華中科技大學

数据中心课程设计报告

院　　系　计算机科学与技术学院

班　　级　硕 2110 班

学　　号　M202173849

姓　　名　熊益

2022 年　01 月　07 日

一、系统搭建

1. 实验环境

操作系统：Windows 10 家庭中文版
处理器：11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz
RAM：16GB
开发语言：Python

2. 服务端搭建

首先下载 Minio，并创建两个密钥 RootUser 和 RootPassword，获得对象存储资源访问的授权，在本次实验中设置为’admin’和‘12345678’。

运行 Minio 时，将其工作在服务器模式，并将配置文件存入 Minio 所在的目录中，如图 1 所示。

```
D:\download\minio\minio_server>cd D:\download\minio\minio_server

D:\download\minio\minio_server>set MINIO_ROOT_USER=admin

D:\download\minio\minio_server>set MINIO_ROOT_PASSWORD=12345678

D:\download\minio\minio_server>set MINIO_PROMETHEUS_AUTH_TYPE=public

D:\download\minio\minio_server>minio.exe -C ./ server D:\download\minio\minio_server\data --console-address ":9090"
API: http://10.21.174.117:9000  http://127.0.0.1:9000
RootUser: admin
RootPass: 12345678

Console: http://10.21.174.117:9090 http://127.0.0.1:9090
RootUser: admin
RootPass: 12345678

Command-line: https://docs.min.io/docs/minio-client-quickstart-guide
   $ mc.exe alias set myminio http://10.21.174.117:9000 admin 12345678

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

图 1 运行服务器

本次实验绑定的 IP 地址为 http://10.21.174.117:9000、http://127.0.0.1:9000，能通过其进行访问，如图 2 所示。

在网页上建一个 bucket：loadgen，如图 3 所示。

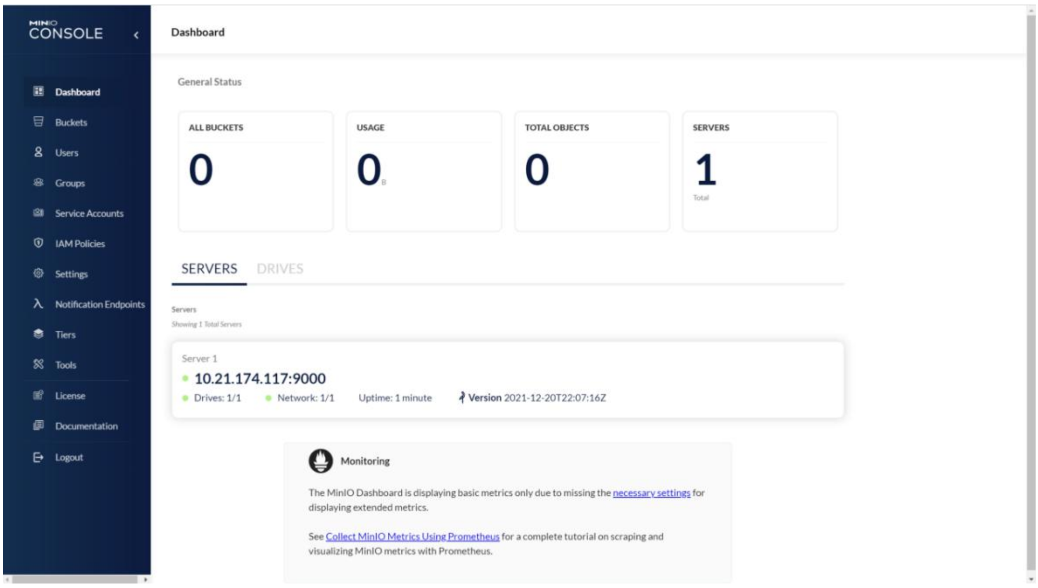


图 2 利用 IP 地址访问服务器

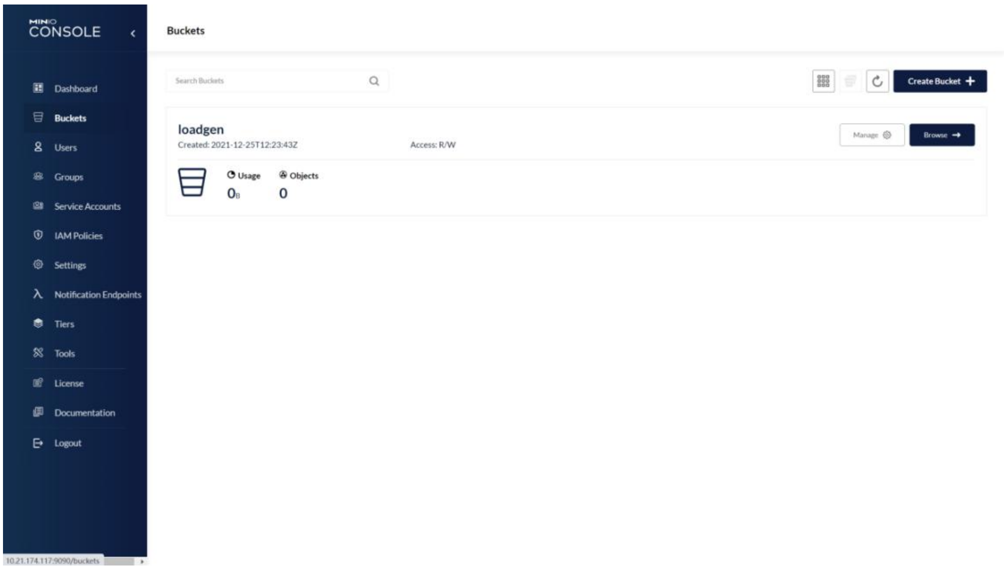


图 3 在网页上创建一个新的 bucket

二、性能观测

利用测评工具 S3 bench，在 bucket loadgen 中进行跑分，避免与用户数据重叠在一起，将 accessKey 和 accessSecret 修改为预设的'admin'和'12345678'，endpoint 修改问本次实验绑定的 IP 地址。

初步设置为 8 个客户端，实验结果如图 4 所示。

从实验结果中观察到，8 个客户端的负载较轻，无法从数据中看出规律性，因此需要加重负载。

加客户端数量由 8 个上调至 32 个，实验结果如图 5 所示。

观察读延迟的数据，我们能发现，有 50%的请求在 0.003s 内完成，但 90%的请求在 0.006s 内完成，100%的请求在 0.009s 内完成，即尾延迟现象，在写延迟中我们也能看到相同的的现象。

```
D:\download\minio\minio_server\data>s3bench.exe ^
More? -accessKey=admin ^
More? -accessSecret=12345678 ^
More? -bucket=loadgen ^
More? -endpoint=http://127.0.0.1:9000 ^
More? -numClients=8 ^
More? -numSamples=256 ^
More? -objectNamePrefix=loadgen ^
More? -objectSize=1024
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.0010 MB
numClients:       8
numSamples:       256
verbose:       %!d(bool=false)


Generating in-memory sample data... Done (1.0017ms)

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.0010 MB
numClients:       8
numSamples:       256
verbose:       %!d(bool=false)


Results Summary for Write Operation(s)
Total Transferred: 0.250 MB
Total Throughput:  0.29 MB/s
Total Duration:    0.866 s
Number of Errors:  0
------------------------------------
Write times Max:       0.084 s
Write times 99th %ile: 0.081 s
Write times 90th %ile: 0.051 s
Write times 75th %ile: 0.039 s
Write times 50th %ile: 0.020 s
Write times 25th %ile: 0.009 s
Write times Min:       0.004 s


Results Summary for Read Operation(s)
Total Transferred: 0.250 MB
Total Throughput:  10.36 MB/s
Total Duration:    0.024 s
Number of Errors:  0
------------------------------------
Read times Max:       0.003 s
Read times 99th %ile: 0.003 s
Read times 90th %ile: 0.001 s
Read times 75th %ile: 0.001 s
Read times 50th %ile: 0.001 s
Read times 25th %ile: 0.001 s
Read times Min:       0.000 s


Cleaning up 256 objects...
Deleting a batch of 256 objects in range {0, 255}... Succeeded
Successfully deleted 256/256 objects in 107.3407ms
```

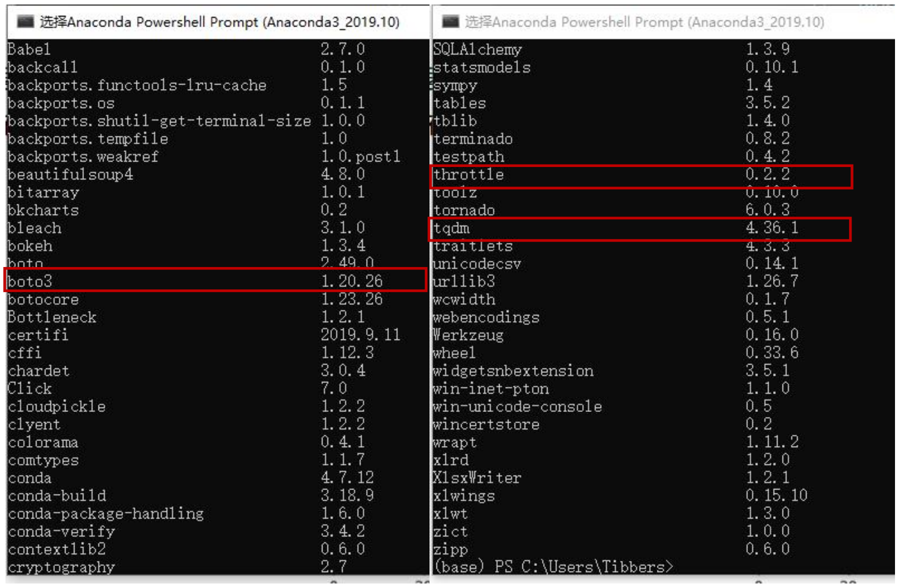
图 4 设置 8 个客户端

```
D:\download\minio\minio_server\data>s3bench.exe ^
More? -accessKey=admin ^
More? -accessSecret=12345678 ^
More? -bucket=loadgen ^
More? -endpoint=http://127.0.0.1:9000 ^
More? -numClients=32 ^
More? -numSamples=256 ^
More? -objectNamePrefix=loadgen ^
More? -objectSize=1024
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.0010 MB
numClients:       32
numSamples:       256
verbose:       %!d(bool=false)


Generating in-memory sample data... Done (0s)

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.0010 MB
numClients:       32
numSamples:       256
verbose:       %!d(bool=false)


Results Summary for Write Operation(s)
Total Transferred: 0.250 MB
Total Throughput:  0.13 MB/s
Total Duration:    1.902 s
Number of Errors:  0
------------------------------------
Write times Max:       0.438 s
Write times 99th %ile: 0.383 s
Write times 90th %ile: 0.290 s
Write times 75th %ile: 0.261 s
Write times 50th %ile: 0.220 s
Write times 25th %ile: 0.197 s
Write times Min:       0.010 s


Results Summary for Read Operation(s)
Total Transferred: 0.250 MB
Total Throughput:  10.60 MB/s
Total Duration:    0.024 s
Number of Errors:  0
------------------------------------
Read times Max:       0.009 s
Read times 99th %ile: 0.006 s
Read times 90th %ile: 0.004 s
Read times 75th %ile: 0.004 s
Read times 50th %ile: 0.003 s
Read times 25th %ile: 0.002 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 106.659ms
```

图 5 上调至 32 个客户端

三、观测尾延迟

安装 Anaconda，并将其配置到 VS Code 环境中，在运行 latency-collect 代码前，需要在 Anaconda 中使用 pip install 指令安装 botocore、tqdm、throttle 包，如图 6 所示。



Babel	2.7.0
backcall	0.1.0
backports.functools-lru-cache	1.5
backports.os	0.1.1
backports.shutil-get-terminal-size	1.0.0
backports.tempfile	1.0
backports.weakref	1.0.post1
beautifulsoup4	4.8.0
bitarray	1.0.1
bkcharts	0.2
bleach	3.1.0
bokeh	1.3.4
boto	2.49.0
boto3	1.20.26
botocore	1.23.26
Bottleneck	1.2.1
certifi	2019.9.11
cffi	1.12.3
chardet	3.0.4
Click	7.0
cloudpickle	1.2.2
clyent	1.2.2
colorama	0.4.1
comtypes	1.1.7
conda	4.7.12
conda-build	3.18.9
conda-package-handling	1.6.0
conda-verify	3.4.2
contextlib2	0.6.0
cryptography	2.7

SQLAlchemy	1.3.9
statsmodels	0.10.1
sympy	1.4
tables	3.5.2
tblib	1.4.0
terminado	0.8.2
testpath	0.4.2
throttle	0.2.2
toolz	0.10.0
tornado	6.0.3
tqdm	4.36.1
traitlets	4.3.3
unicodecsv	0.14.1
urllib3	1.26.7
wcwidth	0.1.7
webencodings	0.5.1
Werkzeug	0.16.0
wheel	0.33.6
widgetsnbextension	3.5.1
win-inet-pton	1.1.0
win-unicode-console	0.5
wincertstore	0.2
wrapt	1.11.2
xlrd	1.2.0
XlsxWriter	1.2.1
xlwings	0.15.10
xlwt	1.3.0
zict	1.0.0
zipp	0.6.0

(base) PS C:\Users\Tibbers>

图 6 Anaconda 安装包

在 python 环境中建立与服务器的对话。

```python
# 准备密钥
aws_access_key_id = 'admin'
aws_secret_access_key = '12345678'

# 本地S3服务地址
local_s3 = 'http://127.0.0.1:9000'

# 建立会话
session = Session(aws_access_key_id=aws_access_key_id, aws_secret_access_key=aws_secret_access_key)

# 连接到服务
s3 = session.resource('s3', endpoint_url=local_s3)
```

新建一个实验用的 bucket，并查看所有 bucket

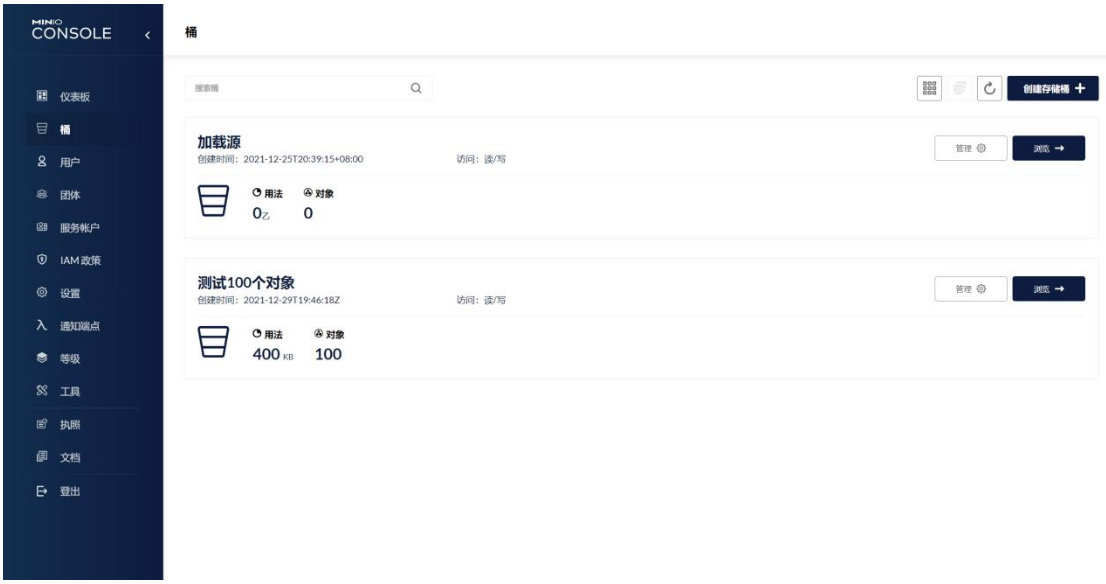
```python
#新建一个实验用 bucket (注意: "bucket name" 中不能有下划线)
bucket_name = 'test100objs'
if s3.Bucket(bucket_name) not in s3.buckets.all():
    s3.create_bucket(Bucket=bucket_name)

#查看所有pocket
for bucket in s3.buckets.all():
    print('bucket name:%s' % bucket.name)

```

问题 输出 调试控制台 终端 Code

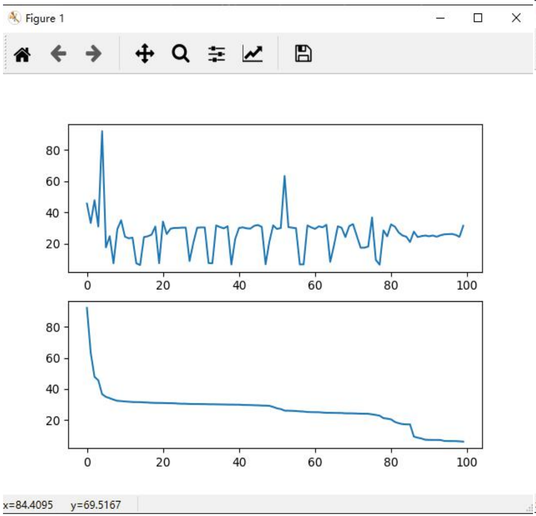
```
[Running] python -u "d:\download\vscode\workplace\test\test.py"
bucket name:loadgen
bucket name:test100objs

[Done] exited with code=0 in 0.329 seconds
```

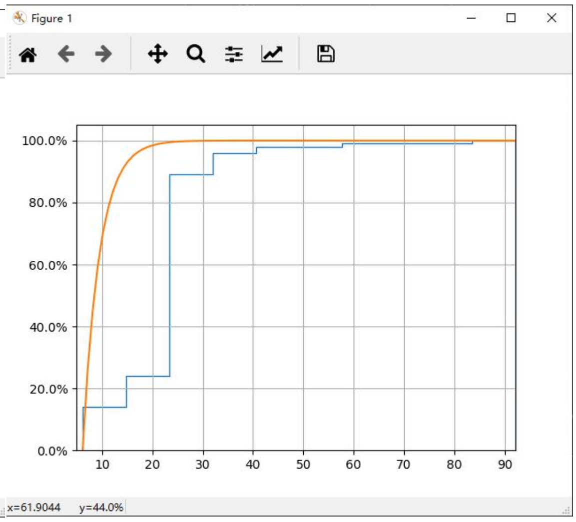
准备负载，并按照与设 IAT 发起请求

```
[Running] python -u "d:\download\vscode\workplace\test\test.py"
bucket name:loadgen
bucket name:test100objs

Accessing S3:   0%|          | 0/100 [00:00<?, ?it/s]
Accessing S3:   1%|          | 1/100 [00:00<00:35,  2.81it/s]
Accessing S3:  16%|����        | 16/100 [00:00<00:21,  3.97it/s]
Accessing S3:  20%|����        | 20/100 [00:00<00:14,  5.40it/s]
Accessing S3:  24%|������       | 24/100 [00:00<00:10,  7.19it/s]
Accessing S3:  28%|������       | 28/100 [00:00<00:07,  9.37it/s]
Accessing S3:  35%|��������      | 35/100 [00:00<00:05, 12.49it/s]
Accessing S3:  40%|��������      | 40/100 [00:01<00:03, 15.29it/s]
Accessing S3:  44%|����������     | 44/100 [00:01<00:03, 18.04it/s]
Accessing S3:  48%|����������     | 48/100 [00:01<00:02, 20.80it/s]
Accessing S3:  52%|������������    | 52/100 [00:01<00:01, 24.24it/s]
Accessing S3:  56%|������������    | 56/100 [00:01<00:01, 26.48it/s]
Accessing S3:  61%|������������    | 61/100 [00:01<00:01, 29.59it/s]
Accessing S3:  66%|��������������   | 66/100 [00:01<00:01, 32.20it/s]
Accessing S3:  70%|��������������   | 70/100 [00:01<00:00, 32.16it/s]
Accessing S3:  74%|����������������  | 74/100 [00:02<00:00, 31.87it/s]
Accessing S3:  78%|����������������  | 78/100 [00:02<00:00, 32.11it/s]
Accessing S3:  82%|������������������ | 82/100 [00:02<00:00, 32.27it/s]
Accessing S3:  88%|������������������ | 88/100 [00:02<00:00, 35.60it/s]
Accessing S3:  92%|████████████████████| 92/100 [00:02<00:00, 34.70it/s]
Accessing S3:  97%|████████████████████| 97/100 [00:02<00:00, 36.23it/s]
Accessing S3: 100%|████████████████████| 100/100 [00:02<00:00, 35.76it/s]

[Done] exited with code=0 in 3.156 seconds
```

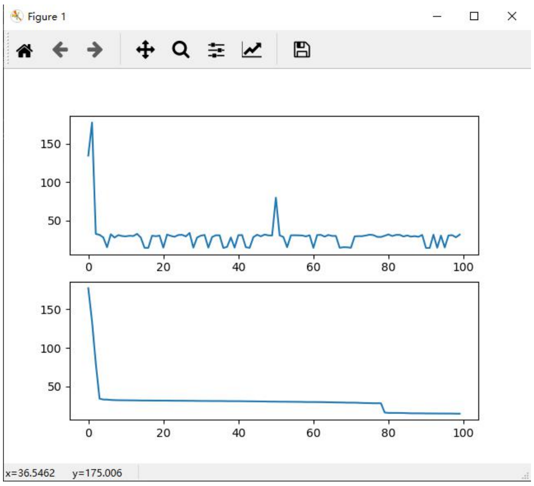


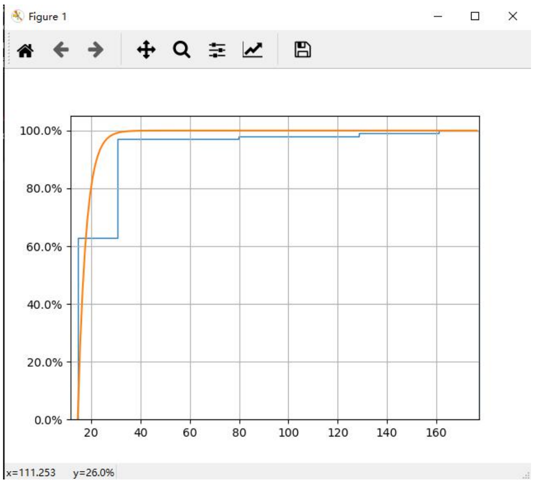
当设置文件设置为 4k，并将 latency 存入文件并，并运行 latency-plot 的代码，将数据绘制为图表，下图为是实验结果：





将文件大小上调至 32k 后的实验结果为:





左边图表中的上方图表将请求依次打印出来，我们可以看到前期的波动有剧烈的波动，中间也存在较大波动的地方，这些地方的延迟超过了远平均延迟之上。下方表格将其排序，能看出只有少部分的延迟较高，大部分地方的延迟为短时间延迟。

右侧的表格可以看出多达 65%的请求能够早 10ms 以内完成，但 90%的请求要在 30ms 以内完成，若请求全部完成，则需要 160ms，这也就是实验中的长尾延迟现象。