

2016 级

《物联网数据存储与管理》课程

实 验 报 告

姓 名 杨钧淮

学 号 U201614907

班 号 物联网 1601 班

日 期 2018.05.28

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
五、实验过程.....	2
1. 基础实验.....	2
1.1 搭建 Minio 服务器	2
1.2 验证服务器	4
1.3 使用 minio client 客户端	4
2. 中等难度实验.....	5
2.1 搭建 s3proxy 服务器	5
2.2 验证服务器	6
2.3 使用 s3cmd 客户端.....	7
3. s3bench 测试及选题研究	8
3.1 Minio 测试结果	11
3.2 S3proxy 测试结果.....	11
3.3 结果分析与选题研究	12
六、实验总结.....	16
参考文献.....	17

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

物联网数据存储面临的挑战：

庞大：随着物联网的发展，联网设备数量以几何级数增长，所需要处理和存储的数据量也越来越庞大。

复杂：数据内容结构越来越丰富，越来越复杂。

所以我们需要一个易于扩展，支持并行的海量储存系统。这就有了我们的面向对象的存储技术。就像文件一样，对象包含数据，但是和文件不同的是，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。是一种综合了 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。

三、实验环境

CPU： Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz

内存： 8G

硬盘： SSD 240G

使用 Windows 10 1803 上的虚拟机运行 Ubuntu 16.04 LTS

虚拟机： VirtualBox 版本 6.0.4 r128413 (Qt5.6.2)

虚拟机内环境版本：

Ubuntu 16.04.6 LTS

Linux junhuai 4.15.0-47-generic #50~16.04.1-Ubuntu SMP Fri Mar 15 16:06:21

UTC 2019 x86_64 x86_64 x86_64 GNU/Linux

Java:

openjdk version "1.8.0_191"

OpenJDK Runtime Environment (build 1.8.0_191-8u191-b12-2ubuntu0.16.04.1-b12)

OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)

Go: go version go1.12.5 linux/amd64

Docker: Docker version 18.09.6, build 481bc77

四、实验内容

	基础实验	中等实验
Client	Minio Client	S3cmd
Server	Minio	S3proxy
Test	S3bench	S3bench

依照上表，针对不同难度的实验，服务器和客户端的选择如上所述。

使用自己改编的 s3bench 作为性能测试的工具，对对象存储性能的进行分析，测试读写性能与哪些参数有关。

五、实验过程

1. 基础实验

1.1 搭建 Minio 服务器

在实验中我使用 docker 虚拟机直接运行 Minio 服务器。

Docker 配置完成后，直接运行 docker hub 上的推荐命令。

```
sudo docker run -p 9000:9000 -v /mnt/data:/data -v /mnt/config:/root/.minio minio/minio server /data
```

docker 就会自动下载所有需要依赖的关系和包，下载完成后自动运行。

如下图所示。

```
sudo docker run -p 9000:9000 -v /mnt/data:/data -v /mnt/config:/root/.minio
[yjh@junhuai /etc/profile.d]
$ sudo docker run -p 9000:9000 -v /mnt/data:/data -v /mnt/config:/root/.minio
minio/minio server /data
Unable to find image 'minio/minio:latest' locally
latest: Pulling from minio/minio
bdf0201b3a05: Pull complete
67dced34a7ad: Pull complete
d328376474b1: Pull complete
f1fd3319a325: Pull complete
Digest: sha256:7ec09db11d048c48e3d63d507fda56feeb26d1dba317d082e774567fe49ff32d
Status: Downloaded newer image for minio/minio:latest

Endpoint: http://172.17.0.2:9000 http://127.0.0.1:9000

Browser Access:
http://172.17.0.2:9000 http://127.0.0.1:9000

Object API (Amazon S3 compatible):
Go: https://docs.min.io/docs/golang-client-quickstart-guide
Java: https://docs.min.io/docs/java-client-quickstart-guide
Python: https://docs.min.io/docs/python-client-quickstart-guide
JavaScript: https://docs.min.io/docs/javascript-client-quickstart-guide
.NET: https://docs.min.io/docs/dotnet-client-quickstart-guide
```

图 1 下载运行 Minio

由于我的电脑在直接启动时不会显示密钥的信息，所以在启动时我加入了自己配置的密钥信息。使用下面的命令生成一个 minio 镜像，并把名字命名为 myminio1，方便以后直接可以启动。

```
sudo docker run -p 9000:9000 --name myminio1 \
-e "MINIO_ACCESS_KEY=junhuai1" \
-e "MINIO_SECRET_KEY=yjh971029" \
-v /mnt/data:/data \
-v /mnt/config:/root/.minio \
minio/minio server /data
```

```
sudo docker run -p 9000:9000 --name myminio1 -e "MINIO_ACCESS_KEY=junhuai1" -
ready in use by container "cf29cd01d861175dd4b8f43a8e28016e5e41963cd6bc7b4cb208f
f5a93b1b016". You have to remove (or rename) that container to be able to reuse
that name.
See 'docker run --help'.
[yjh@junhuai /etc/profile.d]
$ sudo docker run -p 9000:9000 --name myminio1 \
-e "MINIO_ACCESS_KEY=junhuai1" \
-e "MINIO_SECRET_KEY=yjh971029" \
-v /mnt/data:/data \
-v /mnt/config:/root/.minio \
minio/minio server /data
Endpoint: http://172.17.0.2:9000 http://127.0.0.1:9000

Browser Access:
http://172.17.0.2:9000 http://127.0.0.1:9000

Object API (Amazon S3 compatible):
Go: https://docs.min.io/docs/golang-client-quickstart-guide
Java: https://docs.min.io/docs/java-client-quickstart-guide
Python: https://docs.min.io/docs/python-client-quickstart-guide
JavaScript: https://docs.min.io/docs/javascript-client-quickstart-guide
.NET: https://docs.min.io/docs/dotnet-client-quickstart-guide
```

图 2

如上图所示，可以看到 Minio 服务器已经正确运行在本地服务器的 9000 端口上。

1.2 验证服务器

为了验证服务器是否搭建成功,由于 Minio 提供一个 HTML 实现的图形界面,所以我们可以打开浏览器来验证,直接访问 127.0.0.1:9000。输入上面配置好的密钥,就可以进入 Minio Browser。

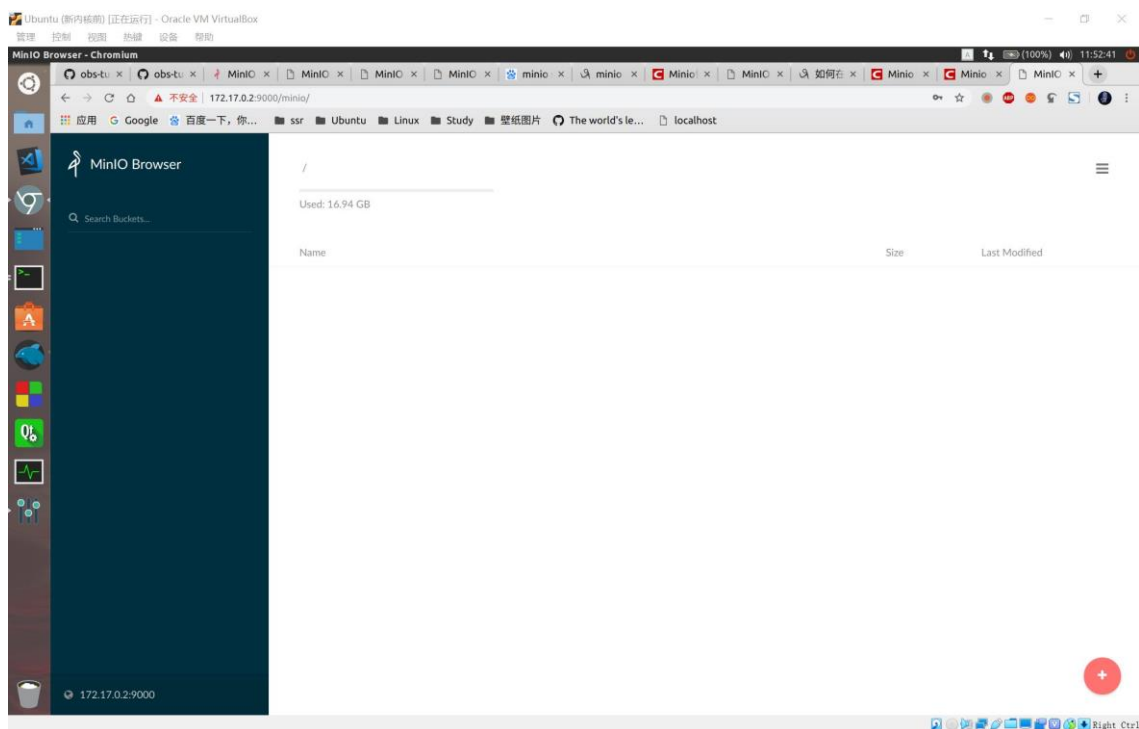


图 3 验证 minio 服务器

在后续的运行中,直接使用命令 `sudo docker start myminio1` 就可以直接启动服务器。



图 4 直接打开镜像

1.3 使用 minio client 客户端

首先配置 Minio Client 客户端,将当前的服务器配置保存。

如下图所示:

```
yjh@junhuai ~/project/StorageLab
$ sudo ./mc config host add yjh http://127.0.0.1:9000 junhuai1 yjh971029
[sudo] yjh 的密码:
Added `yjh` successfully.
```

图 5 配置 host

使用 `cp` 命令上传文件到对象服务器。

使用 `ls` 命令查看对象服务器桶内数据。

```
yjh@junhuai ~/project/StorageLab
$ sudo ./mc cp ~/project/OSDesign/README.md yjh/yjh
...README.md: 10 B / 10 B | ██████████ | 100.00% 575 B/s 0s
yjh@junhuai ~/project/StorageLab
$ sudo ./mc ls yjh/yjh
[2019-05-15 09:07:55 CST]      10B README.md
[2019-05-15 08:51:29 CST]   11KiB databackup.sql
yjh@junhuai ~/project/StorageLab
$
```

图 6 上传查看数据

可以看到浏览器中的图形界面文件上传成功。

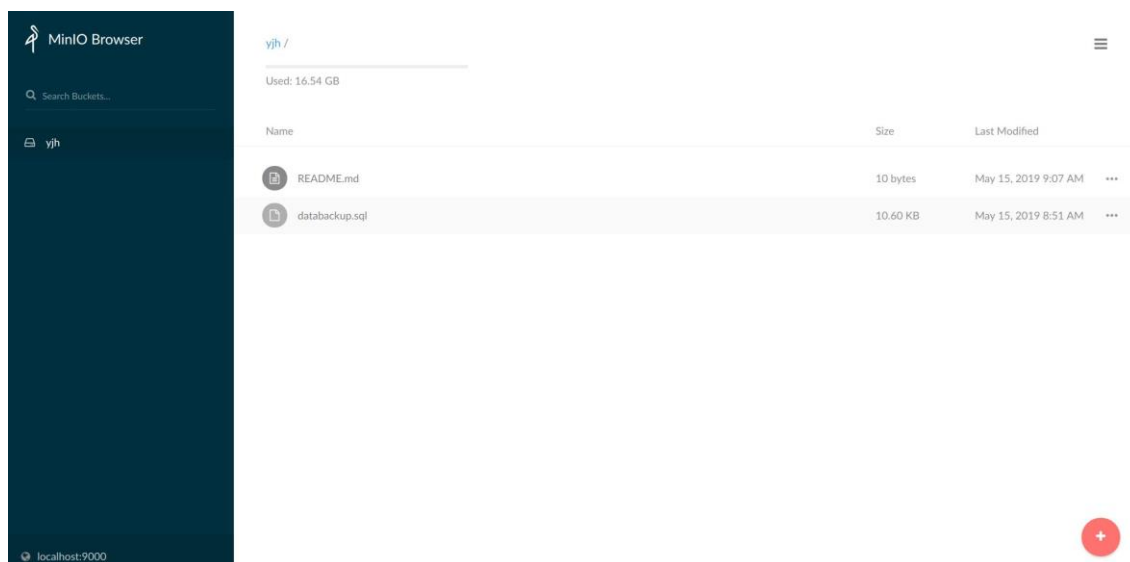


图 7

2. 中等难度实验

2.1 搭建 s3proxy 服务器

直接使用 `docker` 容器 `docker pull andrewgua/s3proxy` 即可将所有依赖的软件和包全部下载下来。

如下图所示。


```

root@junhuai ~/# docker pull andrewgaul/s3proxy
Using default tag: latest
latest: Pulling from andrewgaul/s3proxy
bdf0201b3a05: Already exists
9e12771959ad: Pull complete
fd21c20bbe6e: Pull complete
37219a05c8bc: Pull complete
03e9826f3521: Pull complete
Digest: sha256:321a3679d05cfe9ad28ca6d8dbe982e28b2d25665827b49f3fb4e2de8a1170e7
Status: Downloaded newer image for andrewgaul/s3proxy:latest

```

图 8 docker pull

然后直接使用 `docker run` 来运行 `s3Proxy` 服务器。需要注意的是，`s3proxy` 内部默认的端口为 80，但是我的环境中 80 端口已经被 `Apache` 服务器占用了。所以我将它映射到 9010 端口。再给镜像命名 `mys3proxy`，方便下次启动。如下图所示：

```

yjh@junhuai ~$ sudo docker run --name mys3proxy --publish 9010:80 --env S3PROXY_AUTHORIZATION=none andrewgaul/s3proxy
[s3proxy] I 05-22 02:46:38.498 main o.g.s.CrossOriginResourceSharing:82 [::] CORS allowed origins: []
[s3proxy] I 05-22 02:46:38.504 main o.g.s.CrossOriginResourceSharing:83 [::] CORS allowed methods: []
[s3proxy] I 05-22 02:46:38.504 main o.g.s.CrossOriginResourceSharing:84 [::] CORS allowed headers: []
[s3proxy] I 05-22 02:46:38.585 main o.g.s.o.eclipse.jetty.util.log:186 [::] Logging initialized @3942ms
[s3proxy] I 05-22 02:46:38.887 main o.g.s.o.e.jetty.server.Server:327 [::] jetty-9.2.z-SNAPSHOT
[s3proxy] I 05-22 02:46:38.997 main o.g.s.o.e.j.s.ServerConnector:266 [::] Started ServerConnector@f4080ae{HTTP/1.1}{0.0.0.0:80}
[s3proxy] I 05-22 02:46:38.998 main o.g.s.o.e.jetty.server.Server:379 [::] Started @4355ms

```

图 9 运行 s3proxy

2.2 验证服务器

测试服务器是否成功启动。使用一个 `request PUT` 命令，创建一个 `testbucket`

```

yjh@junhuai ~$ curl --request PUT http://127.0.0.1:9010/testbucket

```

图 10 测试创建 bucket

在浏览器中输入 <http://127.0.0.1:9010> 就可以看到新的 bucket 创建成功了。

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```

<ListAllMyBucketsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Owner>
    <ID>
      75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6cae54ba06a
    </ID>
    <DisplayName>CustomersName@amazon.com</DisplayName>
  </Owner>
  <Buckets>
    <Bucket>
      <Name>testbucket</Name>
      <CreationDate>2019-05-22T00:43:36.000Z</CreationDate>
    </Bucket>
  </Buckets>
</ListAllMyBucketsResult>

```

图 11 浏览器中打开

2.3 使用 s3cmd 客户端

使用 pip 下载 s3cmd 客户端 `pip install s3cmd`

使用 `s3cmd config` 命令来运行配置引导。这里遇到的问题是，由于 s3proxy 我没有设置密钥，所以在配置时，我也没有输入密钥，这样就导致了程序出现了 bug，无法成功连接到服务器。

S3cmd 的配置如下图：

```
New settings: 99aa0c8caeb4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a
Access Key: sdf
Secret Key: dfg
Default Region: US
S3 Endpoint: 127.0.0.1:9010
DNS-style bucket+hostname:port template for accessing a bucket: 127.0.0.1:9010
Encryption password:
Path to GPG program: /usr/bin/gpg
Use HTTPS protocol: False
HTTP Proxy server name:
HTTP Proxy server port: 0
Test access with supplied credentials? [Y/n] y
Please wait, attempting to list all buckets...
Success. Your access key and secret key worked fine :)
Now verifying that encryption works...
Not configured. Never mind.
Save settings? [y/N] y
Configuration saved to '/home/yjh/.s3cfg'
```

图 12 s3cmd

后面开始测试 s3cmd 所具有的命令。

```
yjh@junhuai ~
$ s3cmd mb s3://testbucket
Bucket 's3://testbucket/' created
yjh@junhuai ~
$ s3cmd ls
2019-05-22 02:48 s3://testbucket
```

图 13 测试创建 bucket

如上传文件 `put`。将一张图片上传到对象存储服务器。通过 `ls` 命令就可以看到当前服务器所有存储的对象文件。

```
yjh@junhuai ~
$ s3cmd put 1.jpg s3://testbucket/image/1.jpg
upload: '1.jpg' -> 's3://testbucket/image/1.jpg' [1 of 1]
799757 of 799757 (100%) in 0s 15.12 MB/s done
yjh@junhuai ~
$ s3cmd ls s3://testbucket
DIR s3://testbucket/image/
yjh@junhuai ~
$ s3cmd ls -r s3://testbucket
2019-05-22 02:36 799757 s3://testbucket/image/1.jpg
yjh@junhuai ~
```

图 14 上传文件并查看

可以看到浏览器端，该文件以及在服务器中了

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>testbucket</Name>
  <Prefix/>
  <MaxKeys>1000</MaxKeys>
  <Marker/>
  <IsTruncated>false</IsTruncated>
  <Contents>
    <Key>image/1.jpg</Key>
    <LastModified>2019-05-22T02:36:48Z</LastModified>
    <ETag>"a2a7207d53fc52614633674d2e8fe8bc"</ETag>
    <Size>799757</Size>
    <StorageClass>STANDARD</StorageClass>
    <Owner>
      <ID>
        75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a
      </ID>
      <DisplayName>CustomersName@amazon.com</DisplayName>
    </Owner>
  </Contents>
</ListBucketResult>
```

图 15 浏览器端

测试下载文件。使用 get 命令

```
yjh@junhuai ~
$ s3cmd get s3://testbucket/image/1.jpg 2.jpg
download: 's3://testbucket/image/1.jpg' -> '2.jpg' [1 of 1]
799757 of 799757 100% in 0s 15.47 MB/s done
yjh@junhuai ~
$ ls 2.jpg
2.jpg
```

图 16 下载

删除命令 del

```
yjh@junhuai ~
$ s3cmd del s3://testbucket/image/1.jpg
delete: 's3://testbucket/image/1.jpg'
yjh@junhuai ~
$ s3cmd ls -r s3://testbucket
```

图 17 删除命令

3. s3bench 测试及选题研究

为了测试搭建的服务器的性能，以及服务器的百分位延迟，我们使用 s3bench 测试。

s3bench 使用 go 语言开发，程序简介易懂。但是由于它不像 cosbench 有批测试的功能。所以我们需要自己写脚本文件来测试。但是由于测试的输出是一个很详细的文本，即使重定向到文件之后，数据的处理也很困难。所以我直接修改 cosbench 的源代码，使其测试文件生成一个.csv 文件，易于对数据分析处理。

```
#!/bin/bash

# minio on port 9000
# s3proxy on port 9010
endpoint="http://127.0.0.1:9000"
# endpoint="http://127.0.0.1:9010"
bucket="testbucket"
ObjectNamePrefix="test"
AccessKey="junhuai1"
AccessSecret="yjh971029"
# filepath="s3proxy.csv"
filepath="minio.csv"

declare -a NumClient
declare -a NumSample
declare -a ObjectSize

NumClient=(1      1      1      1      1      1      1      1      1      1      1      1
1    2    4    8   16   32   64   70   80   90  100  2          4      8
16   32   64   1    1    1    1    1    1    1    1    1    1 )
NumSample=(100  100  100  100  100  100  100  100  100  100  100  100  100  100
100    100  100  100  100  100  100  100  100  100  100  100  100  100
100    100  100  100    5   10   20  40  80  160  320  640  1280 )
ObjectSize=(1024 2048 4096 10240 20480 40960 102400 204800 409600 1048576
4194304 4096 4096 4096 4096 4096 4096 4096 4096 4096 4096 4096 102400
102400 102400 102400 102400 102400    4096  4096 4096 4096 4096 4096 4096
4096 4096 )

for(( i=0;i<${#NumClient[@]};i++))
do
    # å‘ä»æä»¶
    ./s3bench      -accessKey=$AccessKey      -accessSecret=$AccessSecret -
bucket=$bucket -endpoint=$endpoint \
        -numClients=${NumClient[i]}           -numSamples=${NumSample[i]} -
objectNamePrefix=$ObjectNamePrefix            -objectSize=${ObjectSize[i]} -
filepath=$filepath

    echo "-----done"
done
```

```
NumClient=(1 1 1 1 1 1 1 1 1 1 1)
NumSample=(100 100 100 100 100 100 100 100 100 100 100)
ObjectSize=(1024 2048 4096 10240 20480 40960 102400 204800 409600 1048576 4194304)
```

9

象存储系统的关系。

测试数据集 2:

1	2	4	8	16	32	64	70	80	90	100
100	100	100	100	100	100	100	100	100	100	100
4096	4096	4096	4096	4096	4096	4096	4096	4096	4096	4096

对象大小为 4kb，并发数量从 1 增加到 100，用于测试并发客户端对对象存储系统的影响。

测试数据集 3:

2	4	8	16	32	64
100	100	100	100	100	100
102400	102400	102400	102400	102400	102400

同 2，只是将对象大小改为 100kb，查看两者之间的关系。

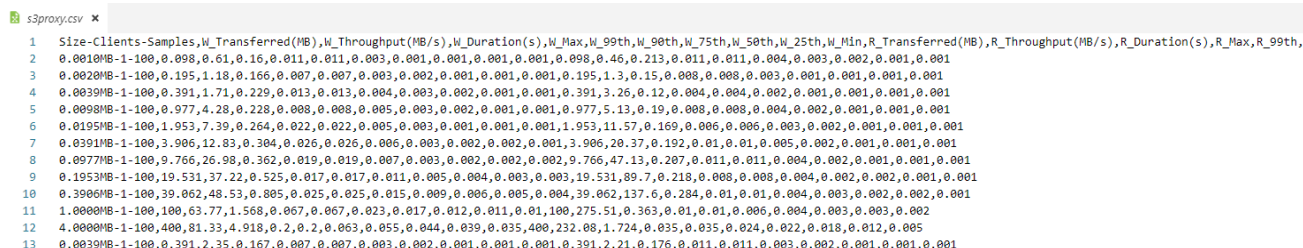
测试数据集 4:

1	1	1	1	1	1	1	1	1)
5	10	20	40	80	160	320	640	1280)
4096	4096	4096	4096	4096	4096	4096	4096	4096)

对象大小为 4kb，并发客户端为 1，只是对象数量变多了，从 5 增加到 1280，用于查看对象数量对对象存储系统的影响。

我们分别对 minio 和 s3Proxy 进行测试，测试结果自动生成.csv 文件。

生成的.csv 文件部分如下所示：



```
s3proxy.csv
1 Size-Clients-Samples,W_Transferred(MB),W_Throughput(MB/s),W_Duration(s),W_Max,W_99th,W_90th,W_75th,W_50th,W_25th,W_Min,R_Transferred(MB),R_Throughput(MB/s),R_Duration(s),R_Max,R_99th,
2 0.0010MB-1-100,0.098,0.61,0.16,0.011,0.011,0.003,0.001,0.001,0.001,0.001,0.098,0.46,0.213,0.011,0.011,0.004,0.003,0.002,0.001,0.001
3 0.0020MB-1-100,0.195,1.18,0.166,0.007,0.007,0.003,0.002,0.001,0.001,0.001,0.195,1.3,0.15,0.008,0.008,0.003,0.001,0.001,0.001,0.001
4 0.0030MB-1-100,0.391,1.71,0.229,0.013,0.013,0.004,0.003,0.002,0.001,0.001,0.391,3.26,0.12,0.004,0.004,0.002,0.001,0.001,0.001,0.001
5 0.0090MB-1-100,0.977,4.28,0.228,0.008,0.008,0.005,0.003,0.002,0.001,0.001,0.977,5.13,0.19,0.008,0.008,0.004,0.002,0.001,0.001,0.001
6 0.0195MB-1-100,1.953,7.39,0.264,0.022,0.022,0.005,0.003,0.001,0.001,0.001,1.953,11.57,0.169,0.006,0.006,0.003,0.002,0.001,0.001,0.001
7 0.0391MB-1-100,3.906,12.83,0.304,0.026,0.026,0.006,0.003,0.002,0.002,0.001,3.906,20.37,0.192,0.01,0.01,0.005,0.002,0.001,0.001,0.001
8 0.0977MB-1-100,9.766,26.98,0.362,0.019,0.019,0.007,0.003,0.002,0.002,0.002,9.766,47.13,0.207,0.011,0.011,0.004,0.002,0.001,0.001,0.001
9 0.1953MB-1-100,19.531,37.22,0.525,0.017,0.017,0.011,0.005,0.004,0.003,0.003,19.531,89.7,0.218,0.008,0.008,0.004,0.002,0.002,0.001,0.001
10 0.3906MB-1-100,39.062,48.53,0.805,0.025,0.025,0.015,0.009,0.006,0.005,0.004,39.062,137.6,0.284,0.01,0.01,0.004,0.003,0.002,0.002,0.001
11 1.0000MB-1-100,100.63,77.1,568,0.067,0.067,0.023,0.017,0.012,0.011,0.01,100.275,51.363,0.01,0.01,0.006,0.004,0.003,0.003,0.002
12 4.0000MB-1-100,400.81,33.4,918,0.2,0.2,0.063,0.055,0.044,0.039,0.035,400.232,0.08,1.724,0.035,0.035,0.024,0.022,0.018,0.012,0.005
13 0.0039MB-1-100,0.391,2.35,0.167,0.007,0.007,0.003,0.002,0.001,0.001,0.001,0.391,2.21,0.176,0.011,0.011,0.003,0.002,0.001,0.001,0.001
```

图 18 CSV 文件

我们使用 Excel 打开，对格式进行一定的处理，方便我们分析数据。

3.1 Minio 测试结果

1	Size-Clients-Samples	W_Transferred(MB)	W_Throughput(MB/s)	W_Duration(s)	W_Max	W_99th	W_90th	W_75th	W_50th	W_25th	W_Min	R_Transferred(MB)	R_Throughput(MB/s)	R_Duration(s)	R_Max	R_99th	R_90th	R_75th	R_50th	R_25th	R_Min
2	0.0010MB-1-100	0.098	0.1	0.974	0.032	0.032	0.013	0.01	0.009	0.009	0.004	0.098	0.55	0.177	0.01	0.01	0.003	0.002	0.001	0.001	0.001
3	0.0020MB-1-100	0.195	0.21	0.908	0.016	0.016	0.012	0.01	0.009	0.009	0.004	0.195	1.11	0.176	0.005	0.005	0.003	0.002	0.001	0.001	0.001
4	0.0039MB-1-100	0.391	0.41	0.942	0.015	0.015	0.011	0.01	0.009	0.009	0.004	0.391	1.71	0.229	0.009	0.009	0.005	0.002	0.002	0.001	0.001
5	0.0098MB-1-100	0.977	1.04	0.943	0.014	0.014	0.011	0.01	0.009	0.009	0.004	0.977	4.33	0.226	0.01	0.01	0.004	0.002	0.001	0.001	0.001
6	0.0195MB-1-100	1.953	2.04	0.957	0.016	0.016	0.012	0.01	0.009	0.009	0.004	1.953	9.71	0.201	0.015	0.015	0.004	0.002	0.001	0.001	0.001
7	0.0391MB-1-100	3.906	4.08	0.955	0.016	0.016	0.011	0.01	0.009	0.009	0.006	3.906	17.78	0.22	0.013	0.013	0.004	0.002	0.002	0.001	0.001
8	0.0977MB-1-100	9.766	9.83	0.993	0.019	0.019	0.014	0.01	0.009	0.009	0.004	9.766	37.28	0.262	0.023	0.023	0.005	0.003	0.002	0.001	0.001
9	0.1953MB-1-100	19.531	19.74	0.989	0.025	0.025	0.014	0.01	0.009	0.009	0.005	19.531	83.72	0.233	0.012	0.012	0.004	0.002	0.002	0.001	0.001
10	0.3906MB-1-100	39.062	38.49	1.015	0.028	0.028	0.014	0.01	0.009	0.009	0.007	39.062	138.95	0.281	0.014	0.014	0.005	0.003	0.002	0.002	0.001
11	1.0000MB-1-100	100	56.94	1.756	0.041	0.041	0.025	0.02	0.017	0.013	0.01	100	277.44	0.36	0.015	0.015	0.005	0.004	0.003	0.003	0.002
12	4.0000MB-1-100	400	63.04	6.345	1.037	1.037	0.074	0.05	0.039	0.035	0.028	400	484.94	0.825	0.028	0.028	0.012	0.009	0.007	0.006	0.005
13	0.0039MB-1-100	0.391	0.42	0.921	0.015	0.015	0.011	0.01	0.009	0.009	0.003	0.391	1.96	0.199	0.013	0.013	0.003	0.002	0.001	0.001	0.001
14	0.0039MB-2-100	0.391	0.82	0.476	0.023	0.023	0.012	0.01	0.009	0.008	0.004	0.391	2.57	0.152	0.015	0.015	0.007	0.003	0.002	0.001	0.001
15	0.0039MB-4-100	0.391	1.21	0.323	0.03	0.03	0.02	0.015	0.011	0.008	0.005	0.391	2.58	0.152	0.019	0.019	0.012	0.008	0.005	0.003	0.001
16	0.0039MB-8-100	0.391	1.75	0.223	0.054	0.054	0.024	0.02	0.016	0.012	0.007	0.391	3.09	0.126	0.033	0.033	0.02	0.013	0.008	0.005	0.001
17	0.0039MB-16-100	0.391	1.62	0.241	0.085	0.085	0.056	0.04	0.031	0.025	0.008	0.391	2.69	0.145	0.059	0.059	0.044	0.031	0.016	0.01	0.003
18	0.0039MB-32-100	0.391	1.61	0.243	0.162	0.162	0.12	0.093	0.06	0.041	0.008	0.391	2.97	0.131	0.086	0.086	0.065	0.047	0.034	0.022	0.006
19	0.0039MB-64-100	0.391	1.35	0.29	0.245	0.245	0.202	0.167	0.135	0.088	0.017	0.391	2.12	0.184	0.178	0.178	0.142	0.115	0.075	0.044	0.012
20	0.0039MB-70-100	0.391	1.32	0.296	0.244	0.244	0.213	0.166	0.115	0.089	0.013	0.391	2.36	0.166	0.139	0.139	0.115	0.094	0.076	0.058	0.017
21	0.0039MB-80-100	0.391	1.43	0.272	0.246	0.246	0.212	0.203	0.151	0.115	0.033	0.391	2.44	0.16	0.145	0.145	0.108	0.082	0.069	0.053	0.013
22	0.0039MB-90-100	0.391	1.41	0.277	0.256	0.256	0.243	0.199	0.168	0.101	0.008	0.391	1.44	0.272	0.233	0.233	0.189	0.127	0.079	0.069	0.01
23	0.0039MB-100-100	0.391	1	0.39	0.374	0.374	0.28	0.263	0.199	0.164	0.112	0.391	1.66	0.236	0.219	0.219	0.19	0.185	0.143	0.117	0.032
24	0.0977MB-2-100	9.766	16.68	0.586	0.028	0.028	0.018	0.012	0.01	0.009	0.005	9.766	46.1	0.212	0.017	0.017	0.009	0.005	0.003	0.002	0.001
25	0.0977MB-4-100	9.766	23.28	0.42	0.039	0.039	0.028	0.02	0.015	0.01	0.004	9.766	66.3	0.147	0.037	0.037	0.01	0.008	0.004	0.003	0.001
26	0.0977MB-8-100	9.766	25.91	0.377	0.074	0.074	0.044	0.035	0.026	0.018	0.007	9.766	62.63	0.156	0.042	0.042	0.019	0.015	0.01	0.007	0.002
27	0.0977MB-16-100	9.766	26.56	0.368	0.101	0.101	0.078	0.065	0.049	0.037	0.013	9.766	83.67	0.117	0.045	0.045	0.03	0.021	0.014	0.011	0.002
28	0.0977MB-32-100	9.766	27.87	0.35	0.148	0.148	0.119	0.093	0.071	0.053	0.026	9.766	80.48	0.121	0.093	0.093	0.055	0.039	0.024	0.018	0.003
29	0.0977MB-64-100	9.766	24.52	0.398	0.393	0.393	0.335	0.275	0.175	0.122	0.021	9.766	43.65	0.224	0.214	0.214	0.188	0.142	0.116	0.062	0.004
30	0.0039MB-1-5	0.02	0.33	0.06	0.019	0.019	0.019	0.017	0.009	0.009	0.005	0.02	1.52	0.013	0.004	0.004	0.004	0.003	0.002	0.002	0.002
31	0.0039MB-1-10	0.039	0.44	0.089	0.011	0.011	0.011	0.01	0.009	0.007	0.006	0.039	1.45	0.027	0.007	0.007	0.007	0.004	0.002	0.001	0.001
32	0.0039MB-1-20	0.078	0.4	0.193	0.016	0.016	0.015	0.01	0.009	0.009	0.005	0.078	1.73	0.045	0.008	0.008	0.003	0.003	0.002	0.002	0.001
33	0.0039MB-1-40	0.156	0.43	0.363	0.014	0.014	0.012	0.01	0.009	0.009	0.004	0.156	2.03	0.077	0.006	0.006	0.003	0.002	0.002	0.001	0.001
34	0.0039MB-1-80	0.312	0.41	0.753	0.016	0.016	0.013	0.01	0.009	0.009	0.004	0.312	1.4	0.223	0.023	0.023	0.005	0.003	0.002	0.001	0.001
35	0.0039MB-1-160	0.625	0.42	1.495	0.02	0.02	0.013	0.01	0.009	0.009	0.004	0.625	2.15	0.291	0.009	0.007	0.003	0.002	0.001	0.001	0.001
36	0.0039MB-1-320	1.25	0.43	2.901	0.015	0.015	0.01	0.009	0.009	0.009	0.003	1.25	2.3	0.543	0.022	0.008	0.002	0.002	0.001	0.001	0.001
37	0.0039MB-1-640	2.5	0.43	5.84	0.023	0.015	0.01	0.009	0.009	0.009	0.003	2.5	2.25	1.113	0.013	0.006	0.002	0.002	0.002	0.001	0.001
38	0.0039MB-1-1280	5	0.43	11.721	0.025	0.016	0.011	0.009	0.009	0.009	0.003	5	2.54	1.972	0.016	0.006	0.002	0.002	0.001	0.001	0.001

图 19 Minio 测试结果

3.2 S3proxy 测试结果

	Size-Clients-Samples	W_Transferred(MB)	W_Throughput(MB/s)	W_Duration(s)	W_Max	W_99th	W_90th	W_75th	W_50th	W_25th	W_Min	R_Transferred(MB)	R_Throughput(MB/s)	R_Duration(s)	R_Max	R_99th	R_90th	R_75th	R_50th	R_25th	R_Min
2	0.0010MB-1-100	0.098	0.61	0.16	0.011	0.011	0.003	0.001	0.001	0.001	0.001	0.098	0.46	0.213	0.011	0.011	0.004	0.003	0.002	0.001	0.001
3	0.0020MB-1-100	0.195	1.18	0.166	0.007	0.007	0.003	0.002	0.001	0.001	0.001	0.195	1.3	0.15	0.008	0.008	0.003	0.001	0.001	0.001	0.001
4	0.0039MB-1-100	0.391	1.71	0.229	0.013	0.013	0.004	0.003	0.002	0.001	0.001	0.391	3.26	0.12	0.004	0.004	0.002	0.001	0.001	0.001	0.001
5	0.0098MB-1-100	0.977	4.28	0.228	0.008	0.008	0.005	0.003	0.002	0.001	0.001	0.977	5.13	0.19	0.008	0.008	0.004	0.002	0.001	0.001	0.001
6	0.0195MB-1-100	1.953	7.39	0.264	0.022	0.022	0.005	0.003	0.001	0.001	0.001	1.953	11.57	0.169	0.006	0.006	0.003	0.002	0.001	0.001	0.001
7	0.0391MB-1-100	3.906	12.83	0.304	0.026	0.026	0.006	0.003	0.002	0.002	0.001	3.906	20.37	0.192	0.01	0.01	0.005	0.002	0.001	0.001	0.001
8	0.0977MB-1-100	9.766	26.98	0.362	0.019	0.019	0.007	0.003	0.002	0.002	0.002	9.766	47.13	0.207	0.011	0.011	0.004	0.002	0.001	0.001	0.001
9	0.1953MB-1-100	19.531	37.22	0.525	0.017	0.017	0.011	0.005	0.004	0.003	0.003	19.531	89.7	0.218	0.008	0.008	0.004	0.002	0.002	0.001	0.001
10	0.3906MB-1-100	39.062	48.53	0.805	0.025	0.025	0.015	0.009	0.006	0.005	0.004	39.062	137.6	0.284	0.01	0.01	0.004	0.003	0.002	0.002	0.001
11	1.0000MB-1-100	100	63.77	1.568	0.067	0.067	0.023	0.017	0.012	0.011	0.01	100	275.51	0.363	0.01	0.01	0.006	0.004	0.003	0.003	0.002
12	4.0000MB-1-100	400	81.33	4.918	0.2	0.2	0.083	0.055	0.044	0.039	0.035	400	232.08	1.724	0.035	0.035	0.024	0.022	0.018	0.012	0.005
13	0.0039MB-1-100	0.391	2.35	0.167	0.007	0.007	0.003	0.002	0.001	0.001	0.001	0.391	2.21	0.176	0.011	0.011	0.003	0.002	0.001	0.001	0.001
14	0.0039MB-2-100	0.391	2.92	0.134	0.015	0.015	0.006	0.003	0.002	0.001	0.001	0.391	2.83	0.138	0.012	0.012	0.006	0.003	0.002	0.001	0.001
15	0.0039MB-4-100	0.391	3.73	0.105	0.011	0.011	0.007	0.004	0.003	0.002	0.001	0.391	3.27	0.12	0.022	0.022	0.01	0.005	0.003	0.002	0.001
16	0.0039MB-8-100	0.391	3.36	0.116	0.031	0.031	0.015	0.011	0.007	0.004	0.001	0.391	3.22	0.121	0.025	0.025	0.012	0.009	0.006	0.004	0.001
17	0.0039MB-16-100	0.391	2.52	0.155	0.048	0.048	0.034	0.023	0.016	0.012	0.001	0.391	3.59	0.109	0.041	0.041	0.025	0.017	0.011	0.009	0.002
18	0.0039MB-32-100	0.391	2.2	0.177	0.084	0.084	0.063	0.046	0.031	0.026	0.003	0.391	4.31	0.091	0.04	0.04	0.029	0.024	0.02	0.015	0.004
19	0.0039MB-64-100	0.391	2.74	0.143	0.112	0.112	0.085	0.067	0.05	0.039	0.013	0.391	4.54	0.086	0.051	0.051	0.043	0.039	0.03	0.02	0.008
20	0.0039MB-70-100	0.391	1.55	0.252	0.164	0.164	0.138	0.102	0.082	0.059	0.023	0.391	1.79	0.218	0.202	0.202	0.178	0.164	0.136	0.047	0.009
21	0.0039MB-80-100	0.391	1.51	0.258	0.223	0.223	0.187	0.153	0.116	0.095	0.021	0.391	2.23	0.175	0.162	0.162	0.13	0.121	0.096	0.772	0.02
22	0.0039MB-90-100	0.391	1.66	0.236	0.102	0.102	0.09	0.082	0.06	0.04	0.005	0.391	3.19	0.122	0.103	0.103	0.088	0.067	0.055	0.03	0.013
23	0.0039MB-100-100	0.391	2.12	0.184	0.084	0.084	0.067	0.059	0.052	0.04	0.003	0.391	2.59	0.151	0.128	0.128	0.114	0.106	0.092	0.081	0.043
24	0.0977MB-2-100	9.766	35.5	0.275	0.016	0.016	0.012	0.006	0.005	0.003	0.002	9.766	65.26	0.15	0.014	0.014	0.005	0.003	0.002	0.002	0.001
25	0.0977MB-4-100	9.766	31.71	0.308	0.033	0.033	0.021	0.015	0.01	0.006	0.002	9.766	65.2	0.15	0.015	0.015	0.011	0.008	0.005	0.003	0.002
26	0.0977MB-8-100	9.766	36.21	0.27	0.075	0.075	0.043	0.025	0.014	0.009	0.003	9.766	73.07	0.134	0.038	0.038	0.02	0.011	0.008	0.005	0.002
27	0.0977MB-16-100	9.766	38.68	0.252	0.064	0.064	0.048	0.036	0.026	0.016	0.005	9.766	61.33	0.159	0.043	0.043	0.031	0.027	0.02	0.012	0.001
28	0.0977MB-32-100	9.766	32.56	0.3	0.182	0.182	0.105	0.084	0.062	0.046	0.01	9.766	51.68	0.189	0.089	0.089	0.068	0.053	0.034	0.022	0.004
29	0.0977MB-64-100	9.766	30.45	0.32	0.148	0.148	0.254	0.183	0.133	0.106	0.039	9.766	49.57	0.191	0.155	0.155	0.128	0.107	0.081	0.058	0.021
30	0.0039MB-1-5	0.02	0.17	0.01	0.01	0.01	0.001	0.001	0.001	0.001	0.001	0.02	0.22	0.009	0.003	0.003	0.003	0.002	0.001	0.001	0.001
31	0.0039MB-1-10	0.039	0.78	0.05	0.02	0.02	0.02	0.006	0.002	0.002	0.001	0.039	2.22	0.018	0.005	0.005	0.005	0.002	0.001	0.001	0.001
32	0.0039MB-1-20	0.078	1.28	0.034	0.005	0.005	0.005	0.002	0.001	0.001	0.001	0.078	3.16	0.025	0.003	0.003	0.002	0.001	0.001	0.001	0.001
33	0.0039MB-1-40	0.156	1.47	0.106	0.013	0.013	0.006	0.003	0.002	0.001	0.001	0.156	2.2	0.071	0.005	0.005	0.004	0.002	0.001	0.001	0.001
34	0.0039MB-1-80	0.312	1.93	0.162	0.01	0.01	0.004	0.002	0.001	0.001	0.001	0.312	2.47	0.126	0.006	0.006	0.003	0.002	0.001	0.001	0.001
35	0.0039MB-1-160	0.625	2.46	0.254	0.01	0.006	0.003	0.001	0.001	0.001	0.001	0.625	2.44	0.256	0.015	0.01	0.003	0.002	0.001	0.001	0.001
36	0.0039MB-1-320	1.25	2.54	0.492	0.009	0.006	0.002	0.001	0.001	0.001	0.001	1.25	2.76	0.453	0.015	0.009	0.002	0.001	0.001	0.001	0.001
37	0.0039MB-1-640	2.5	2.87	0.87	0.005	0.005	0.002	0.001	0.001	0.001	0.001	2.5	3.43	0.729	0.007	0.004	0.001	0.001	0.001	0.001	0.001
38	0.0039MB-1-1280	5	3.04	1.644	0.013	0.005	0.002	0.001	0.001	0.001	0.001	5	3.33	1.499	0.008	0.004	0.001	0.001	0.001	0.001	0.001

3.3 结果分析与选题研究

1. 对象尺寸如何影响性能?

首先我们分析 minio 的对象尺寸对性能的影响，如下表。

1	Size-Clients-Samples	W_Throughput(MB/s)	W_Duration(s)	W_99th	W_90th	R_Throughput(MB/s)	R_Duration(s)	R_99th	R_90th
2	0.0010MB-1-100	0.1	0.974	0.032	0.013	0.55	0.177	0.01	0.003
3	0.0020MB-1-100	0.21	0.908	0.016	0.012	1.11	0.176	0.005	0.003
4	0.0039MB-1-100	0.41	0.942	0.015	0.011	1.71	0.229	0.009	0.005
5	0.0098MB-1-100	1.04	0.943	0.014	0.011	4.33	0.226	0.01	0.004
6	0.0195MB-1-100	2.04	0.957	0.016	0.012	9.71	0.201	0.015	0.004
7	0.0391MB-1-100	4.08	0.958	0.016	0.011	17.78	0.22	0.013	0.004
8	0.0977MB-1-100	9.83	0.993	0.019	0.014	37.28	0.262	0.023	0.005
9	0.1953MB-1-100	19.74	0.989	0.025	0.014	83.72	0.233	0.012	0.004
10	0.3906MB-1-100	38.49	1.015	0.028	0.014	138.95	0.281	0.014	0.005
11	1.0000MB-1-100	56.94	1.756	0.041	0.025	277.44	0.36	0.015	0.005
12	4.0000MB-1-100	63.04	6.345	1.037	0.074	484.94	0.825	0.028	0.012

图 21 Minio

可以看到，对象尺寸越大，吞吐率也越大。在 Minio 中，当对象尺寸为 4MB 时，写入速率可达到 63MB/s，读取速率可以达到 485MB/s。这个速率是服务器运行在本地的速率，对网络性能的影响可以忽略。而在实际应用中，吞吐率往往还与网络带宽速率有关。

下面我们来看对象的百分位延迟。可以看到，无论是读或写，99%的对象延迟随着对象大小先减小后增大，对象尺寸为 4MB 时变得非常大。

如下图所示：

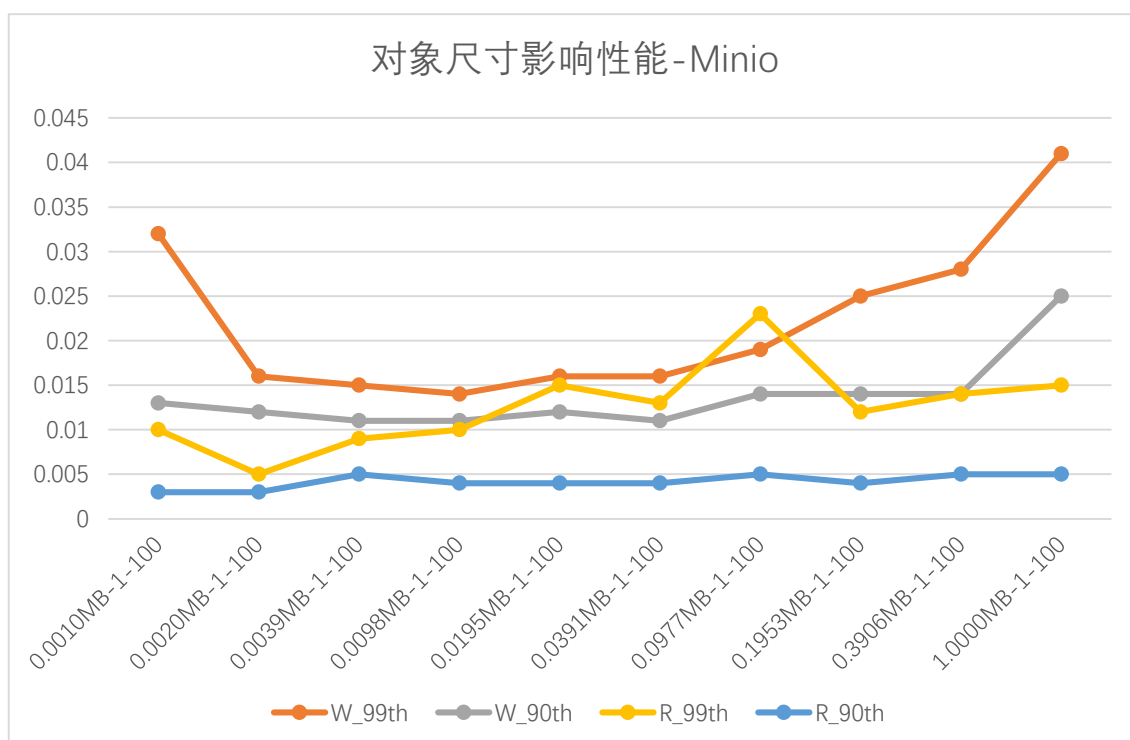


图 22 Minio 延迟

可以看到对于写操作中，90%的对象在对象较小时，延迟其实变化不大。但是99%的对象在对象很小时也有较大的延迟。

可以看到对于读操作中，延迟变化也不大。

从表格和折线图中我们可以得到一个定性的结论，对象尺寸越大，读写延迟越大，但是吞吐率也越大。

同理，我们观察 s3Proxy 的结果。

1	Size-Clients-Samples	W_Throughput(MB/s)	W_Duration(s)	W_99th	W_90th	R_Throughput(MB/s)	R_Duration(s)	R_99th	R_90th
2	0.0010MB-1-100	0.61	0.16	0.011	0.003	0.46	0.213	0.011	0.004
3	0.0020MB-1-100	1.18	0.166	0.007	0.003	1.3	0.15	0.008	0.003
4	0.0039MB-1-100	1.71	0.229	0.013	0.004	3.26	0.12	0.004	0.002
5	0.0098MB-1-100	4.28	0.228	0.008	0.005	5.13	0.19	0.008	0.004
6	0.0195MB-1-100	7.39	0.264	0.022	0.005	11.57	0.169	0.006	0.003
7	0.0391MB-1-100	12.83	0.304	0.026	0.006	20.37	0.192	0.01	0.005
8	0.0977MB-1-100	26.98	0.362	0.019	0.007	47.13	0.207	0.011	0.004
9	0.1953MB-1-100	37.22	0.525	0.017	0.011	89.7	0.218	0.008	0.004
10	0.3906MB-1-100	48.53	0.805	0.025	0.015	137.6	0.284	0.01	0.004
11	1.0000MB-1-100	63.77	1.568	0.067	0.023	275.51	0.363	0.01	0.006
12	4.0000MB-1-100	81.33	4.918	0.2	0.063	232.08	1.724	0.035	0.024

图 23 s3proxy

可以看到相比 Minio，s3Proxy 在对象较小时，可以提供更高的吞吐率和更小的延迟，且写入效率比 Minio 更高，吞吐率更高和延迟更低。但是当对象尺寸较大时，其读取吞吐率不如 Minio。

下面看其对延迟的影响。

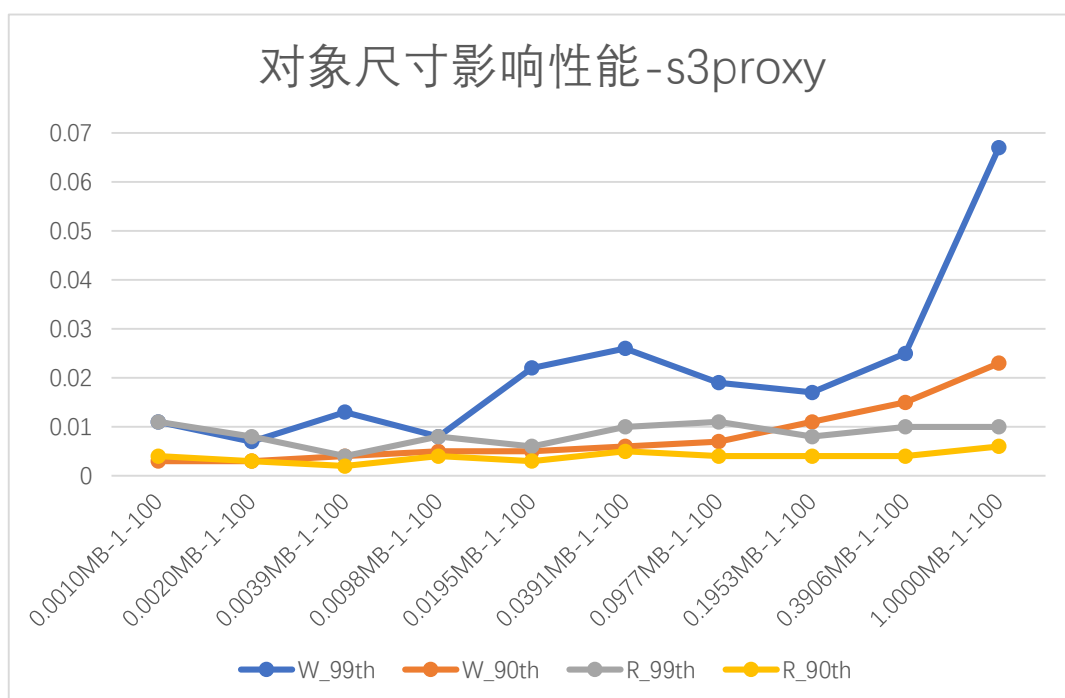


图 24 s3proxy 延迟

可以看到结论与 Minio 相似，对象尺寸越大，读写延迟越大，但是吞吐率也越大。

对于熟悉的某类应用，根据其数据访问特性，怎样适配对象存储最合适？

从上述结果来看，对象尺寸越小，读写延迟越小，但是吞吐率低。对象尺寸越大，读写延迟越高，但是吞吐率可以提高。

所以在我们常用的应用中，需要读写延迟较小的应用场景，应该选择对象尺寸小的对象存储，如搜索、查询等对延迟要求较高的场景。当需要吞吐率和速率时，应该选择对象尺寸较大的对象存储，如网盘、云相册等环境。

2. 连接数量对性能的影响

由于 s3bench 只能支持 1-100 的并发数量测试，所以无法测试超高并发的情况。并发测试的最大值只能为 100，所以我们选择的测试就为 1-100

首先我们观察 Minio 连接数对结果的影响

1	Size-Clients-Samples	W_Throughput(MB/s)	W_Duration(s)	W_99th	W_90th	R_Throughput(MB/s)	R_Duration(s)	R_99th	R_90th
13	0.0039MB-1-100	0.42	0.921	0.015	0.011	1.96	0.199	0.013	0.003
14	0.0039MB-2-100	0.82	0.476	0.023	0.012	2.57	0.152	0.015	0.007
15	0.0039MB-4-100	1.21	0.323	0.03	0.02	2.58	0.152	0.019	0.012
16	0.0039MB-8-100	1.75	0.223	0.054	0.024	3.09	0.126	0.033	0.02
17	0.0039MB-16-100	1.62	0.241	0.085	0.056	2.69	0.145	0.059	0.044
18	0.0039MB-32-100	1.61	0.243	0.162	0.12	2.97	0.131	0.086	0.065
19	0.0039MB-64-100	1.35	0.29	0.245	0.202	2.12	0.184	0.178	0.142
20	0.0039MB-70-100	1.32	0.296	0.244	0.213	2.36	0.166	0.139	0.115
21	0.0039MB-80-100	1.43	0.272	0.246	0.212	2.44	0.16	0.145	0.108
22	0.0039MB-90-100	1.41	0.277	0.256	0.243	1.44	0.272	0.233	0.189
23	0.0039MB-100-100	1	0.39	0.374	0.28	1.66	0.236	0.219	0.19
24	0.0977MB-2-100	16.68	0.586	0.028	0.018	46.1	0.212	0.017	0.009
25	0.0977MB-4-100	23.28	0.42	0.039	0.028	66.3	0.147	0.037	0.01
26	0.0977MB-8-100	25.91	0.377	0.074	0.044	62.63	0.156	0.042	0.019
27	0.0977MB-16-100	26.56	0.368	0.101	0.078	83.67	0.117	0.045	0.03
28	0.0977MB-32-100	27.87	0.35	0.148	0.119	80.48	0.121	0.093	0.055
29	0.0977MB-64-100	24.52	0.398	0.393	0.335	43.65	0.224	0.214	0.188

图 25 Minio

可以看到无论是在哪个对象大小，连接数越多，对象存储的延时越大。虽然吞吐率有所提高，但是每个连接存储每个对象的时间变长了，也就是服务器整体的性能有所下降。

下面观察 S3Proxy 的结果。

1	Size-Clients-Samples	W_Throughput(MB/s)	W_Duration(s)	W_99th	W_90th	R_Throughput(MB/s)	R_Duration(s)	R_99th	R_90th
13	0.0039MB-1-100	2.35	0.167	0.007	0.003	2.21	0.176	0.011	0.003
14	0.0039MB-2-100	2.92	0.134	0.015	0.006	2.83	0.138	0.012	0.006
15	0.0039MB-4-100	3.73	0.105	0.011	0.007	3.27	0.12	0.022	0.01
16	0.0039MB-8-100	3.36	0.116	0.031	0.015	3.22	0.121	0.025	0.012
17	0.0039MB-16-100	2.52	0.155	0.048	0.034	3.59	0.109	0.041	0.025
18	0.0039MB-32-100	2.2	0.177	0.084	0.063	4.31	0.091	0.04	0.029
19	0.0039MB-64-100	2.74	0.143	0.112	0.085	4.54	0.086	0.051	0.043
20	0.0039MB-70-100	1.55	0.252	0.164	0.138	1.79	0.218	0.202	0.178
21	0.0039MB-80-100	1.51	0.258	0.223	0.187	2.23	0.175	0.162	0.13
22	0.0039MB-90-100	1.66	0.236	0.102	0.09	3.19	0.122	0.103	0.088
23	0.0039MB-100-100	2.12	0.184	0.084	0.067	2.59	0.151	0.128	0.114
24	0.0977MB-2-100	35.5	0.275	0.016	0.012	65.26	0.15	0.014	0.005
25	0.0977MB-4-100	31.71	0.308	0.033	0.021	65.2	0.15	0.015	0.011
26	0.0977MB-8-100	36.21	0.27	0.075	0.043	73.07	0.134	0.038	0.02
27	0.0977MB-16-100	38.68	0.252	0.064	0.048	61.33	0.159	0.043	0.031
28	0.0977MB-32-100	32.56	0.3	0.182	0.105	51.68	0.189	0.089	0.068
29	0.0977MB-64-100	30.48	0.32	0.248	0.224	49.57	0.197	0.155	0.119

图 26 s3proxy

可以看到虽然 s3proxy 也同 Minio 一样，连接数越多，对象存储的延时越大。但是 s3Proxy 在高并发场景中，性能要大大优于 Minio。在读写延迟的性能指标中，s3Proxy 都优于 Minio。

综上，在高并发中，连接数量越多，对象存储的延迟越大。且 s3Proxy 要优于 Minio。

3. 对象数量对性能的影响

下面我们测试的是对象数量对性能的影响。下面分别测试 Minio 和 s3Proxy

1	Size-Clients-Samples	W_Throughput(MB/s)	W_Duration(s)	W_99th	W_90th	R_Throughput(MB/s)	R_Duration(s)	R_99th	R_90th
30	0.0039MB-1-5	0.33	0.06	0.019	0.019	1.52	0.013	0.004	0.004
31	0.0039MB-1-10	0.44	0.089	0.011	0.011	1.45	0.027	0.007	0.007
32	0.0039MB-1-20	0.4	0.193	0.016	0.015	1.73	0.045	0.008	0.003
33	0.0039MB-1-40	0.43	0.363	0.014	0.012	2.03	0.077	0.006	0.003
34	0.0039MB-1-80	0.41	0.753	0.016	0.013	1.4	0.223	0.023	0.005
35	0.0039MB-1-160	0.42	1.495	0.02	0.013	2.15	0.291	0.007	0.003
36	0.0039MB-1-320	0.43	2.901	0.015	0.01	2.3	0.543	0.008	0.002
37	0.0039MB-1-640	0.43	5.84	0.015	0.01	2.25	1.113	0.006	0.002
38	0.0039MB-1-1280	0.43	11.721	0.016	0.011	2.54	1.972	0.006	0.002

图 27 Minio

1	Size-Clients-Samples	W_Throughput(MB/s)	W_Duration(s)	W_99th	W_90th	R_Throughput(MB/s)	R_Duration(s)	R_99th	R_90th
30	0.0039MB-1-5	1.15	0.017	0.01	0.01	2.2	0.009	0.003	0.003
31	0.0039MB-1-10	0.78	0.05	0.02	0.02	2.22	0.018	0.005	0.005
32	0.0039MB-1-20	2.28	0.034	0.005	0.005	3.16	0.025	0.003	0.002
33	0.0039MB-1-40	1.47	0.106	0.013	0.006	2.2	0.071	0.005	0.004
34	0.0039MB-1-80	1.93	0.162	0.01	0.004	2.47	0.126	0.006	0.003
35	0.0039MB-1-160	2.46	0.254	0.006	0.003	2.44	0.256	0.01	0.003
36	0.0039MB-1-320	2.54	0.492	0.006	0.002	2.76	0.453	0.009	0.002
37	0.0039MB-1-640	2.87	0.87	0.005	0.002	3.43	0.729	0.004	0.001
38	0.0039MB-1-1280	3.04	1.644	0.005	0.002	3.33	1.499	0.004	0.001

图 28 s3Proxy

可以看到 s3Proxy 比起 minio 其吞吐率和延迟都好很多。总的来说，对象数量对性能的影响相对来说不是太大，随着对象数量的倍增，两个服务端的延时也没有倍增，即对延时的影响不是很大。这是因为客户端只是串行的将数据传输给服务器，对服务器的性能影响不大。

4. I/O 延迟背后的关键影响要素?

根据上面的 3 个分析，可以得出以下的结论：

I/O 延迟受多个方面因素的影响，其中主要受对象大小和连接数量。对象大小越大，IO 延迟越大；连接数量越多，IO 延迟也越大。

对象大小的影响其实是由于最底层的磁盘延迟。由于要从磁盘出读出较大的数据，磁盘要读取很多块，所以导致延迟变高了。

而连接数量的影响主要在于很多客户端并发请求，导致服务端出现拥塞，出现了排队现象。服务器需要排队处理每个连接所请求的信息，导致平均延迟就变高了。

六、实验总结

这次实验的过程相对来说比较简单，而且实验内容也非常贴近实际，都是现在用到的新技术和对我们以后非常实用的技能。比如说 `git`，虽然我在之前就一直使用 `git` 管理代码，但是这次实验也让我们体会到了如何合作开发项目，如何使用 `pull request`，也让更多的同学学到了 `git` 的使用方法。

在这次实验搭建服务端时，我发现所有服务端几乎都可以用 `docker` 一键部署，所以我配置安装了 `docker`，后面的服务端配置就十分简单了。但是在配置过程中也遇到了一些问题，如给 `docker` 镜像命名，方便下次启动；`docker` 容器内外的端口映射问题等等。在配置 `s3proxy` 时，由于默认的端口为 80 端口，但是我的环境中装有 `Apache`，已经将 80 端口占用了，导致服务无法使用。后来查阅资料后发现，可以将容器内端口映射到不同的端口上去，成功解决了这个问题。

在使用 `s3cmd` 出现了一个奇怪的 `bug`，在不输入密钥时，提示找不到配置文件。我一开始以为是文件权限的问题，后来更改权限后发现这个问题依旧存在。后来我又尝试另一个客户端 `aws`，也不能正确连接 `s3Proxy` 服务器。和老师交流后，发现是因为没有输入密钥，导致其配置文件解析错误，提示了错误的信息。这可以算是 `s3cmd` 的一个小 `bug` 吧，让我苦恼了很久。

这个实验让我们体会到了目前应用的对象存储服务器、客户端、评测工具的使用，还发现了 `docker` 这个非常好用的工具，学会了他的用途，收获非常之大

最后，感谢老师、同学对我的帮助，希望这个实验越办越好！

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.