



2019 级

《物联网数据存储与管理》课程

实 验 报 告

姓 名 杨超淇

学 号 U201914974

班 号 计算机 1903 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.....	2
4.1 对象存储技术实践.....	2
4.2 对象存储性能分析.....	2
五、实验过程.....	2
5.1 minio 与 minio client	2
5.2 mock-s3 与 s3cmd	4
5.3 s3bench 性能测试	5
六、实验总结.....	9
参考文献.....	10

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，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对象。和文件存储不同的是，对象在一个层结构中不会再有层级结构。每个对象都在一个被称作存储池的扁平地址空间的同一级别里，一个对象不会属于另一个对象的下一级。

文件和对象都有与它们所包含的数据相关的元数据，但是对象是以扩展元数据为特征的。每个对象都被分配一个唯一的标识符，允许一个服务器或者最终用户来检索对象，而不必知道数据的物理地址。这种方法对于在云计算环境中自动化和简化数据存储有帮助。

而基于这一技术产生的对象存储系统是综合了 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。

MinIO 是一款高性能、分布式的对象存储系统，适合初学者了解对象存储系统的应用。MinIO 一开始就只为对象存储而设计，所以它采用了更易用的方式进行设计，它能实现对象存储所需要的全部功能，在性能上也更加强劲，它不会为了更多的业务功能而妥协，失去 MinIO 的易用性、高效性。

mock-s3 和 s3cmd 都源自于 Amazon S3(由亚马逊 AWS 提供的简单存储服务)。mock-s3 重写了 fake-s3 实现了对 Amazon S3 的轻量级模仿，可以让使用者在本地创建一个简单的对象存储服务器。s3cmd 是 Amazon S3 的兼容客户端工具，可以通过命令行的方式对 S3 服务器发送 HTTP 请求。

s3bench 是使用了 AWS Go SDK 实现的对象存储服务器测试程序，可以通过设置请求对象大小和数量、并行客户端数量等参数，根据吞吐率、延迟等结果评测对象存储系统的性能。

三、实验环境

本次对象存储测试实验通过虚拟机在 Linux 系统中完成，具体的实验环境以及所用到的工具如表 1 所示。

表 1 实验环境

操作系统	ubuntu-20.04.3
虚拟机软件	VMware Workstation 16 Pro
CPU	Intel® Core™ i5-9300H CPU @ 2.40GHz
分配内存	4GB
程序语言环境	python-3.8.4、go-1.13
服务端	minio、mock-s3
客户端	minio client、s3cmd
测试程序	s3bench
服务器 IP	localhost:9000

四、实验内容

本次实验的主要内容为使用已实现的对象存储系统程序，了解对象存储技术的原理以及实际应用，并对不同情况下的对象存储系统进行性能分析，研究这项技术的优势与缺陷。

4.1 对象存储技术实践

- (1) 在 Linux 虚拟机中配置实验所需的 python 与 go 环境。
- (2) 通过 minio 官网教程获取 minio 与 minio client 可执行程序。
- (3) 使用 minio 创建服务器，然后通过 minio client 向服务器发送请求，通过服务器网页查看请求是否成功。
- (4) 通过 github 以及 python pip 分别获取 mock-s3 和 s3cmd 程序，并用 s3cmd 连接到 mock-s3 服务器，测试对象存储基本指令。

4.2 对象存储性能分析

- (1) 使用 go get 指令从 github 中获取并自动安装 s3bench。
- (2) 使用实验指导中的 run-s3bench.sh 对 minio 服务器进行样例测试，测试 minio 服务器的性能。
- (3) 修改 run-s3bench.sh，使其在多次循环中改变特定参数，并对 mock-s3 进行性能测试。
- (4) 收集性能测试结果数据，通过图表分析对象存储系统的性能影响因素。

五、实验过程

5.1 minio 与 minio client

- (1) 根据 minio 官方文档给出的引导，在 Linux 终端输入如下命令。wget 从官网获取程序文件，chmod 赋予 minio 程序执行权限。设置 minio 服务器用户名与密码、服务器在本地的根目录、minio 控制台页面端口，启动 minio 程序创建服务器。

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
chmod +x minio
MINIO_ROOT_USER=U20194974 MINIO_ROOT_PASSWORD=password
./minio server ./data --console-address ":9001"

root@ubuntu:/home/ycq/bigdata-storage# MINIO_ROOT_USER=U20194974 MINIO_ROOT_PASSWORD=password
./minio server ./data --console-address ":9001"
Finished loading IAM sub-system (took 0.0s of 0.0s to load data).
API: http://192.168.100.134:9000 http://127.0.0.1:9000
RootUser: U20194974
RootPass: password

Console: http://192.168.100.134:9001 http://127.0.0.1:9001
RootUser: U20194974
RootPass: password

Command-line: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc alias set myminio http://192.168.100.134:9000 U20194974 password

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

图 1 创建 minio 服务器

- (2) 根据图 1，服务器 IP 为 127.0.0.1（本地 IP），服务器端口号为 9000，控制台端口号为 9001。使用浏览器打开控制台，并测试创建一个新的 bucket：loadgen，如图 2 所示。

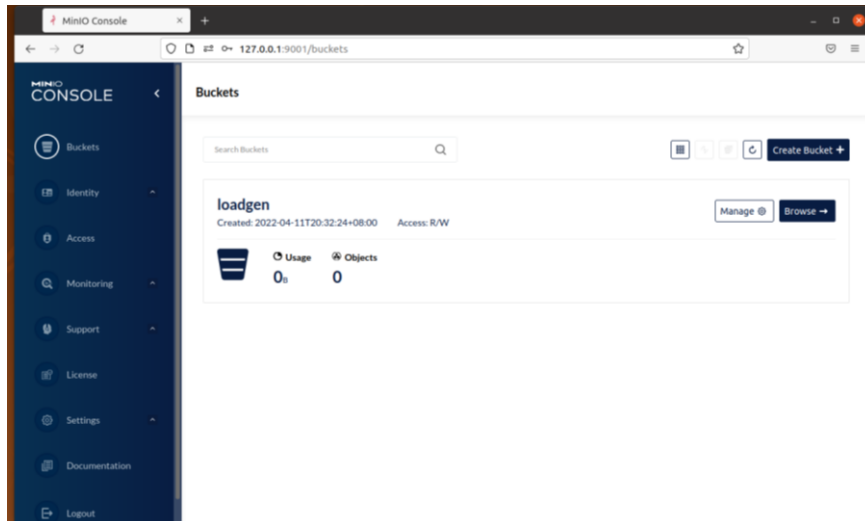


图 2 测试 minio 控制台页面

- (3) 通过 `wget` 命令获取 minio client 可执行程序, 并使用 `chmod` 命令赋予 minio client 执行权限。

```
wget https://dl.min.io/client/mc/release/linux-amd64/mc
chmod +x mc
```

- (4) 运行 `mc` 尝试 minio 基本指令, 结果如图 3 所示

`alias set` 指令为 minio 服务器地址 `192.168.100.134:9000` 定义一个别名 `myminio`, 便于后续调用服务器, 同时输入预设的用户名与密码登录服务器。

`mb` 指令用于创建 bucket。

`ls` 指令显示出服务器目录下所有的 bucket 和 object。

```
root@ubuntu:/home/ycq/bigdata-storage# ./mc alias set myminio http://192.168.100.134:9000 U201914974 password
Added 'myminio' successfully.
root@ubuntu:/home/ycq/bigdata-storage# ./mc mb myminio/test
Bucket created successfully 'myminio/test'.
root@ubuntu:/home/ycq/bigdata-storage# ./mc ls myminio
[2022-04-11 20:32:24 CST]    0B loadgen/
[2022-04-13 20:46:46 CST]    0B test/
```

图 3 测试 minio 基本指令

- (5) 再次打开 minio 控制台, 查看 `mc` 是否成功向服务器发送请求。

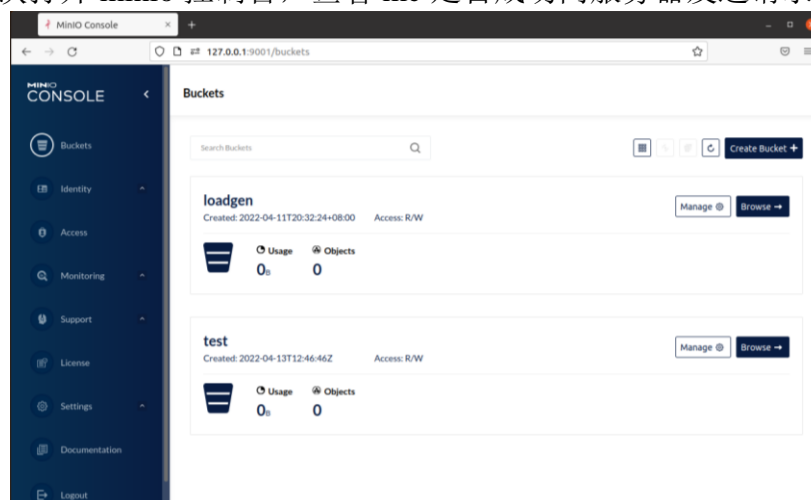


图 4 查看 mc 指令执行结果

5.2 mock-s3 与 s3cmd

- (1) 从 github 中获取 mock-s3 的源代码文件夹，在目录 mock-s3 下执行如下命令创建服务器，服务器根目录为 data。

通过浏览器输入服务器地址：127.0.0.1:9000，打开服务器页面，如图 5 所示。

```
python3 main.py --hostname 0.0.0.0 --port 9000 --root ./data
```



图 5 查看服务器页面

- (2) 通过 python pip 安装 s3cmd 客户端程序，s3cmd 程序运行需要.s3cfg 配置文件。在 root 文件夹中创建.s3cfg 文件，具体配置如图 6 所示。

其中 host 地址为服务器所在 ip 地址（127.0.0.1:9000），由于 mock-s3 使用 HTTP 协议，所以要关闭使用 HTTPS 协议。

```
1 # Setup endpoint
2 host_base = localhost:9000
3 host_bucket = localhost:9000
4 bucket_location = us-east-1
5 use_https = False
6
7 # Setup access keys
8 access_key = Q3AM3UQ867SPQQA43P2F
9 secret_key = zuf+tfteSlswRu7BJ86wekitnifILbZam1KYY3TG
10
11 # Enable S3 v4 signature APIs
12 signature_v2 = False
```

图 6 客户端 s3cmd 配置文件

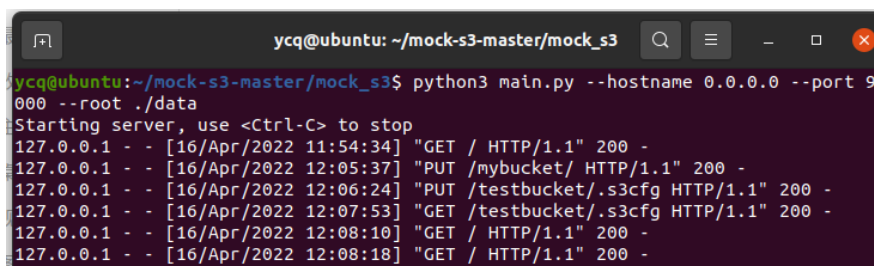
- (3) 由于配置文件中已经设定了服务器地址、用户名、密码等参数，运行 s3cmd 时不需要像 mc 那样输入。尝试使用 s3cmd 指令，如图 7 所示。

mb 指令创建 mybucket，put 指令将本地文件作为对象传递给服务器，get 指令从服务器中获取对象，ls 指令展示服务器目录项。

```
root@ubuntu:/home/ycq/.local/bin# ./s3cmd mb s3://mybucket
Bucket 's3://mybucket/' created
root@ubuntu:/home/ycq/.local/bin# ./s3cmd put .s3cfg s3://testbucket
upload: '.s3cfg' -> 's3://testbucket/.s3cfg' [1 of 1]
280 of 280 100% in 0s 63.13 KB/s done
root@ubuntu:/home/ycq/.local/bin# rm .s3cfg
root@ubuntu:/home/ycq/.local/bin# ./s3cmd get s3://testbucket/.s3cfg
download: 's3://testbucket/.s3cfg' -> './.s3cfg' [1 of 1]
280 of 280 100% in 0s 15.30 KB/s done
root@ubuntu:/home/ycq/.local/bin# ./s3cmd ls s3://
2022-04-16 12:05 s3://mybucket
2022-04-16 12:06 s3://testbucket
```

图 7 尝试 s3cmd 指令

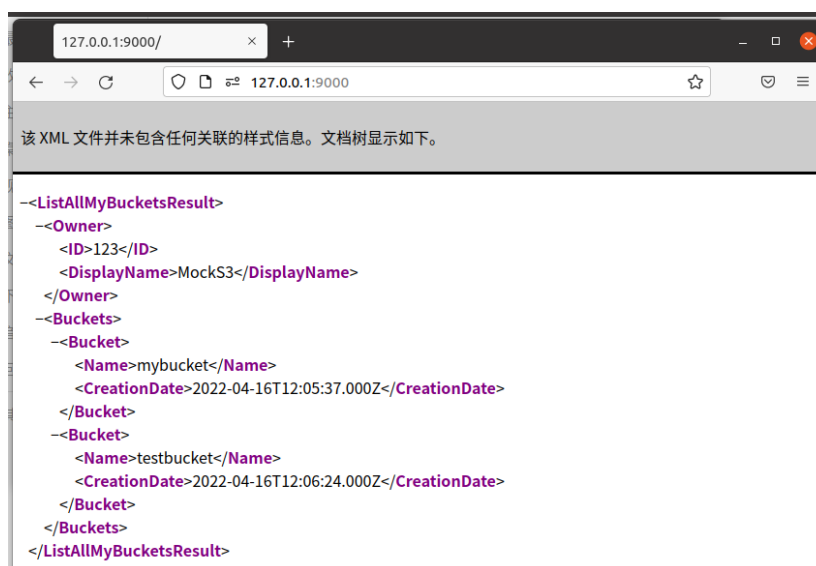
- (4) 查看服务器端，mock-s3 服务端程序会将收到的 HTTP 请求逐行显示在终端内。如图 8 所示，尝试 s3cmd 指令操作产生的 HTTP 请求均被 mock-s3 接受，并作出回应。

A terminal window titled 'ycq@ubuntu: ~/mock-s3-master/mock_s3'. The command 'python3 main.py --hostname 0.0.0.0 --port 9000 --root ./data' has been executed. The output shows the server starting and then logging several HTTP requests: GET / HTTP/1.1, PUT /mybucket/ HTTP/1.1, PUT /testbucket/.s3cfg HTTP/1.1, and GET /testbucket/.s3cfg HTTP/1.1, all returning 200 status codes.

```
ycq@ubuntu: ~/mock-s3-master/mock_s3$ python3 main.py --hostname 0.0.0.0 --port 9000 --root ./data
Starting server, use <Ctrl-C> to stop
127.0.0.1 - - [16/Apr/2022 11:54:34] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2022 12:05:37] "PUT /mybucket/ HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2022 12:06:24] "PUT /testbucket/.s3cfg HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2022 12:07:53] "GET /testbucket/.s3cfg HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2022 12:08:10] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [16/Apr/2022 12:08:18] "GET / HTTP/1.1" 200 -
```

图 8 服务器收到 HTTP 请求

- (5) 再次打开服务器页面，查看服务器内的 buckets。

A web browser window showing the response from 127.0.0.1:9000/. The response is an XML document. The text above the XML says '该 XML 文件并未包含任何关联的样式信息。文档树显示如下。'. The XML structure is as follows:

```
--<ListAllMyBucketsResult>
  --<Owner>
    <ID>123</ID>
    <DisplayName>MockS3</DisplayName>
  </Owner>
  --<Buckets>
    --<Bucket>
      <Name>mybucket</Name>
      <CreationDate>2022-04-16T12:05:37.000Z</CreationDate>
    </Bucket>
    --<Bucket>
      <Name>testbucket</Name>
      <CreationDate>2022-04-16T12:06:24.000Z</CreationDate>
    </Bucket>
  </Buckets>
</ListAllMyBucketsResult>
```

图 9 查看服务器页面

5.3 s3bench 性能测试

执行 `go get -u github.com/igneous-systems/s3bench` 命令，从 github 获取并安装 s3bench。

s3bench 样例测试：

运行 minio 服务端，创建对象存储服务器。

下载实验指导中的 `run-s3bench.sh` 文件，修改参数如图 10 所示。其中用户名、密码、ip 与 minio 服务器一致，设置测试用的并行客户端数量为 8 个，数据对象大小为 32KB，对象数量为 256，使用 minio 服务器中的 bucket: loadgen 进行实验。

```
23 $s3bench \
24   -accessKey=U201914974 \
25   -accessSecret=password \
26   -bucket=loadgen \
27   -endpoint=http://127.0.0.1:9000 \
28   -numClients=8 \
29   -numSamples=256 \
30   -objectNamePrefix=loadgen \
31   -objectSize=$(( 1024*32 ))
```

图 10 s3bench 参数配置

s3bench 样例测试实验结果如图 11 所示。

根据 s3bench 的输出，可以在固定的并行客户端数量、对象大小与数量等参数的影响下，对对象存储系统的具体性能（通过服务器吞吐率、百分位延迟等信息体现）进行分析。

```
Test parameters
endpoint(s): [http://127.0.0.1:9000]
bucket: loadgen
objectNamePrefix: loadgen
objectSize: 0.0312 MB
numClients: 8
numSamples: 256
verbose: %!d(bool=false)

Results Summary for Write Operation(s)
Total Transferred: 8.000 MB
Total Throughput: 31.03 MB/s
Total Duration: 0.258 s
Number of Errors: 0
-----
Write times Max: 0.024 s
Write times 99th %ile: 0.019 s
Write times 90th %ile: 0.014 s
Write times 75th %ile: 0.011 s
Write times 50th %ile: 0.007 s
Write times 25th %ile: 0.004 s
Write times Min: 0.002 s

Results Summary for Read Operation(s)
Total Transferred: 8.000 MB
Total Throughput: 87.42 MB/s
Total Duration: 0.092 s
Number of Errors: 0
-----
Read times Max: 0.018 s
Read times 99th %ile: 0.015 s
Read times 90th %ile: 0.006 s
Read times 75th %ile: 0.004 s
Read times 50th %ile: 0.002 s
Read times 25th %ile: 0.001 s
Read times Min: 0.001 s

Cleaning up 256 objects...
Deleting a batch of 256 objects in range {0, 255}... Succeeded
Successfully deleted 256/256 objects in 57.532118ms
```

图 11 样例测试结果

s3bench 对象大小的影响测试：

运行 mock-s3 服务端程序，创建服务器。

修改 run-s3bench.sh 文件，将原先单次运行 s3bench 的代码，改为循环 10 次运行，并在每次运行时修改对象大小（从 1024*100B 到 1024*1000B）。

记录每次循环时服务器的读/写吞吐率，以及服务器读/写到 99% 与 90% 时的延迟，得到图 12 的图表。

对象大小(MB)	客户端数量	对象数量	写吞吐率	写99%延迟	写90%延迟	读吞吐率	读99%延迟	读90%延迟
0.0977	8	256	23.94	0.019	0.012	20.00	0.207	0.011
0.1953	8	256	44.00	0.019	0.016	48.78	0.017	0.012
0.2930	8	256	68.05	0.024	0.017	73.22	0.015	0.012
0.3906	8	256	96.85	0.025	0.020	92.04	0.020	0.013
0.4883	8	256	102.17	0.031	0.022	114.98	0.018	0.013
0.5859	8	256	140.50	0.031	0.024	138.34	0.018	0.014
0.6836	8	256	99.52	0.034	0.026	156.42	0.019	0.014
0.7812	8	256	168.90	0.036	0.029	156.63	0.022	0.015
0.8789	8	256	197.94	0.039	0.028	219.76	0.019	0.015
0.9766	8	256	197.12	0.048	0.031	244.00	0.019	0.015

图 12 对象大小变化影响记录

将吞吐率随对象大小的变化制成图 13 的折线图。可以观察到，随着对象大小的线性增长，读/写吞吐率也近乎成线性增长。

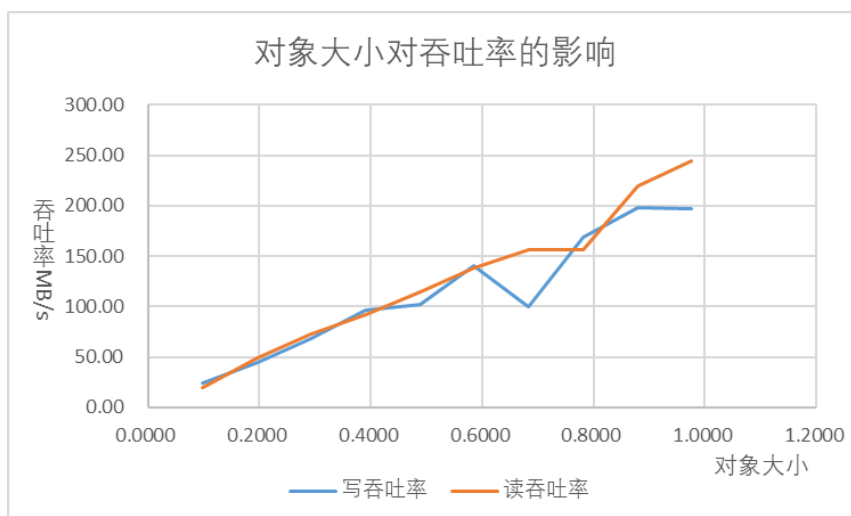


图 13 对象大小对吞吐率的影响

将百分位延迟随对象大小的变化制成图 14 的折线图。可以观察到，随着对象大小的增长，延迟有稍微的增长，且 99%时的延迟的增长幅度明显大于 90%时的延迟的增长幅度。

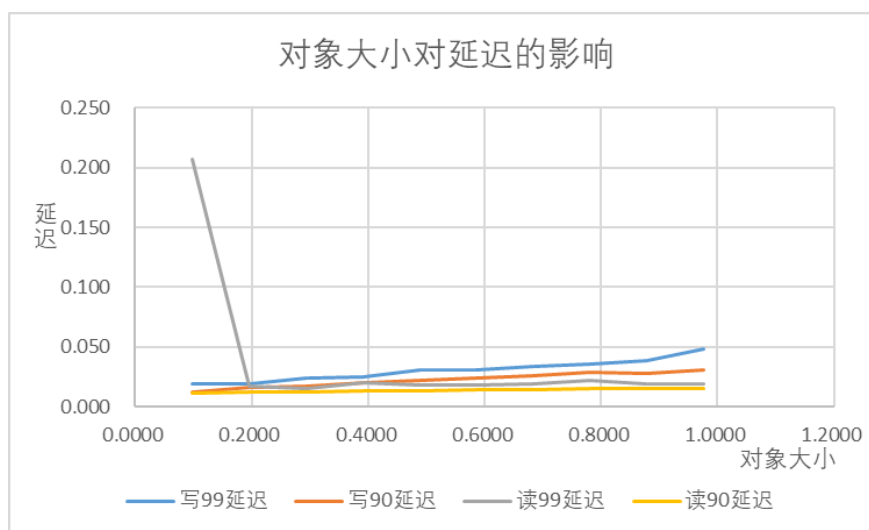


图 14 对象大小对延迟的影响

s3bench 客户端数量的影响测试：

修改 run-s3bench.sh 文件，将原先单次运行 s3bench 的代码，改为循环 10 次运行，并在每次运行时修改客户端数量（从 1 到 10）。

记录每次循环时服务器的读/写吞吐率，以及服务器读/写到 99%与 90%时的延迟，得到图 15 的图表。

对象大小(MB)	客户端数量	对象数量	写吞吐率	写99%延迟	写90%延迟	读吞吐率	读99%延迟	读90%延迟
0.0977	1	256	51.19	0.005	0.002	63.97	0.004	0.002
0.0977	2	256	58.56	0.007	0.005	66.40	0.006	0.004
0.0977	3	256	56.30	0.019	0.008	68.67	0.009	0.006
0.0977	4	256	60.61	0.012	0.008	69.96	0.011	0.007
0.0977	5	256	56.87	0.027	0.012	80.03	0.009	0.008
0.0977	6	256	56.94	0.046	0.013	78.52	0.016	0.010
0.0977	7	256	24.68	0.019	0.014	56.20	0.024	0.015
0.0977	8	256	23.35	0.048	0.019	24.12	0.019	0.011
0.0977	9	256	22.39	1.005	0.013	24.37	1.023	0.010
0.0977	10	256	23.98	1.027	0.014	24.41	1.022	0.013

图 15 客户端数量变化影响记录

将吞吐率随客户端数量的变化制成图 16 的折线图。由图可以看出，吞吐率随着客户端数量的增长，先上升后大幅度下降。

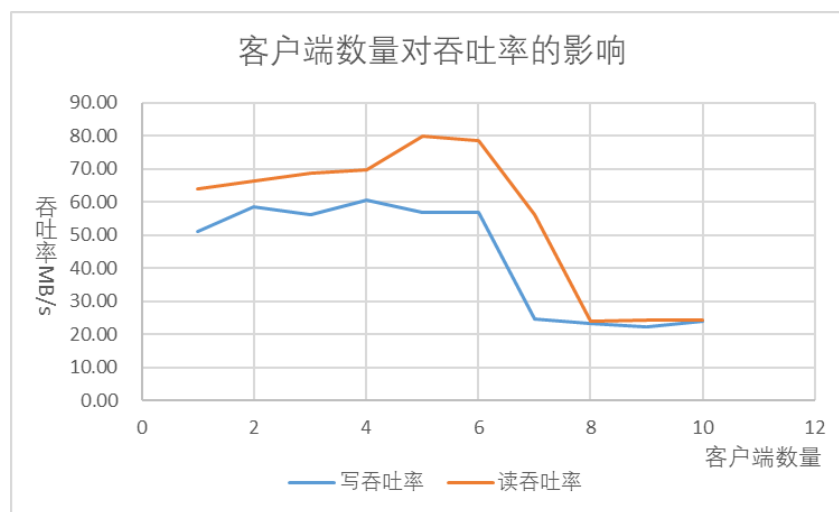


图 16 客户端数量对吞吐率的影响

将百分位延迟随客户端数量的变化制成图 17 的折线图。由图可以看出，99%时的延迟，在客户端数量大于 8 个后，大幅度增加，而 90%时的延迟则几乎不受客户端数量的影响。

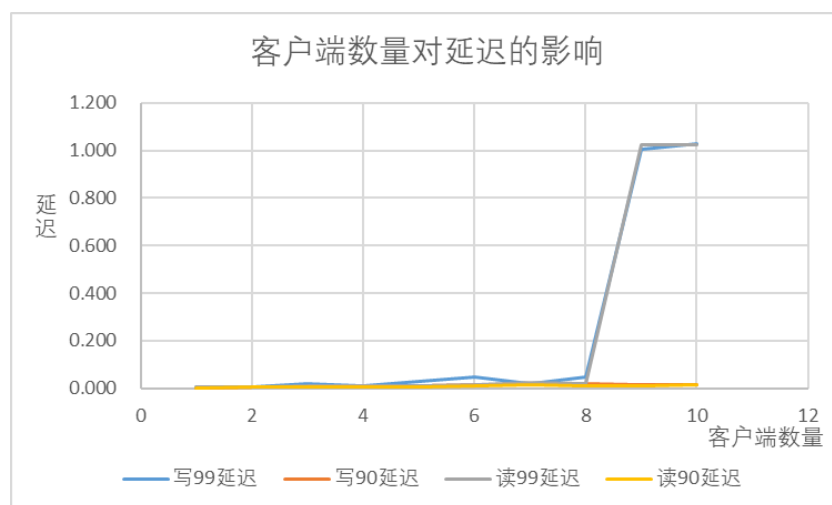


图 17 客户端数量对延迟的影响

六、实验总结

实验结果分析：

对象大小对对象存储系统性能的影响：根据图 13、14，可以分析得到，随着数据对象增大，单次传输量增加，使得服务器吞吐率增大。同时由于对象大小增加导致总体传输数据量增大，传输时延增加。

并行客户端数量对对象存储系统性能的影响：根据图 16、17，可以分析得到，随着并行客户端数量增加，对象存储系统对于并行处理的负载加重，尾延迟大幅度增长。同时，由于延迟增长，总时间变长，导致了整体吞吐率的下降。

实验体会：

通过本次实验，我对对象存储技术有了实际的接触和尝试，对对象存储系统的应用前景、优缺点有了更加深刻的了解。在实验中，我使用 minio、mock-s3 成功建立了对象存储服务，并分别通过 minio client、s3cmd 与对象存储系统进行交互，此外我还尝试了使用 s3bench 测试对象存储系统的性能，认识到了对象存储的工作原理以及尾延迟的原因。尽管本次实验中均是通过现成的服务端、客户端、测试程序，进行对象存储技术的研究，相信随着对这项技术的进一步学习、研究，我能够开发出属于自己的对象存储系统。

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.