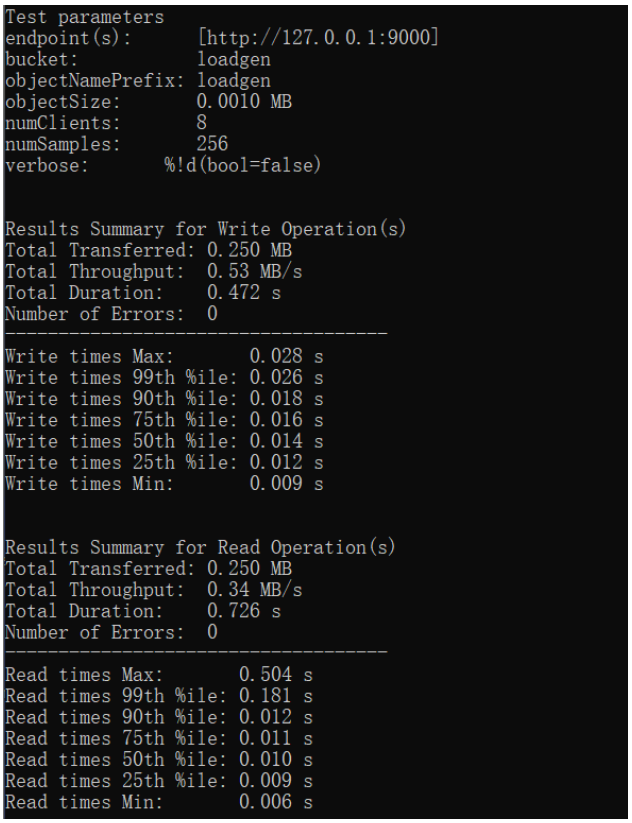


图 4 S3 Bench 范例脚本评测参数与评测结果

3. 修改范例脚本 run-s3bench.cmd，增加循环结构，梯度改变评测参数进行批量测试：
   ① 设置并发客户端数量 numClients = 8，样本数量 numSamples = 256 不变。使对象大小 objectSize 由 2048 字节开始梯度增加，共进行 20 次测试，将测试结果保存到相应文本中。测试脚本代码如下：

```
@echo off
@echo.>result1.txt
set /a numClients = 8
set /a objectSize = 0
setlocal enabledelayedexpansion
for /l %%i in (0,1,19) do (
    set /a objectSize += 2048
    s3bench.exe ^
        -accessKey=hust ^
        -accessSecret=hust_obs ^
        -bucket=loadgen ^
        -endpoint=http://127.0.0.1:9000 ^
```

```
        -numClients=!numClients! ^
        -numSamples=256 ^
        -objectNamePrefix=loadgen ^
        -objectSize=!objectSize! >> result1.txt
        echo !objectSize!      )
```

② 设置对象大小 objectSize = 1024，样本数量 numSamples = 256 不变。使并发客户端数量由 1 个开始梯度增加，共进行 20 次测试，将测试结果保存到相应文本中。测试脚本如下：

```
set /a numClients = 0
set /a objectSize = 1024
setlocal enabledelayedexpansion
for /l %%i in (0,1,19) do (
    set /a numClients +=1
    s3bench.exe ^
        -accessKey=hust ^
        -accessSecret=hust_obs ^
        -bucket=loadgen ^
        -endpoint=http://127.0.0.1:9000 ^
        -numClients=!numClients! ^
        -numSamples=256 ^
        -objectNamePrefix=loadgen ^
        -objectSize=!objectSize! >> result2.txt
        echo !numClients!)
```

4. 从测试结果文本中提取参数与关键数据，整理、保存至 Excel 文件中。
对象尺寸 objectSize 对存储系统性能影响测试数据如表 2 所示；
并发客户端数量 numClients 对存储系统性能影响测试数据如表 3 所示。

表 2 对象尺寸 objectSize 对存储系统性能影响

| 对象尺寸(MB) | 客户端数 | 写吞吐率(MB/s) | 写99th延迟(s) | 写90th延迟(s) | 读吞吐率(MB/s) | 读99th延迟(s) | 读90th延迟(s) |
|---|---|---|---|---|---|---|---|
| 0.0020 | 8 | 0.47 | 0.512 | 0.016 | 0.91 | 0.022 | 0.015 |
| 0.0039 | 8 | 1.65 | 0.034 | 0.019 | 1.89 | 0.019 | 0.014 |
| 0.0059 | 8 | 2.54 | 0.033 | 0.018 | 2.81 | 0.024 | 0.013 |
| 0.0078 | 8 | 3.08 | 0.033 | 0.022 | 1.93 | 0.513 | 0.019 |
| 0.0098 | 8 | 2.37 | 0.042 | 0.022 | 2.37 | 0.518 | 0.017 |
| 0.0117 | 8 | 2.61 | 0.510 | 0.022 | 2.89 | 0.514 | 0.016 |
| 0.0137 | 8 | 3.23 | 0.520 | 0.018 | 6.55 | 0.025 | 0.015 |
| 0.0156 | 8 | 3.69 | 0.519 | 0.018 | 7.31 | 0.020 | 0.015 |
| 0.0176 | 8 | 4.31 | 0.048 | 0.019 | 4.37 | 0.028 | 0.015 |
| 0.0195 | 8 | 8.55 | 0.037 | 0.017 | 9.46 | 0.025 | 0.014 |
| 0.0215 | 8 | 5.18 | 0.512 | 0.018 | 5.33 | 0.515 | 0.016 |
| 0.0234 | 8 | 5.63 | 0.043 | 0.022 | 5.76 | 0.512 | 0.016 |
| 0.0254 | 8 | 6.29 | 0.508 | 0.019 | 6.25 | 0.512 | 0.017 |
| 0.0273 | 8 | 6.72 | 0.043 | 0.021 | 6.72 | 0.025 | 0.017 |
| 0.0293 | 8 | 11.63 | 0.040 | 0.020 | 14.47 | 0.020 | 0.015 |
| 0.0312 | 8 | 7.66 | 0.035 | 0.019 | 7.65 | 0.024 | 0.015 |
| 0.0332 | 8 | 7.59 | 0.508 | 0.020 | 8.19 | 0.511 | 0.018 |
| 0.0352 | 8 | 8.04 | 0.078 | 0.030 | 8.64 | 0.519 | 0.017 |
| 0.0371 | 8 | 9.02 | 0.518 | 0.022 | 9.23 | 0.512 | 0.016 |
| 0.0391 | 8 | 9.31 | 0.106 | 0.025 | 9.56 | 0.518 | 0.019 |

表 3 并发客户端数量 numClients 对存储系统性能影响

| 对象尺寸(MB) | 客户端数 | 写吞吐率(MB/s) | 写99th延迟(s) | 写90th延迟(s) | 读吞吐率(MB/s) | 读99th延迟(s) | 读90th延迟(s) |
|---|---|---|---|---|---|---|---|
| 0.0010 | 1 | 0.15 | 0.011 | 0.008 | 0.20 | 0.009 | 0.007 |
| 0.0010 | 2 | 0.26 | 0.014 | 0.009 | 0.34 | 0.012 | 0.008 |
| 0.0010 | 3 | 0.30 | 0.024 | 0.012 | 0.35 | 0.018 | 0.011 |
| 0.0010 | 4 | 0.28 | 0.028 | 0.016 | 0.33 | 0.025 | 0.014 |
| 0.0010 | 5 | 0.31 | 0.028 | 0.019 | 0.31 | 0.083 | 0.018 |
| 0.0010 | 6 | 0.27 | 0.045 | 0.025 | 0.29 | 0.046 | 0.029 |
| 0.0010 | 7 | 0.31 | 0.067 | 0.025 | 0.24 | 0.033 | 0.020 |
| 0.0010 | 8 | 0.24 | 0.508 | 0.022 | 0.24 | 0.515 | 0.016 |
| 0.0010 | 9 | 0.22 | 0.518 | 0.022 | 0.24 | 0.525 | 0.018 |
| 0.0010 | 10 | 0.24 | 0.544 | 0.020 | 0.24 | 0.533 | 0.018 |
| 0.0010 | 11 | 0.22 | 0.530 | 0.026 | 0.24 | 1.034 | 0.017 |
| 0.0010 | 12 | 0.23 | 1.049 | 0.021 | 0.24 | 1.034 | 0.017 |
| 0.0010 | 13 | 0.23 | 0.556 | 0.023 | 0.24 | 1.038 | 0.016 |
| 0.0010 | 14 | 0.23 | 0.552 | 0.023 | 0.24 | 1.039 | 0.018 |
| 0.0010 | 15 | 0.23 | 0.548 | 0.034 | 0.24 | 1.048 | 0.017 |
| 0.0010 | 16 | 0.23 | 1.047 | 0.028 | 0.24 | 1.044 | 0.020 |
| 0.0010 | 17 | 0.22 | 0.537 | 0.065 | 0.43 | 0.530 | 0.018 |
| 0.0010 | 18 | 0.23 | 1.060 | 0.023 | 0.45 | 0.534 | 0.018 |
| 0.0010 | 19 | 0.23 | 1.027 | 0.038 | 0.45 | 0.542 | 0.017 |
| 0.0010 | 20 | 0.23 | 0.552 | 0.038 | 0.24 | 0.544 | 0.037 |

5. 将结果数据表绘制成折线图，观察存储性能随评测参数变化的规律并分析：

（1） 对象尺寸 objectSize 对存储系统性能影响

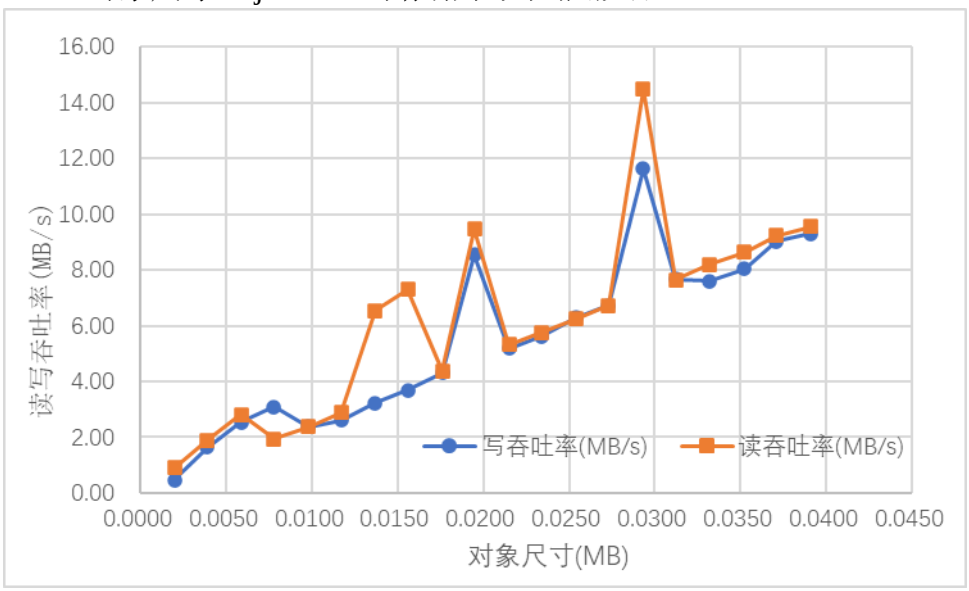


图 5 对象尺寸对读写吞吐率的影响

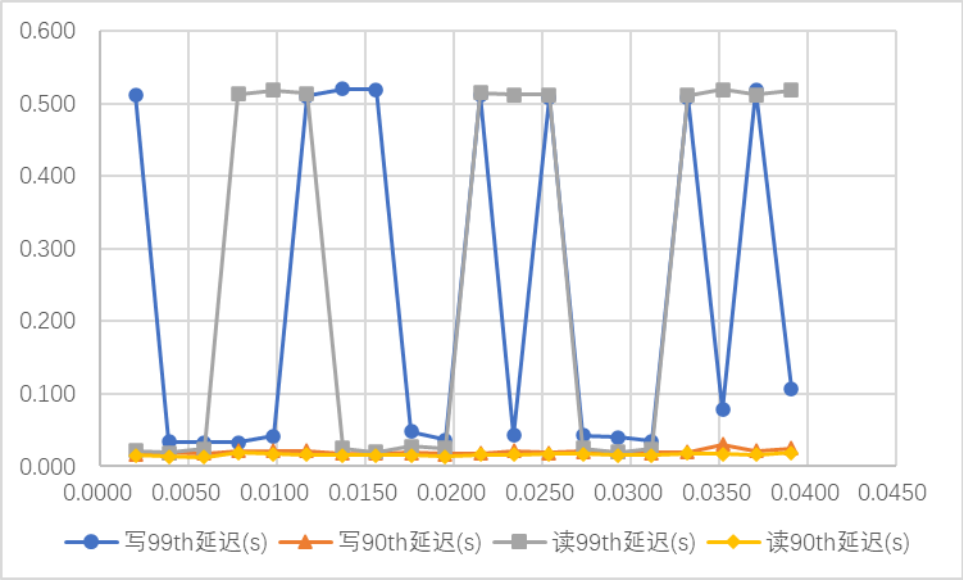


图 6 对象尺寸对读写延迟的影响

并发客户端数量 numClients = 8，样本数量 numSamples = 256 不变，使对象大小 objectSize 由 2048 字节开始梯度增加。

如图 5 所示，随着对象尺寸梯度增大，读写吞吐率 Throughput 均逐渐增大，且近似与对象尺寸呈现线性正相关关系。

如图 6 所示，对象尺寸梯度增大时，读写延迟的第 90 百分位数据（以下简称 90th 延迟）基本不变；读 99th 延迟在 20 轮测试中，在 0.03s 与 0.5s 两个水平附近波动，与对象尺寸增加的关系不明显；写 99th 延迟也在相同水平区域波动，但在对象大小较大（大于 0.03MB）时，其较低水平的波动区域呈现一定的上升趋势。可以认为，读延迟随对象尺寸增大的变化不明显；写延迟在对象尺寸较小时变化不明显，在对象尺寸较大时有一定程度的增大趋势。

（2） 并发客户端数量 numClients 对存储系统性能影响

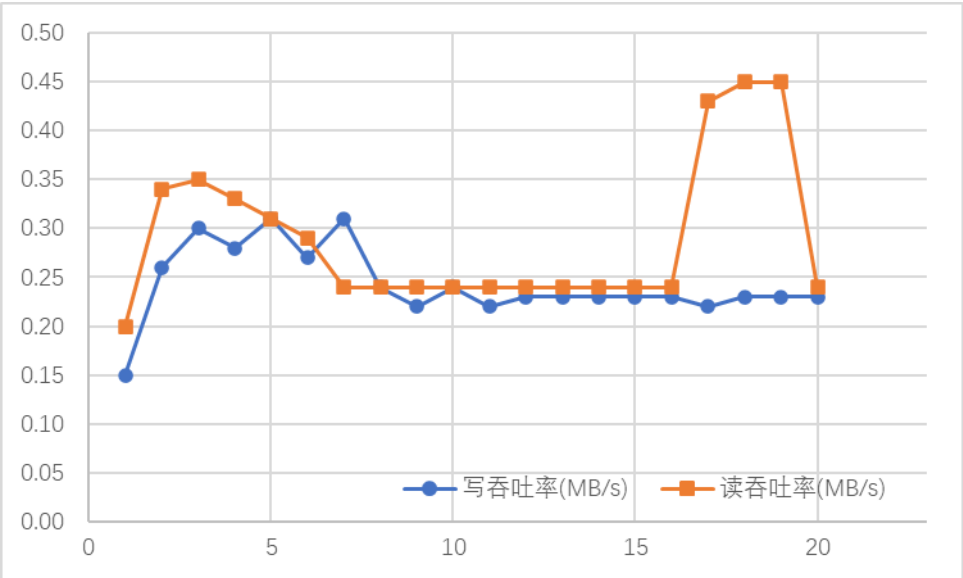


图 7 并发客户端数量对读写吞吐率的影响

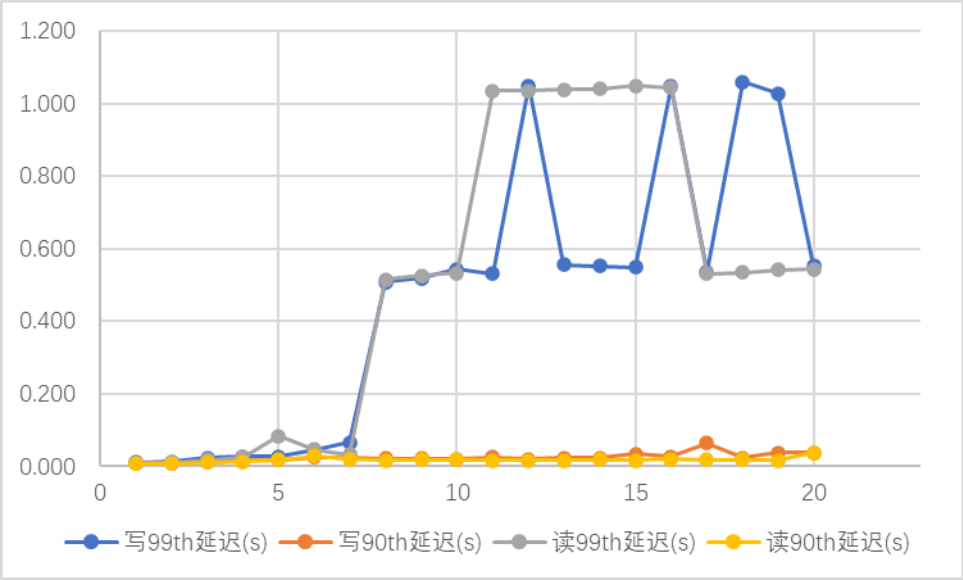


图 8 并发客户端数量对读写延迟的影响

对象尺寸 objectSize= 1024，样本数量 numSamples = 256 不变，使 numClients 由 1 个开始梯度增加。

如图 7 图 5 所示，当初始并发客户端数量较小（约 1~3 个）时，读写吞吐率因并发客户端数量增加而明显增大；随着客户端数量继续增大（4~9 范围内），读写吞吐率随之逐渐减小；最终（大于 10 个）逐渐趋于稳定水平。

如图 8 所示，并发客户端数量增大时，读写 90th 延迟基本不变；当客户端数量处于某一局部区间内时，读写 99th 延迟基本维持不变，但总体随客户端数量增加而呈现增大趋势；当客户端数量较大（大于 10 个）时，写 99th 延迟与写 90th 延迟的波动较为明显，表明并发数较大时，写延迟明显较大的对象增多。

根据上述两组测试结果与数据分析，考虑对象尺寸对存储性能的影响。对于熟悉的某类应用，可以根据其数据访问特性，选择响应的对象存储策略。例如，对于要求低延迟的应用（如搜索引擎），可以选择适配较小的 objectSize；对于要求高吞吐率的应用（如网盘），可以选择适配较大的 objectSize。

## 六、实验总结

本次物联网数据存储与管理实验的设计与安排，给了我许多新的学习体验。

实验内容方面，我在实验指导的帮助下，动手配置对象存储服务器、客户端，最终在本机上搭建出一个简易的对象存储系统，并利用工具对其存储性能进行针对性的评测。在此过程中，我对对象存储技术这一新型网络存储实用架构有了更进一步的理解，于课堂教学中了解到相关原理知识得以具化于实处。

任务设计方面，本次实验针对存储系统服务器、客户端、评测工具提供了多种选择，可以选择不同使用难度的软件平台，提高了实验的自由度，也让像我一样对相关知识了解较少的同学能够亲身参与实践。

此外，本次实验任务要求通过 Git 相关操作，在 Github 仓库中获取实验资源和提交实验成果。在此过程中，我学习和熟悉了 Git 常用命令与关键特性。

我相信本次实验中积累的经验、收获的知识，将对此后学习工作大有帮助。

# 参考文献

[1] ARNOLD J. OpenStack Swift[M]. O’Reilly Media, 2014.
[2] 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.
[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.
（可以根据实际需要更新调整）