2019 级

# 《物联网数据存储与管理》课程

# 实 验 报 告

姓　名　刘劲涛

学　号　U201915101

班　号　计算机 1906 班

日　期　2022.04.21

# 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，分析性能问题，架设应用实践。

# 二、实验背景

Minio 是一个基于 Go 语言的对象存储服务。它实现了大部分亚马逊 S3 云存储服务接口，可以看做是是 S3 的开源版本，非常适合于存储大容量非结构化的数据，例如图片、视频、日志文件、备份数据和容器/虚拟机镜像等，而一个对象文件可以是任意大小，从几 kb 到最大 5T 不等。区别于分布式存储系统，minio 的特色在于简单、轻量级，对开发者友好，认为存储应该是一个开发问题而不是一个运维问题。

Minio Client:mc 提供包括 ls、cat、cp、mirror、diff 等 UNIX 命令。它提供文件系统以及亚马逊 S3 兼容性云存储服务。

S3bench 提供了针对兼容 S3 的端点运行非常基本的吞吐量基准测试的功能。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。该工具使用 AWS Go SDK。

# 三、实验环境

| | |
|---|---|
| 操作系统 | Windows10 |
| 内存 | 1T |
| Python | 3 |
| CPU | Intel® Core™ i7-9750H CPU @ 2.60GHz |

# 四、实验内容

## 4.1 对象存储技术实践

1.配置 Minio 服务端。向配置好的客户端发送文件。

2.配置客户端。在远端利用客户端新建 bucket。

3.下载老师提供的 s3bench 脚本，修改相关参数进行相应的性能测试。

## 4.2 对象存储性能分析

1.对 s3bench 脚本输出的数据进行相应的处理。

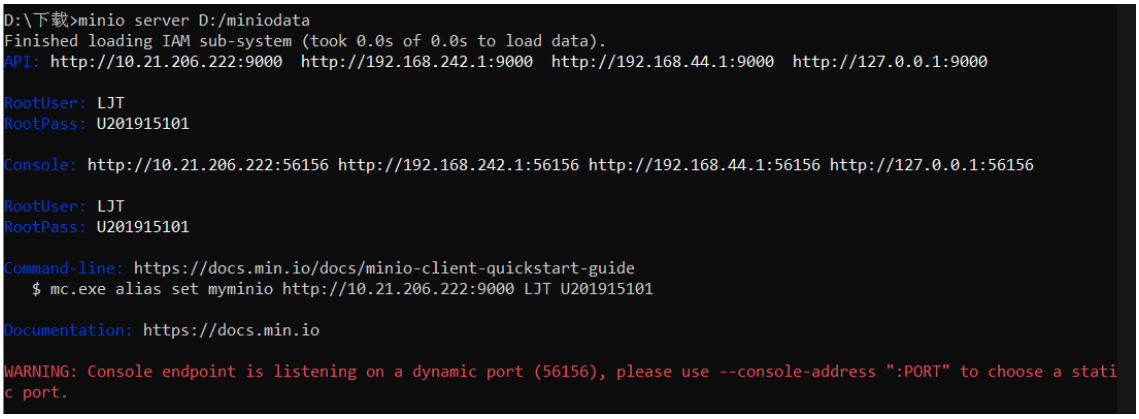
2.改变不同参数观察不同的性能指标。

3.对相应的实验数据做出记录。

## 五、实验过程

1.在 windows 系统下下载对应的 minio.exe 服务器。

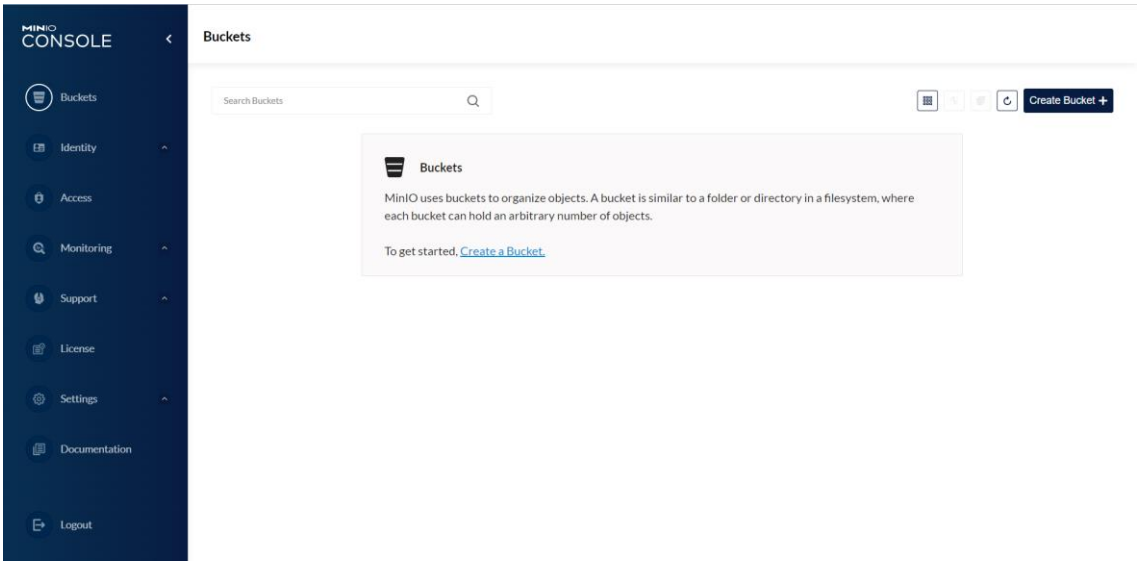
输入指令 minio server D:/miniodata。其中 D:/miniodata 表明 minio 服务端的 bucket 存储于这个文件夹中。运行结果如图 5-1。




5-1

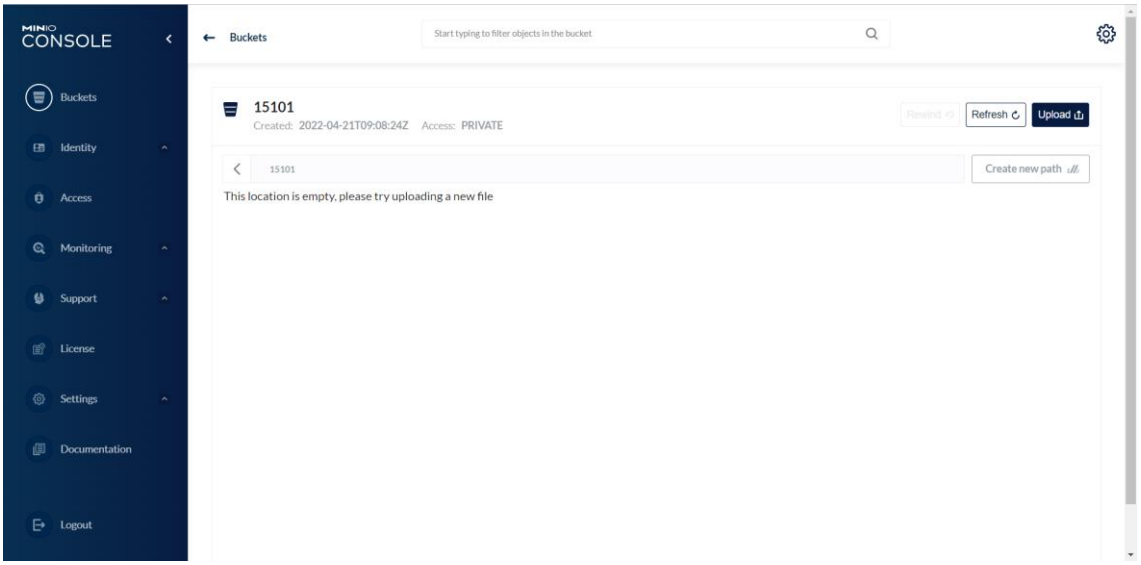
得到对应的服务端的调用的 URL 为本机网络 IP 的 9000 端口，而我们的服务端控制台为本地网络 IP 的 56156 端口。同时我们可以看出我们的 RootUser 和 RootPass 分别为我名字的缩写 LJT 和学号 U201915101。这是因为我使用指令 set MINIO_ACCESS_KEY=LJT，set MINIO_SECRET_KEY=U201915101 所致。

之后打开我们此界面里的 URL http://10.21.206.222:56156，出现如图 3-2 所示界面。




3-2

点击右上角的按钮 Create Bucket，来创建第一个篮子 bucket。我给他命名为 15101。创建完后的界面如图 3-3。




3-3

之后我们向这个 bucket 上传一个文件用以测试。该操作将在测试环节进行相应的描述。

2.在 windows 系统下下载相应的 mc.exe，进行一个 minio 客户端的相关操作。

首先是打开 minio 客户端，输入指令 mc config host add iot http://127.0.0.1:9000 LJT U201915101 --api s3v4，添加一个名为 iot 的存储服务。如图 3-4 所示，服务添加成功。

```
D:\下载>mc config host add iot http://127.0.0.1:9000 LJT U201915101 --api s3v4
Added `iot` successfully.
```

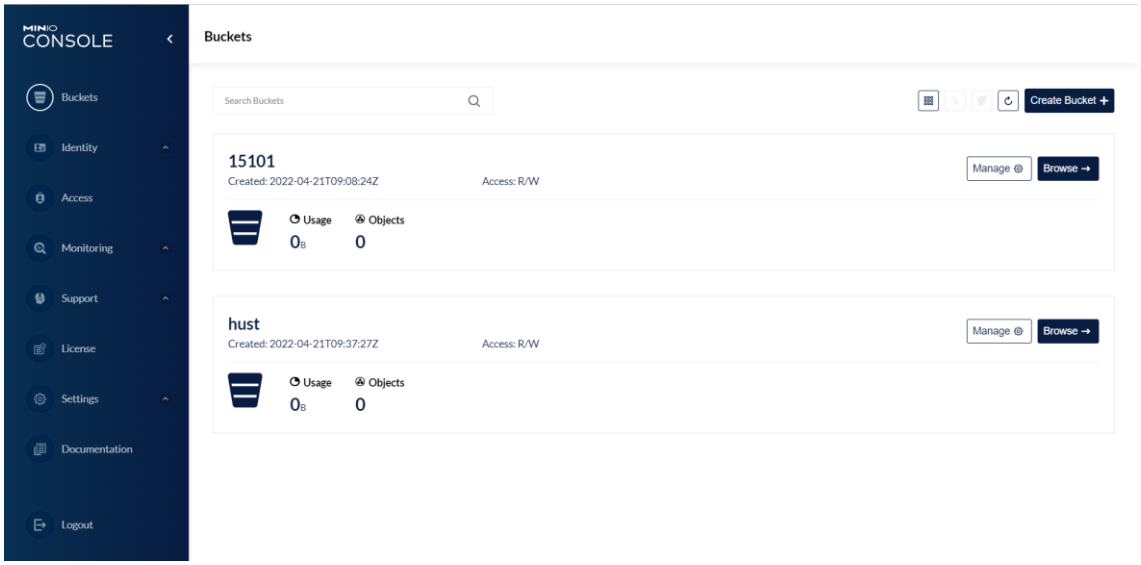
3-4

之后通过 mc nb iot/hust 指令在客户端向服务端添加一个名为 hust 的 bucket。如图 3-5 所示。指令 mc ls iot 是为观测现在的服务端有多少个 bucket。

```
D:\下载>mc ls iot
[2022-04-21 17:08:24 CST]     0B 15101/

D:\下载>mc ls iot /hust
[2022-04-21 17:08:24 CST]     0B 15101/

D:\下载>mc mb iot/hust
Bucket created successfully `iot/hust`.

D:\下载>mc ls iot
[2022-04-21 17:08:24 CST]     0B 15101/
[2022-04-21 17:37:27 CST]     0B hust/
```

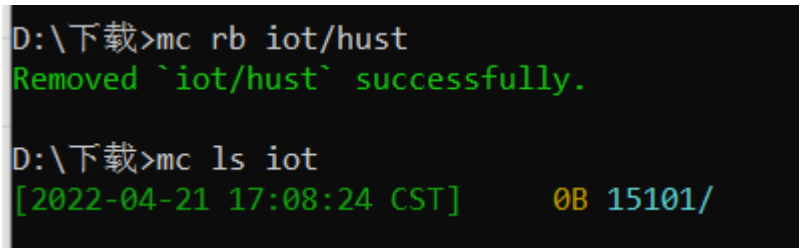
3-5

此时再看看服务端有无相应的 bucket。如图 3-6。



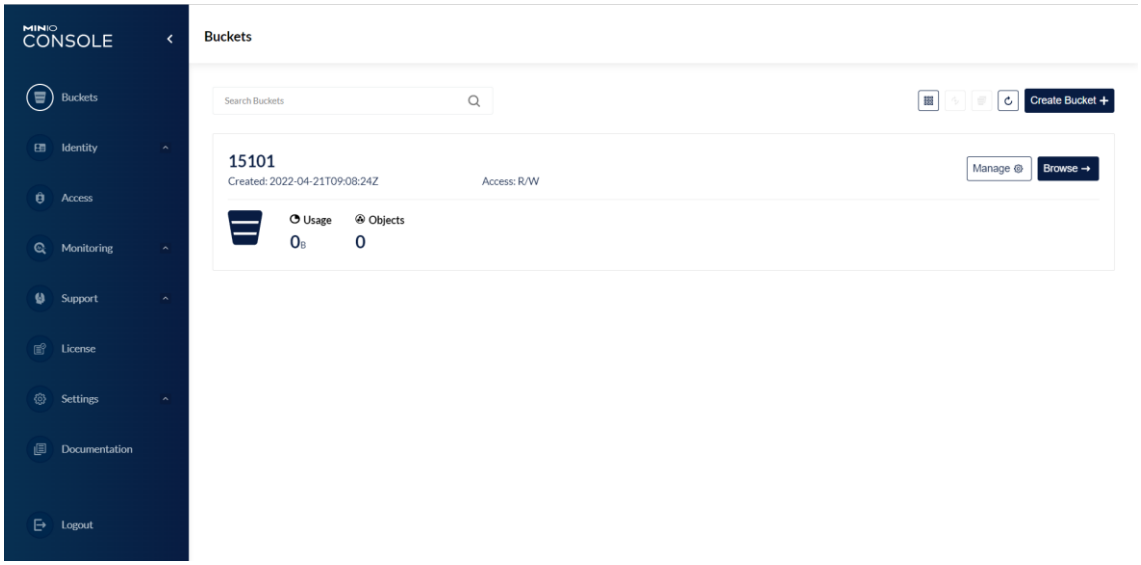

3-6

可以看到现在的 buckets 有两个了。之后通过 mc rb iot/hust 来删除相应的 bucket。如图 3-7。可以看出删除操作成功。




3-7

再看服务端界面，刷新，也没有对应的 bucket。如图 3-8。




3-8

3.利用 s3bench 脚本对对象的存储性能进行相应的分析。

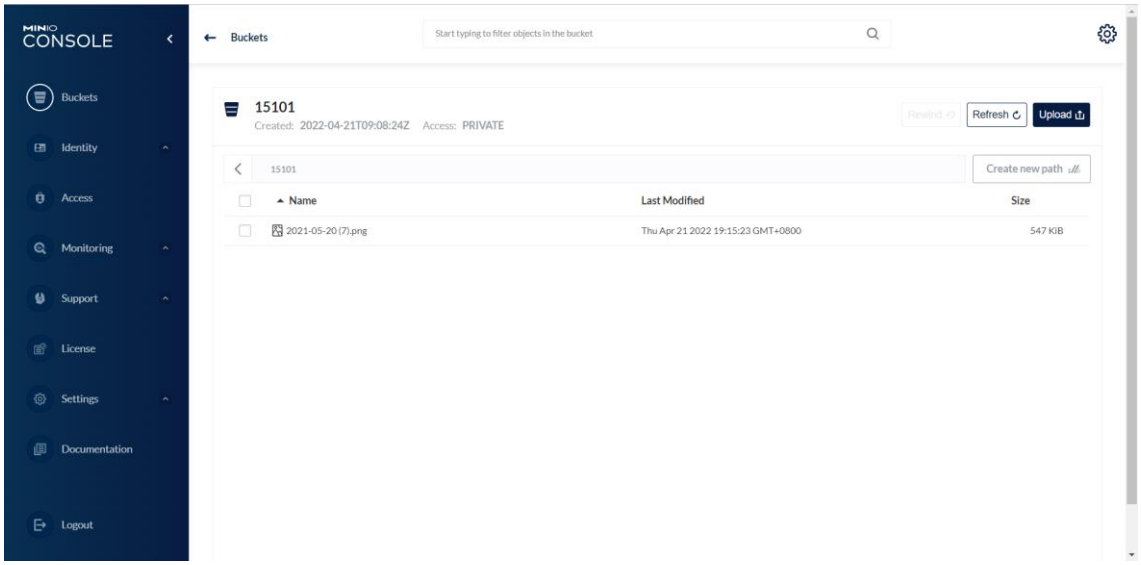
首先下载老师提供的相关测试工具包，之后修改 run-s3bench.cmd 这个文件。

把 accesskey 改为我的账号名，accessSecret 改为学号，bucket 改为相应的 15101。之后运行脚本就可以进行相应的性能测试操作了。修改文件如图 3-9。

```
s3bench.exe ^
    -accessKey=LJT^
    -accessSecret=U201915101^
    -bucket=15101^
    -endpoint=http://127.0.0.1:9000 ^
    -numClients=8 ^
    -numSamples=256 ^
    -objectNamePrefix=loadgen ^
    -objectSize=1024
pause
```

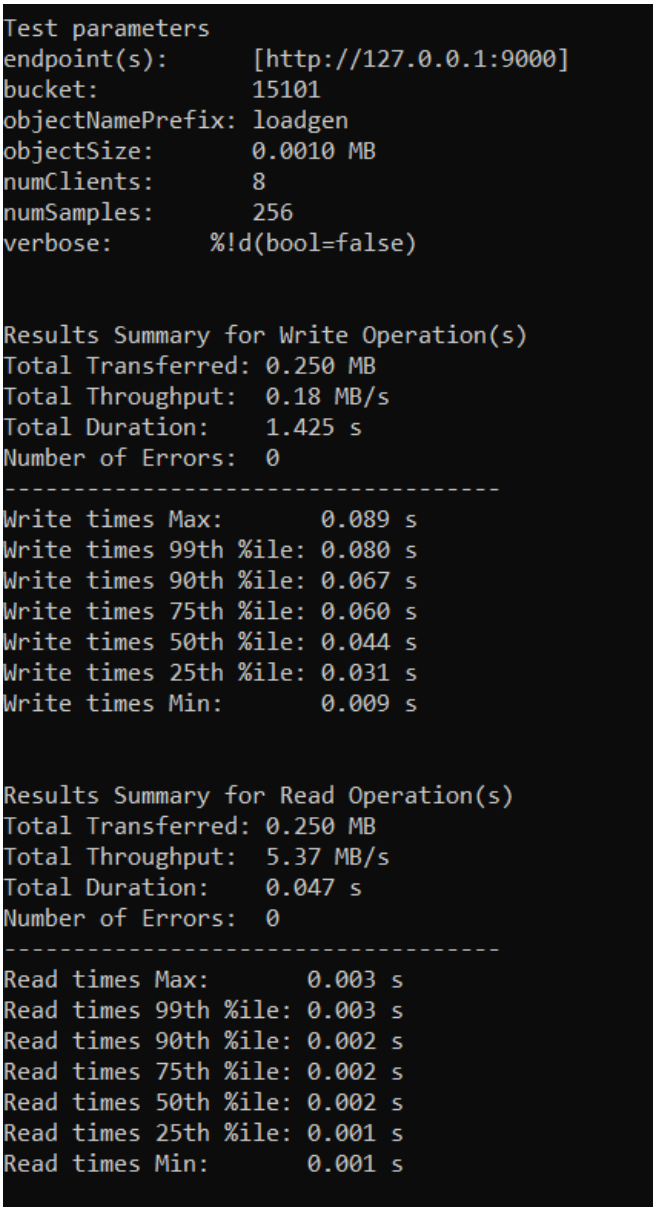
3-9

之后向 15101 上传一个 png 文件。并且进行 s3bench 的测速。文件上传如图 3-10。




3-10

S3bench 测试如图 3-11。分析相关数据，发现服务器的读取速率远大于写入速率。




3-11

这时我们改变不同的 numClients，numSamples，objectSize 的值，分别对于 numClients 取 8，16，24，numSamples 取 128，256，512，objectSize 分别取 512，1024， 2048 得到总共 27 组数据如下表。

| numClients | numSamples | objectSize | write | | | | read | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | throughput | 90% | 99% | max | throughput | 90% | 99% | max |
| 8 | 128 | 512 | 0.09 | 0.075 | 0.098 | 0.104 | 2.39 | 0.003 | 0.004 | 0.004 |
| 8 | 128 | 1024 | 0.14 | 0.08 | 0.109 | 0.111 | 5.02 | 0.002 | 0.003 | 0.003 |
| 8 | 128 | 2048 | 0.28 | 0.091 | 0.108 | 0.109 | 8.62 | 0.002 | 0.003 | 0.003 |
| 8 | 256 | 512 | 0.08 | 0.075 | 0.104 | 0.132 | 2.6 | 0.002 | 0.004 | 0.005 |
| 8 | 256 | 1024 | 0.18 | 0.067 | 0.08 | 0.089 | 5.37 | 0.002 | 0.003 | 0.003 |
| 8 | 256 | 2048 | 0.33 | 0.077 | 0.096 | 0.097 | 11.29 | 0.002 | 0.003 | 0.003 |
| 8 | 512 | 512 | 0.08 | 0.075 | 0.103 | 0.115 | 2.61 | 0.002 | 0.003 | 0.004 |
| 8 | 512 | 1024 | 0.17 | 0.067 | 0.093 | 0.096 | 5.15 | 0.002 | 0.003 | 0.004 |
| 8 | 512 | 2048 | 0.31 | 0.077 | 0.091 | 0.096 | 9.98 | 0.002 | 0.004 | 0.005 |
| 16 | 128 | 512 | 0.07 | 0.142 | 0.168 | 0.171 | 2.69 | 0.004 | 0.005 | 0.007 |
| 16 | 128 | 1024 | 0.12 | 0.158 | 0.182 | 0.187 | 5.17 | 0.004 | 0.006 | 0.006 |
| 16 | 128 | 2048 | 0.31 | 0.136 | 0.143 | 0.144 | 11.11 | 0.004 | 0.005 | 0.006 |
| 16 | 256 | 512 | 0.07 | 0.146 | 0.193 | 0.207 | 2.26 | 0.005 | 0.006 | 0.007 |
| 16 | 256 | 1024 | 0.14 | 0.141 | 0.155 | 0.179 | 5.51 | 0.004 | 0.005 | 0.006 |
| 16 | 256 | 2048 | 0.29 | 0.143 | 0.168 | 0.182 | 7.69 | 0.006 | 0.008 | 0.009 |
| 16 | 512 | 512 | 0.07 | 0.152 | 0.174 | 0.178 | 2.7 | 0.004 | 0.006 | 0.007 |
| 16 | 512 | 1024 | 0.14 | 0.144 | 0.169 | 0.192 | 5.12 | 0.004 | 0.006 | 0.007 |
| 16 | 512 | 2048 | 0.3 | 0.136 | 0.154 | 0.178 | 9.73 | 0.005 | 0.006 | 0.007 |
| 24 | 128 | 512 | 0.06 | 0.238 | 0.253 | 0.253 | 2.51 | 0.009 | 0.013 | 0.013 |
| 24 | 128 | 1024 | 0.13 | 0.209 | 0.229 | 0.244 | 4.33 | 0.009 | 0.015 | 0.016 |
| 24 | 128 | 2048 | 0.28 | 0.201 | 0.227 | 0.228 | 7.52 | 0.01 | 0.015 | 0.016 |
| 24 | 256 | 512 | 0.07 | 0.234 | 0.283 | 0.295 | 3.06 | 0.006 | 0.008 | 0.009 |
| 24 | 256 | 1024 | 0.14 | 0.203 | 0.304 | 0.314 | 6.11 | 0.006 | 0.009 | 0.012 |
| 24 | 256 | 2048 | 0.26 | 0.23 | 0.261 | 0.275 | 11.91 | 0.006 | 0.009 | 0.011 |
| 24 | 512 | 512 | 0.07 | 0.215 | 0.245 | 0.257 | 2.82 | 0.006 | 0.011 | 0.015 |
| 24 | 512 | 1024 | 0.13 | 0.226 | 0.26 | 0.261 | 5.59 | 0.006 | 0.011 | 0.015 |
| 24 | 512 | 2048 | 0.27 | 0.22 | 0.259 | 0.29 | 11.54 | 0.006 | 0.01 | 0.012 |

通过表中的数据进行相关分析，得出以下结论：

（1） 具体来看，随着 Clients 并发数的提升，write 和 read 吞吐率基本不变。Write 和 read 的时间都会有一定的增长。

（2） 随着 Samples 样本数的提升，write 和 read 吞吐率基本不变。而两者时间也会延长。

（3） 随着 object size 文件大小的提升，吞吐率逐渐上升，且上升幅度比 Clients 并发数大。

## 六、实验总结

这次试验让我熟悉了 minio 和 mc 的基本使用方法，也让我懂得了面向对象存储的一些知识。刚开始我妄图想在 linux 系统下完成本次实验，但是很久没有维护的 cosbench 给我下了很大的难题，而且对于 s3bench 的使用一开始我屡屡碰壁。在 linux 系统下我想通过搭建 go 语言环境去实现他的运转。可后来在同学的帮助下我了解到其实老师给的资源中就有相关的脚本。总的来说，这次试验真是非常非常有意义。我还学习到了一些相关的分布式 git 的相关操作。

# 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.