華中科技大學

数据中心技术实验报告

姓　　名：	陈倩
学　　院：	计算机科学与技术学院
专　　业：	电子信息
班　　级：	2110
学　　号：	M202173864
指导教师：	施展、童薇

分数	
教师签名	

2021 年 12 月 26 日

目　录

1　实验目的及实验内容 1

2　实验一：系统搭建 2

2.1　实验环境 2

2.2　基础环境 2

2.3　服务端搭建 2

3　实验二：性能观测 3

3.1　评测工具 3

3.2　标准测试 3

3.2.1　对象尺寸对性能的影响 3

3.2.2　客户端数量对性能的影响 6

4　实验三：尾延迟测试 12

4.1　实验环境 12

4.2　延迟收集 13

4.3　尾延迟展示 13

1 实验目的及实验内容

对象存储（Object Storage Service，OSS），也叫基于对象的存储，是一种解决和处理离散单元的方法，可提供基于分布式系统之上的对象形式的数据存储服务。对象存储和我们经常接触到的块和文件系统等存储形态不同，它提供RESTful API数据读写接口及丰富的SDK接口，并且常以网络服务的形式提供数据的访问。作为新数据时代的新存储形态，相比SAN和NAS，对象存储直接提供API给应用使用，采用扁平化的结构管理所有桶（Bucket）和对象。每个桶和对象都有一个全局唯一的ID，根据ID可快速实现对象的查找和数据的访问。对象存储支持基于策略的自动化管理机制，使得每个应用可根据业务需要动态地控制每个桶的数据冗余策略、数据访问权限控制及数据生命周期管理。这些优势使得对象存储具备极致的扩展和极易的数据管理。本实验是对象存储的一个入门实践。

实验内容包括：

1. 熟悉性能指标：吞吐率、带宽、延迟
2. 分析不同负载下的指标、延迟的分布
3. 观测尾延迟现象
4. 尝试对冲请求方案

2 实验一：系统搭建

2.1 实验环境

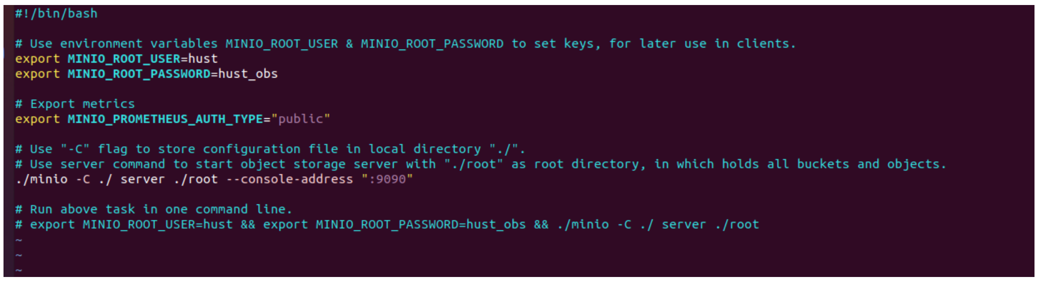
VMware 版本：VMware Workstation 16 Player 16.1.2
Linux 版本：5.11.0-37-generic

2.2 基础环境

Python 版本：3.8.10
Go 版本：go1.10.3 linux/amd64jupyter
Jupyter notebook 版本：6.4.6

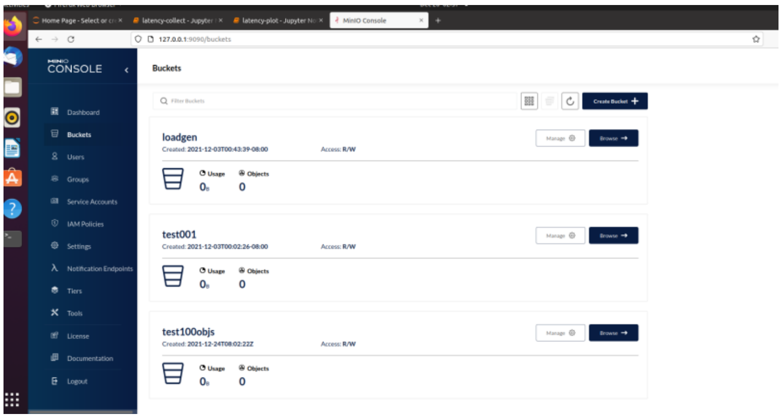
2.3 服务端搭建

首先将脚本（https://gitee.com/shi_zhan/obs-tutorial）克隆到本地，从 minio 官网下载 Linux 版本的 minio 到 obs-tutorial 文件夹，然后进入 obs-tutorial 文件，在命令行键入./run-minio.sh 启动 minio。run-minio.sh 指定 minio 用户名和密码分别为 hust 和 hust_obs，端口为 9090。启动 minio 后，在浏览器内输入 127.0.0.1：9090 打开 minio 登录界面，输入用户名和密码后，进入 minio 主页。



```
#!/bin/bash

# Use environment variables MINIO_ROOT_USER & MINIO_ROOT_PASSWORD to set keys, for later use in clients.
export MINIO_ROOT_USER=hust
export MINIO_ROOT_PASSWORD=hust_obs

# Export metrics
export MINIO_PROMETHEUS_AUTH_TYPE="public"

# Use "-C" flag to store configuration file in local directory "./".
# Use server command to start object storage server with "./root" as root directory, in which holds all buckets and objects.
./minio -C ./ server ./root --console-address ":9090"

# Run above task in one command line.
# export MINIO_ROOT_USER=hust && export MINIO_ROOT_PASSWORD=hust_obs && ./minio -C ./ server ./root
~
~
~
```

图 2-1 run-minio.sh



Home Page - Select or cr | latency-collect - Jupyter | latency-plot - Jupyter No | MinIO Console

127.0.0.1:9090/buckets

Buckets

loadgen
Created: 2021-12-03T00:43:39-08:00　Access: R/W
Usage 0B　Objects 0

test001
Created: 2021-12-03T00:02:26-08:00　Access: R/W
Usage 0B　Objects 0

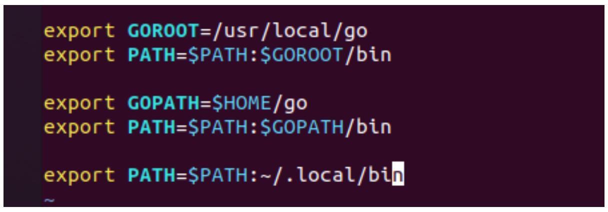
test100objs
Created: 2021-12-24T08:02:22Z　Access: R/W
Usage 0B　Objects 0

图 2-2 minio 主页

3 实验二：性能观测

3.1 评测工具

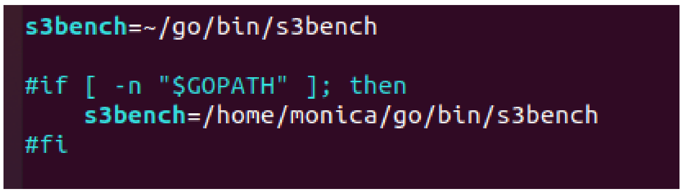
本实验选择 S3 Bench 作为评测工具。在安装 S3 Bench 之前，需要先安装 Go，首先去官网下载最新版 Golang 的安装包，下载完毕后，解压到/usr/local 下。然后编辑/etc/profile 文件配置环境变量。



```
export GOROOT=/usr/local/go
export PATH=$PATH:$GOROOT/bin

export GOPATH=$HOME/go
export PATH=$PATH:$GOPATH/bin

export PATH=$PATH:~/.local/bin
```

图 3-1 Golang 环境变量

配置好 Go 后，在命令行键入 go get -u github.com/igneous-systems/s3bench 下载 s3bench，s3bench 的默认安装路径为~/go/bin/s3bench。安装完成后，将 obs-tutorial 下的 run-s3bench.sh 的 s3bench 路径修改为~/go/bin/s3bench。

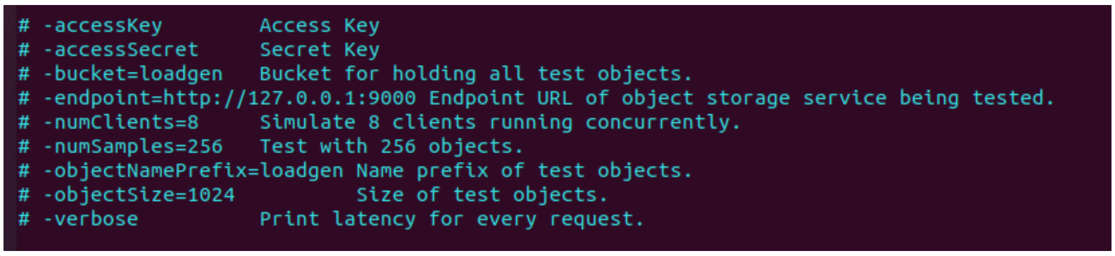


```
s3bench=~/go/bin/s3bench

#if [ -n "$GOPATH" ]; then
    s3bench=/home/monica/go/bin/s3bench
#fi
```

图 3-2 s3bench 路径

3.2 标准测试

指标：吞吐率 Throughput、延迟 Latency、对象尺寸 ObjectSize、客户端数量、服务器数量等。

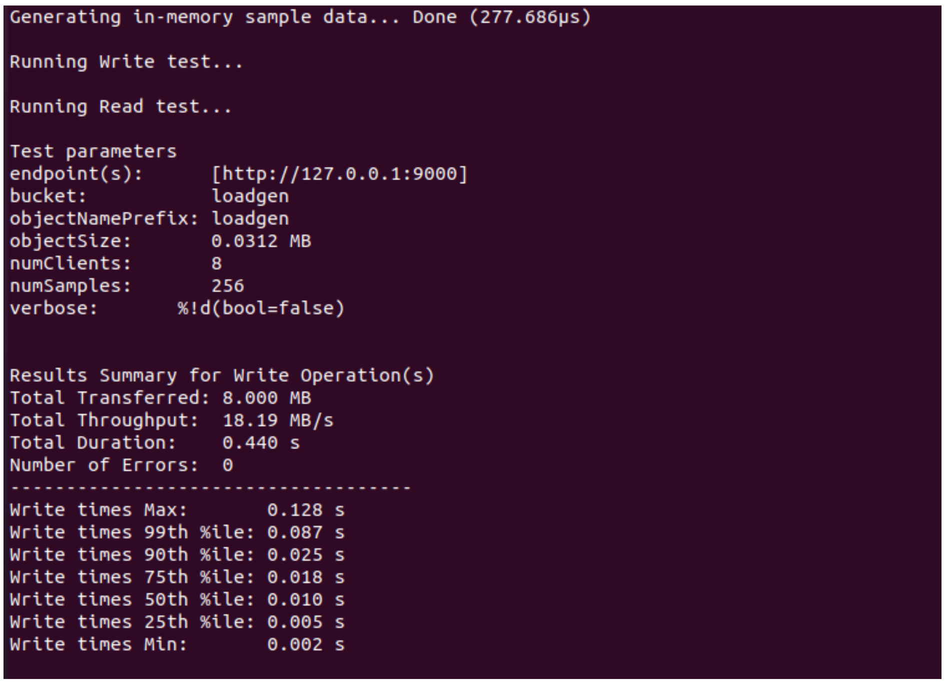


```
# -accessKey        Access Key
# -accessSecret     Secret Key
# -bucket=loadgen   Bucket for holding all test objects.
# -endpoint=http://127.0.0.1:9000 Endpoint URL of object storage service being tested.
# -numClients=8     Simulate 8 clients running concurrently.
# -numSamples=256   Test with 256 objects.
# -objectNamePrefix=loadgen Name prefix of test objects.
# -objectSize=1024          Size of test objects.
# -verbose          Print latency for every request.
```

图 3-3 s3bench 各项参数涵义

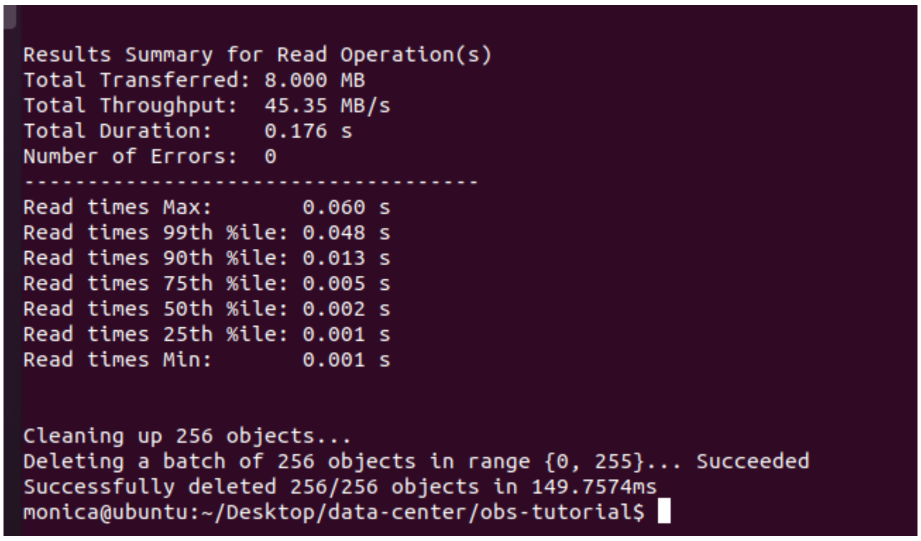
3.2.1 对象尺寸对性能的影响

通过修改 run-s3bench.sh 的 ObjectSize 参数来改变对象尺寸，实验选择的对象尺寸为 1024*32、1024*320 和 1024*3200.



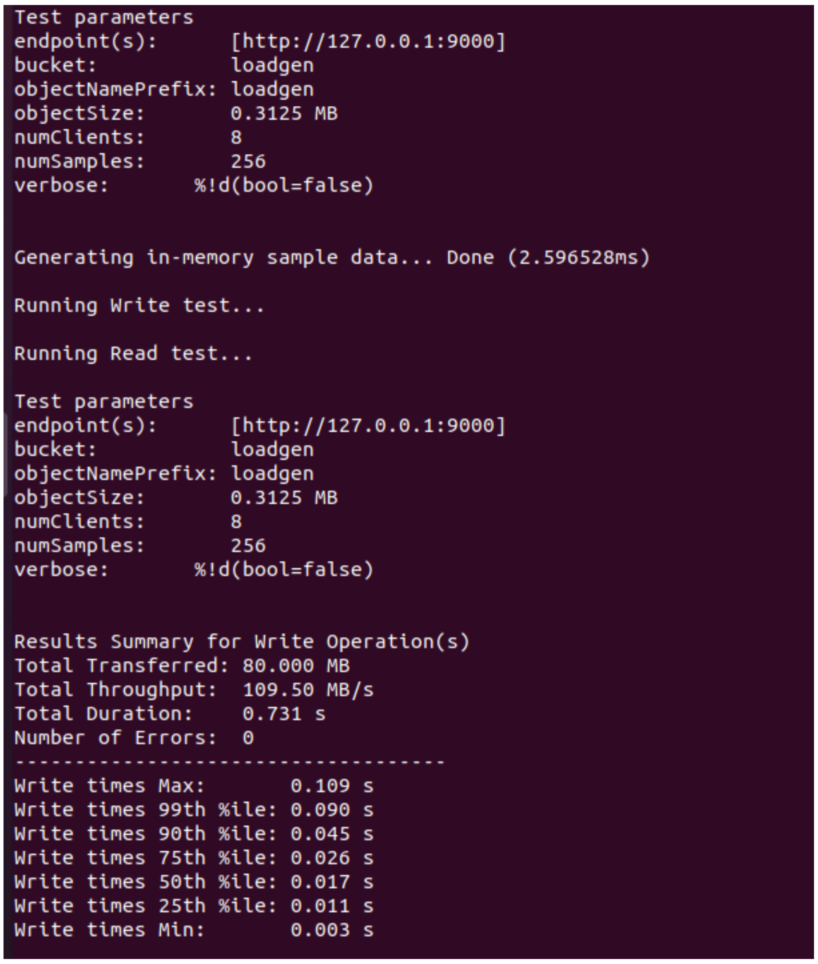
```
Generating in-memory sample data... Done (277.686µs)

Running Write test...

Running Read test...

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:  18.19 MB/s
Total Duration:    0.440 s
Number of Errors:  0
------------------------------------
Write times Max:       0.128 s
Write times 99th %ile: 0.087 s
Write times 90th %ile: 0.025 s
Write times 75th %ile: 0.018 s
Write times 50th %ile: 0.010 s
Write times 25th %ile: 0.005 s
Write times Min:       0.002 s
```

图 3-4 ObjectSize=(1024*32)(a)



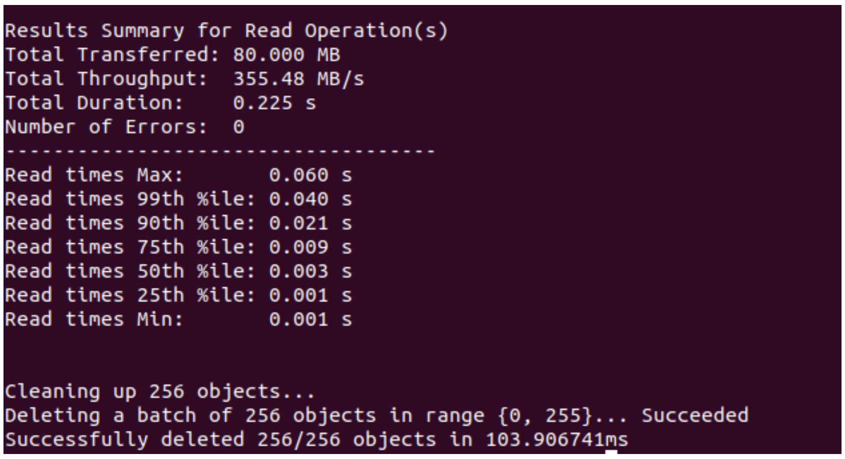
```
Results Summary for Read Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  45.35 MB/s
Total Duration:    0.176 s
Number of Errors:  0
------------------------------------
Read times Max:       0.060 s
Read times 99th %ile: 0.048 s
Read times 90th %ile: 0.013 s
Read times 75th %ile: 0.005 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 149.7574ms
monica@ubuntu:~/Desktop/data-center/obs-tutorial$ 
```

图 3-5 ObjectSize=(1024*32)(b)



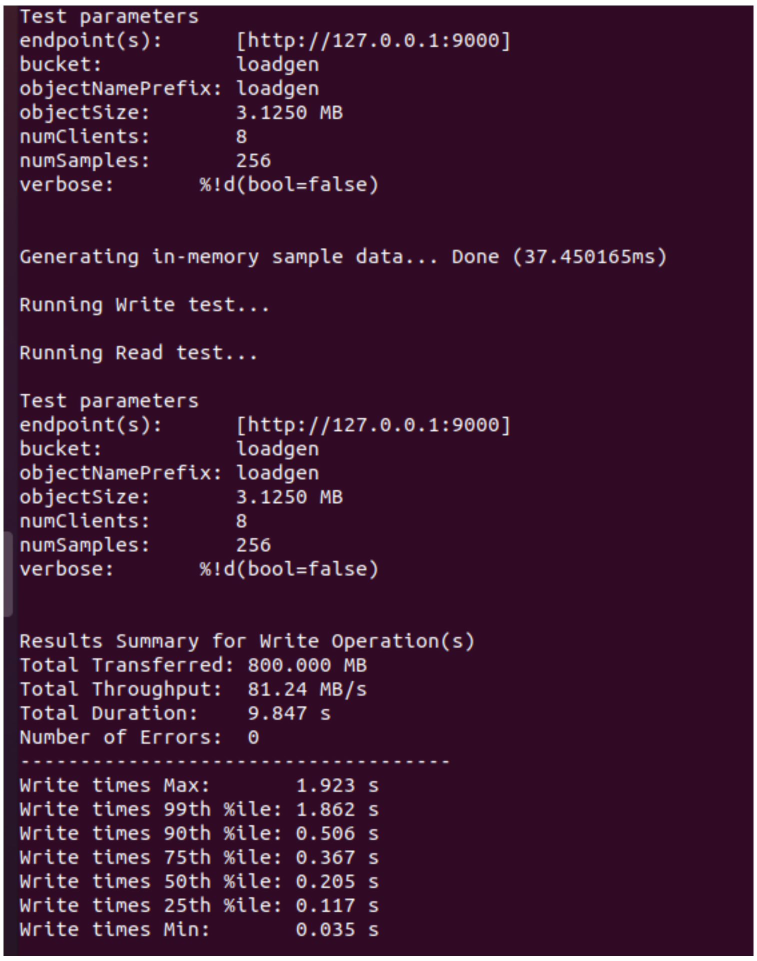
```
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.3125 MB
numClients:       8
numSamples:       256
verbose:       %!d(bool=false)


Generating in-memory sample data... Done (2.596528ms)

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.3125 MB
numClients:       8
numSamples:       256
verbose:       %!d(bool=false)


Results Summary for Write Operation(s)
Total Transferred: 80.000 MB
Total Throughput:  109.50 MB/s
Total Duration:    0.731 s
Number of Errors:  0
------------------------------------
Write times Max:       0.109 s
Write times 99th %ile: 0.090 s
Write times 90th %ile: 0.045 s
Write times 75th %ile: 0.026 s
Write times 50th %ile: 0.017 s
Write times 25th %ile: 0.011 s
Write times Min:       0.003 s
```

图 3-6 ObjectSize=(1024*320)(a)



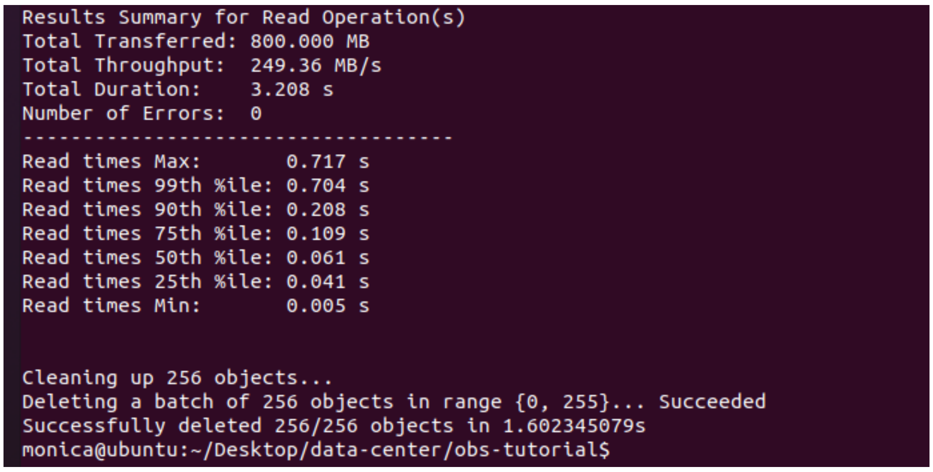
```
Results Summary for Read Operation(s)
Total Transferred: 80.000 MB
Total Throughput:  355.48 MB/s
Total Duration:    0.225 s
Number of Errors:  0
------------------------------------
Read times Max:       0.060 s
Read times 99th %ile: 0.040 s
Read times 90th %ile: 0.021 s
Read times 75th %ile: 0.009 s
Read times 50th %ile: 0.003 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 103.906741ms
```

图 3-7 ObjectSize=(1024*320)(b)



```
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       3.1250 MB
numClients:       8
numSamples:       256
verbose:       %!d(bool=false)


Generating in-memory sample data... Done (37.450165ms)

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       3.1250 MB
numClients:       8
numSamples:       256
verbose:       %!d(bool=false)


Results Summary for Write Operation(s)
Total Transferred: 800.000 MB
Total Throughput:  81.24 MB/s
Total Duration:    9.847 s
Number of Errors:  0
------------------------------------
Write times Max:       1.923 s
Write times 99th %ile: 1.862 s
Write times 90th %ile: 0.506 s
Write times 75th %ile: 0.367 s
Write times 50th %ile: 0.205 s
Write times 25th %ile: 0.117 s
Write times Min:       0.035 s
```

图 3-8 ObjectSize=(1024*3200)(a)



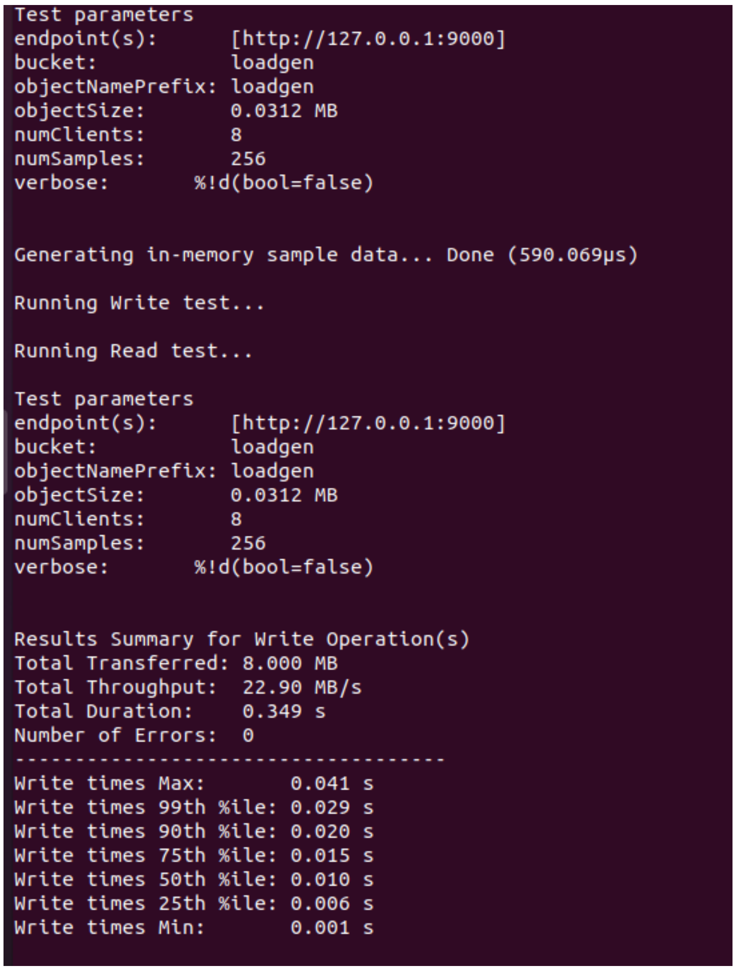
```
Results Summary for Read Operation(s)
Total Transferred: 800.000 MB
Total Throughput:  249.36 MB/s
Total Duration:    3.208 s
Number of Errors:  0
------------------------------------
Read times Max:       0.717 s
Read times 99th %ile: 0.704 s
Read times 90th %ile: 0.208 s
Read times 75th %ile: 0.109 s
Read times 50th %ile: 0.061 s
Read times 25th %ile: 0.041 s
Read times Min:       0.005 s


Cleaning up 256 objects...
Deleting a batch of 256 objects in range {0, 255}... Succeeded
Successfully deleted 256/256 objects in 1.602345079s
monica@ubuntu:~/Desktop/data-center/obs-tutorial$
```

图 3-9 ObjectSize=(1024*3200)(b)

通过实验对比发现，对象尺寸越大，吞吐量、延迟也会增大（也受网络情况影响），同时，长尾效应也会更加严重。

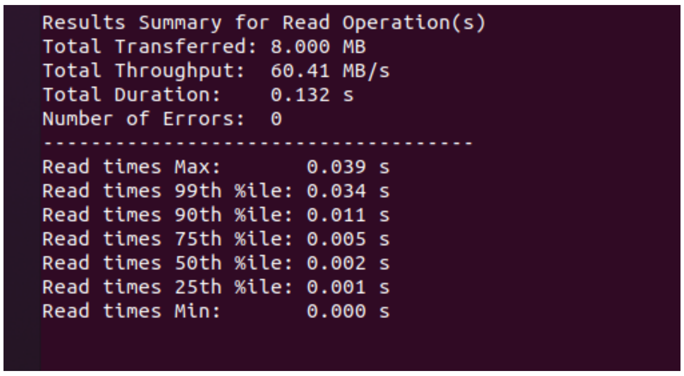
3.2.2 客户端数量对性能的影响

通过修改 run-s3bench.sh 的 numClients 参数来改变客户端数，实验选择的客户端数为 8、32、64、128 和 256.



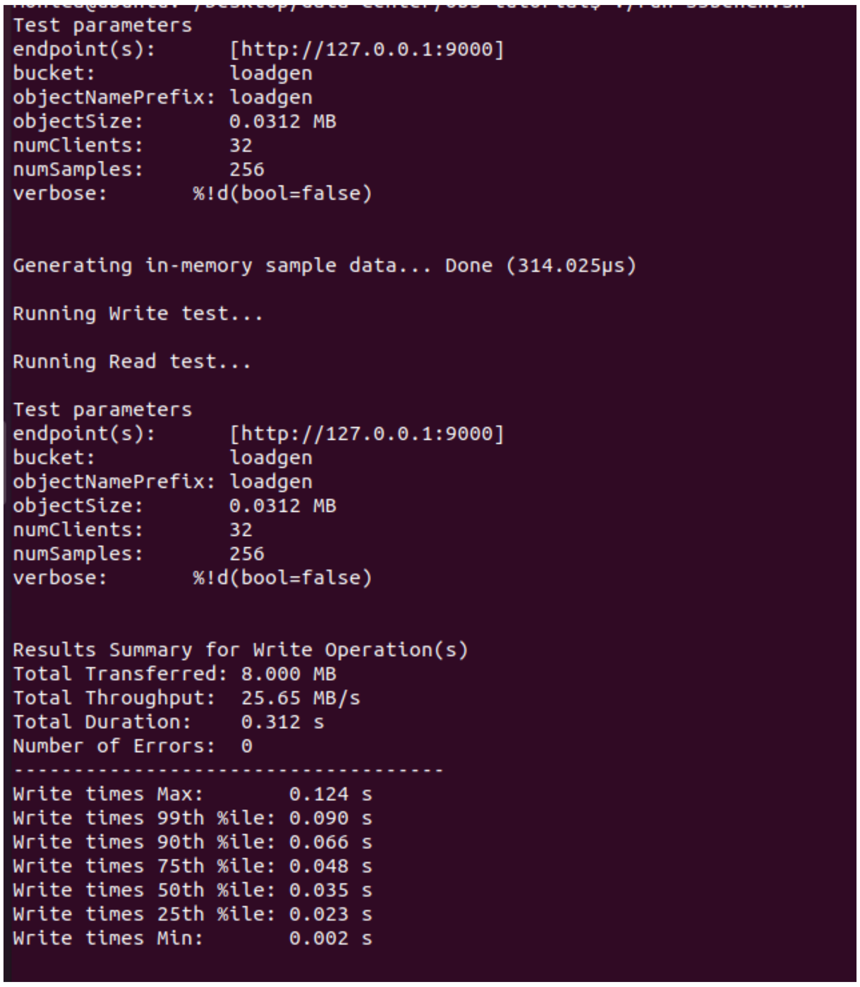
```
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.0312 MB
numClients:       8
numSamples:       256
verbose:       %!d(bool=false)


Generating in-memory sample data... Done (590.069µs)

Running Write test...

Running Read test...

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:  22.90 MB/s
Total Duration:    0.349 s
Number of Errors:  0
------------------------------------
Write times Max:       0.041 s
Write times 99th %ile: 0.029 s
Write times 90th %ile: 0.020 s
Write times 75th %ile: 0.015 s
Write times 50th %ile: 0.010 s
Write times 25th %ile: 0.006 s
Write times Min:       0.001 s
```

图 3-10 numClients=8(a)



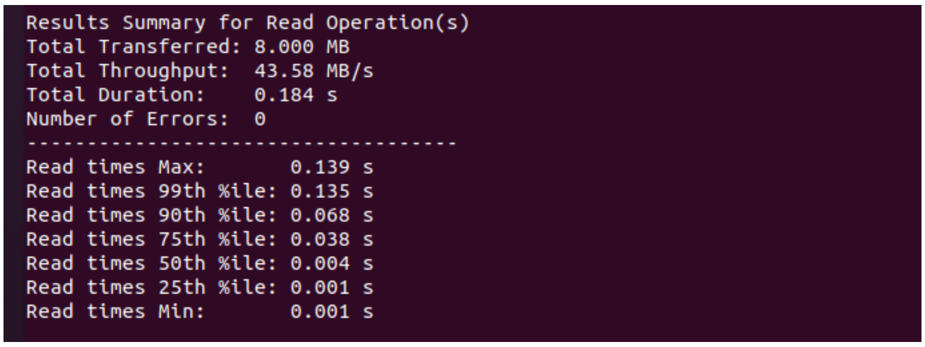
```
Results Summary for Read Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  60.41 MB/s
Total Duration:    0.132 s
Number of Errors:  0
------------------------------------
Read times Max:       0.039 s
Read times 99th %ile: 0.034 s
Read times 90th %ile: 0.011 s
Read times 75th %ile: 0.005 s
Read times 50th %ile: 0.002 s
Read times 25th %ile: 0.001 s
Read times Min:       0.000 s
```

图 3-11 numClients=8(b)



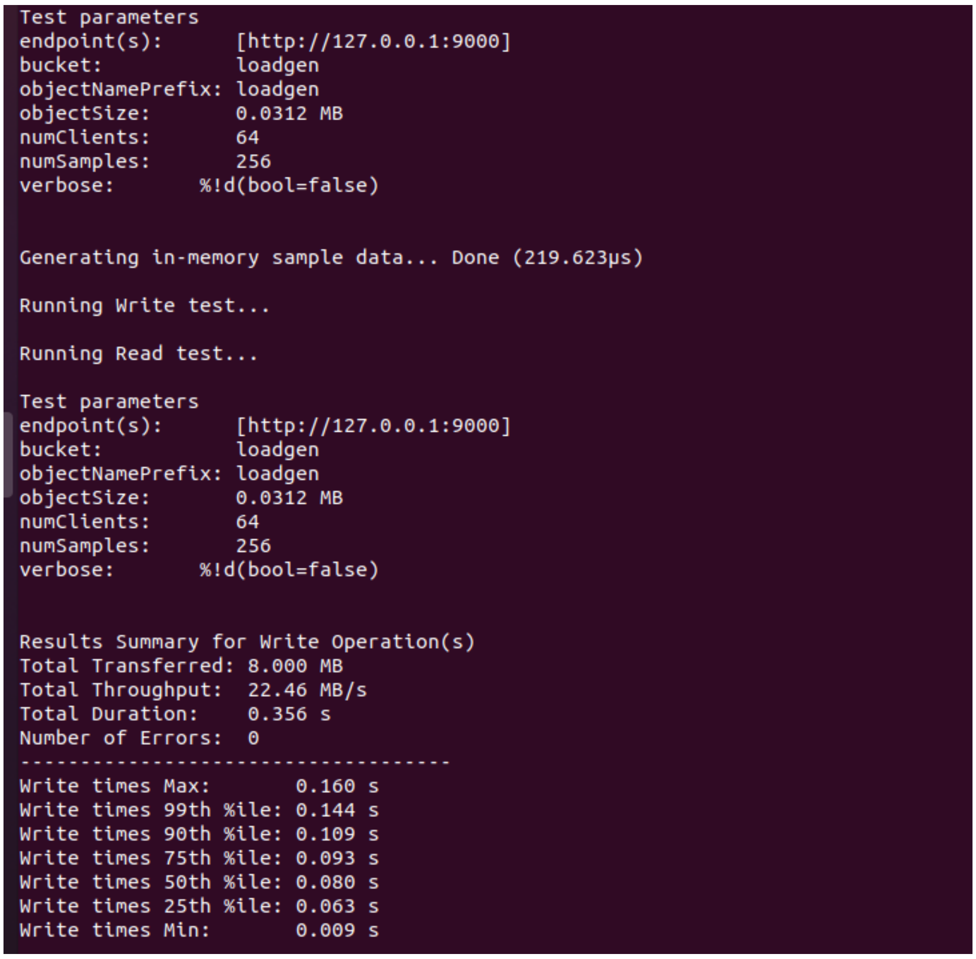
```
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.0312 MB
numClients:       32
numSamples:       256
verbose:       %!d(bool=false)


Generating in-memory sample data... Done (314.025µs)

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.0312 MB
numClients:       32
numSamples:       256
verbose:       %!d(bool=false)


Results Summary for Write Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  25.65 MB/s
Total Duration:    0.312 s
Number of Errors:  0
------------------------------------
Write times Max:       0.124 s
Write times 99th %ile: 0.090 s
Write times 90th %ile: 0.066 s
Write times 75th %ile: 0.048 s
Write times 50th %ile: 0.035 s
Write times 25th %ile: 0.023 s
Write times Min:       0.002 s
```

图 3-12 numClients=32(a)



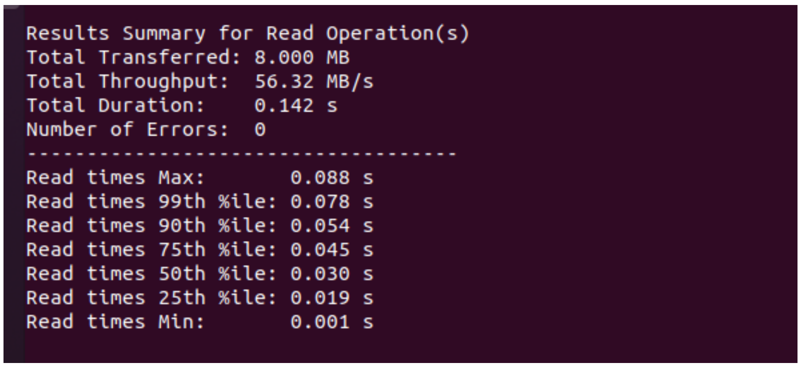
```
Results Summary for Read Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  43.58 MB/s
Total Duration:    0.184 s
Number of Errors:  0
------------------------------------
Read times Max:       0.139 s
Read times 99th %ile: 0.135 s
Read times 90th %ile: 0.068 s
Read times 75th %ile: 0.038 s
Read times 50th %ile: 0.004 s
Read times 25th %ile: 0.001 s
Read times Min:       0.001 s
```

图 3-13 numClients=32(b)



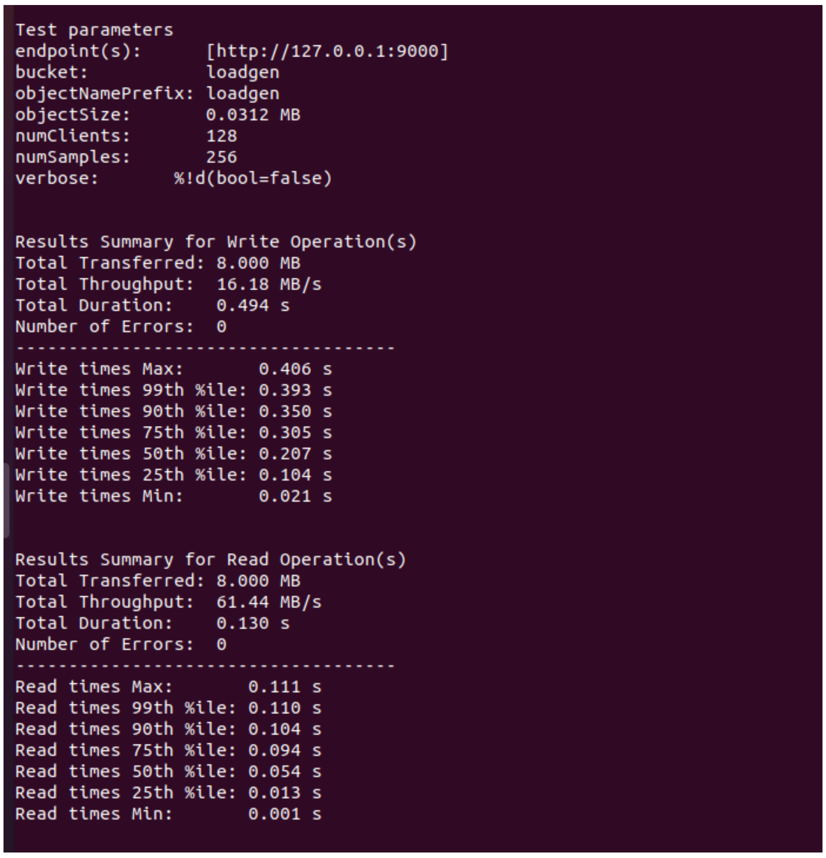
```
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.0312 MB
numClients:       64
numSamples:       256
verbose:       %!d(bool=false)


Generating in-memory sample data... Done (219.623µs)

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.0312 MB
numClients:       64
numSamples:       256
verbose:       %!d(bool=false)


Results Summary for Write Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  22.46 MB/s
Total Duration:    0.356 s
Number of Errors:  0
------------------------------------
Write times Max:       0.160 s
Write times 99th %ile: 0.144 s
Write times 90th %ile: 0.109 s
Write times 75th %ile: 0.093 s
Write times 50th %ile: 0.080 s
Write times 25th %ile: 0.063 s
Write times Min:       0.009 s
```

图 3-14 numClients=64(a)



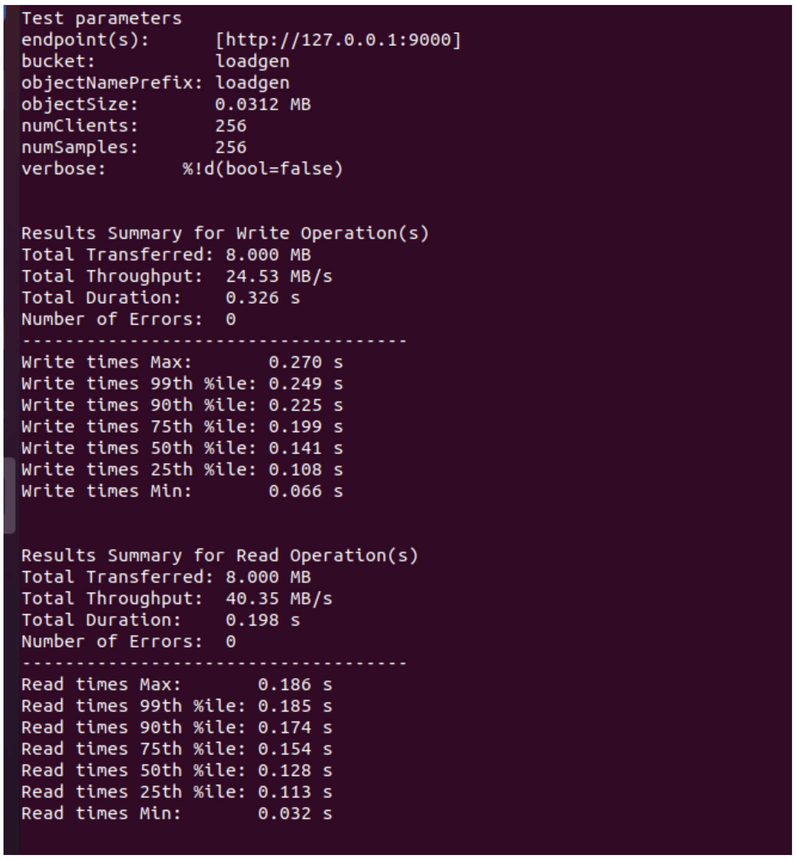
```
Results Summary for Read Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  56.32 MB/s
Total Duration:    0.142 s
Number of Errors:  0
------------------------------------
Read times Max:       0.088 s
Read times 99th %ile: 0.078 s
Read times 90th %ile: 0.054 s
Read times 75th %ile: 0.045 s
Read times 50th %ile: 0.030 s
Read times 25th %ile: 0.019 s
Read times Min:       0.001 s
```

图 3-15 numClients=64(b)



```
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.0312 MB
numClients:       128
numSamples:       256
verbose:       %!d(bool=false)


Results Summary for Write Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  16.18 MB/s
Total Duration:    0.494 s
Number of Errors:  0
------------------------------------
Write times Max:       0.406 s
Write times 99th %ile: 0.393 s
Write times 90th %ile: 0.350 s
Write times 75th %ile: 0.305 s
Write times 50th %ile: 0.207 s
Write times 25th %ile: 0.104 s
Write times Min:       0.021 s


Results Summary for Read Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  61.44 MB/s
Total Duration:    0.130 s
Number of Errors:  0
------------------------------------
Read times Max:       0.111 s
Read times 99th %ile: 0.110 s
Read times 90th %ile: 0.104 s
Read times 75th %ile: 0.094 s
Read times 50th %ile: 0.054 s
Read times 25th %ile: 0.013 s
Read times Min:       0.001 s
```

图 3-16 numClients=128



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


Results Summary for Write Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  24.53 MB/s
Total Duration:    0.326 s
Number of Errors:  0
------------------------------------
Write times Max:       0.270 s
Write times 99th %ile: 0.249 s
Write times 90th %ile: 0.225 s
Write times 75th %ile: 0.199 s
Write times 50th %ile: 0.141 s
Write times 25th %ile: 0.108 s
Write times Min:       0.066 s


Results Summary for Read Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  40.35 MB/s
Total Duration:    0.198 s
Number of Errors:  0
------------------------------------
Read times Max:       0.186 s
Read times 99th %ile: 0.185 s
Read times 90th %ile: 0.174 s
Read times 75th %ile: 0.154 s
Read times 50th %ile: 0.128 s
Read times 25th %ile: 0.113 s
Read times Min:       0.032 s
```

图 3-17 numClients=256

通过实验对比发现，在 256 以内，客户端数越大，吞吐量、总延迟的波动情况不是特别明显，但是，随着客户端的增加，长尾效应会更加严重。

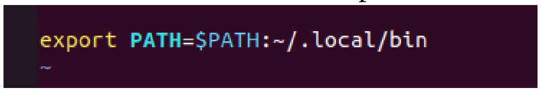
4 实验三：尾延迟测试

4.1 实验环境

Python 版本：3.8.10

Jupyter notebook 版本：6.4.6

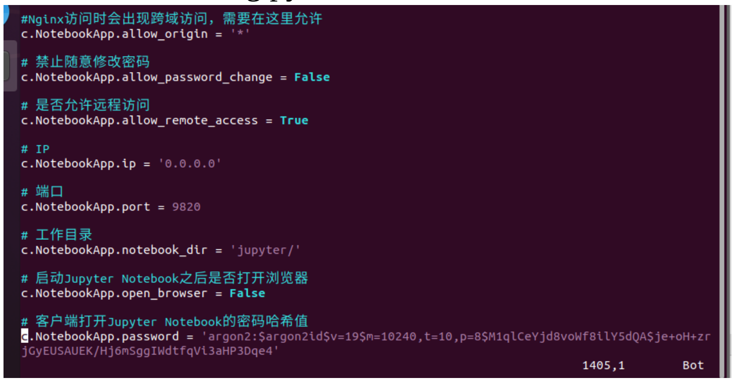
首先在命令行键入 pip3 install jupyter 下载 python3.8 对应版本的 jupyter，然后把 jupyter 的安装路径添加到系统环境变量/etc/profile 中。



```
export PATH=$PATH:~/.local/bin
~
```

图 4-1 jupyter 路径

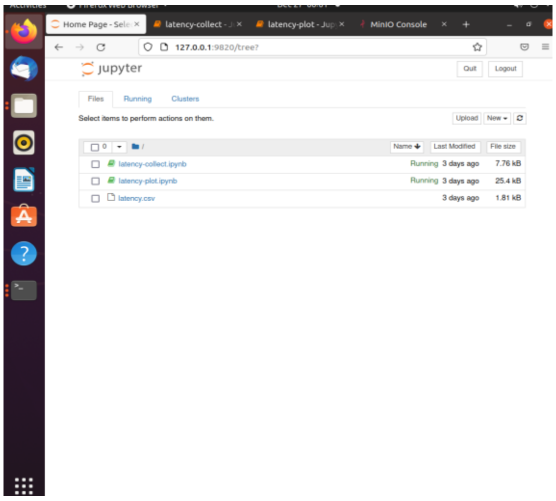
在命令行键入 pip3 install ipython 安装 ipython 模块，终端输入 ipython 进入 ipython 界面，为 jupyter 配置密码，输入 from notebook.auth import passwd，然后输入密码，将生成的 SHA1 密钥复制下来。然后，在终端输入命令：jupyter-notebook –generate-config --allow-root 生成配置文件，默认地址为./.jupyter/jupyter_notebook_config.py，使用 vim 编辑配置文件。



```
#Nginx访问时会出现跨域访问，需要在这里允许
c.NotebookApp.allow_origin = '*'

# 禁止随意修改密码
c.NotebookApp.allow_password_change = False

# 是否允许远程访问
c.NotebookApp.allow_remote_access = True

# IP
c.NotebookApp.ip = '0.0.0.0'

# 端口
c.NotebookApp.port = 9820

# 工作目录
c.NotebookApp.notebook_dir = 'jupyter/'

# 启动Jupyter Notebook之后是否打开浏览器
c.NotebookApp.open_browser = False

# 客户端打开Jupyter Notebook的密码哈希值
c.NotebookApp.password = 'argon2:$argon2id$v=19$m=10240,t=10,p=8$M1qlCeYjd8voWf8ilY5dQA$je+oH+zr
jGyEUSAUEK/Hj6mSggIWdtfqVi3aHP3Dqe4'
                                                                    1405,1          Bot
```

图 4-2 jupyter notebook 配置文件

在终端输入命令：jupyter-notebook –allow-root 启动 jupyter notebook,然后在浏览器中打开 127.0.0.1：9820 进入 jupyter notebook 首页。

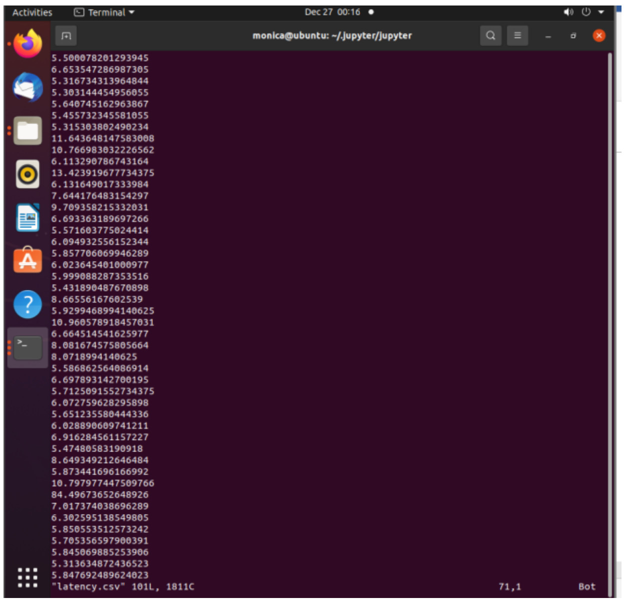


```
127.0.0.1:9820/tree?

Select items to perform actions on them.

latency-collect.ipynb    Running  3 days ago   7.76 kB
latency-plot.ipynb       Running  3 days ago   25.4 kB
latency.csv                       3 days ago   1.81 kB
```

图 4-3 jupyter notebook 首页

4.2 延迟收集

将 obs-tutorial 文件下的 latency-collect.py 上传到 jupyter notebook 上，依次运行并获得 100 个 4Kb 对象的写延迟。



```
monica@ubuntu: ~/.jupyter/jupyter
5.500078201293945
6.653547286987305
5.316734313964844
5.303144454956055
5.640745162963867
5.455732345581055
5.315303802490234
11.643648147583008
10.766983032226562
6.113290786743164
13.423919677734375
6.131649017333984
7.644176483154297
9.709358215332031
6.693363189697266
5.571603775024414
6.094932556152344
5.857706069946289
6.023645401000977
5.999088287353516
5.431890487670898
8.66556167602539
5.9299468994140625
10.960578918457031
6.664514541625977
8.081674575805664
8.0718994140625
5.586862564086914
6.697893142700195
5.7125091552734375
6.072759628295898
5.651235580444336
6.028890609741211
6.916284561157227
5.47480583190918
8.649349212646484
5.873441696166992
10.797977447509766
84.49673652648926
7.017374038696289
6.302595138549805
5.850553512573242
5.705356597900391
5.845069885253906
5.313634872436523
5.847692489624023
"latency.csv" 101L, 1811C                              71,1          Bot
```

图 4-4 收集到的 latency.csv

4.3　尾延迟展示

将 obs-tutorial 文件下的 latency-plot.py 上传到 jupyter notebook 上，依次运行得到延迟分布情况。

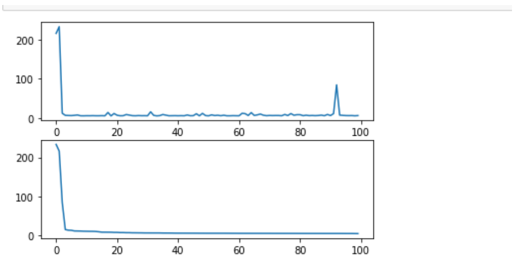


图 4-5 延迟分布情况（上图为无序状态，下图为有序状态）

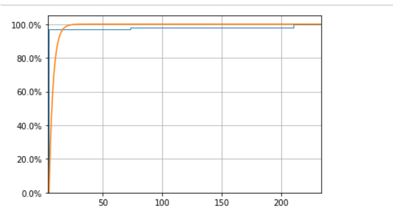


图 4-6 百分位延迟