



《数据中心技术》课程

实 验 报 告

姓 名 万 瑞 萍

学 号 M202173811

班 号 硕 2109 班

日 期 2022.1.5

指导老师 施展 童薇

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	1
五、实验过程	2
六、实验总结	8
参考文献	9

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

Minio: Minio 是个基于 Golang 编写的开源对象存储套件，基于 Apache License v2.0 开源协议，虽然轻量，却拥有着不错的性能。它兼容亚马逊 S3 云存储服务接口。可以很简单的和其他应用结合使用，例如 NodeJS、Redis、MySQL 等。

S3-benchmark: 该工具提供了针对兼容 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。

boto3: Boto 是 AWS 的基于 python 的 SDK（当然还支持其他语言的 SDK，例如 Ruby, Java 等），Boto 允许开发人员编写软件时使用亚马逊等服务像 S3 和 EC2 等，Boto 提供了简单，面向对象的 API，也提供了低等级的服务接入。Boto 有两个版本，其中旧的版本 boto2 已经不推荐使用了，boto3 中新增了许多功能。

三、实验环境

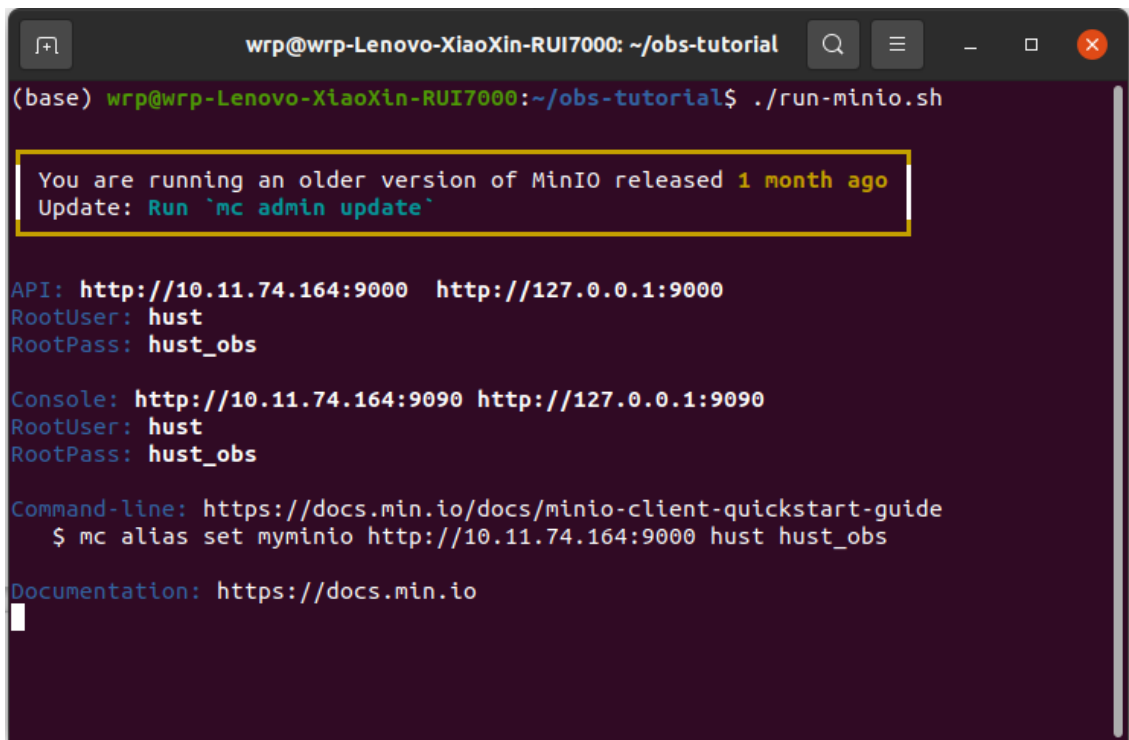
硬件环境	系统类型	x64-based PC
	处理器	Intel(R) Core(TM) i5-7300HQ CPU @ 2.50GHz 2.50 GHz
软件环境	操作系统	Ubuntu 20.04.3 LTS
	其他	go1.13.8 linux/amd64

四、实验内容

- 1.部署 Minio 服务器和客户端，验证 Minio 对象存储的功能。
- 2.安装 s3bench，学习配置 s3bench 的方法。
- 3.安装 boto3
- 4.使用 boto3 进行性能测试。

五、实验过程

1. Minio server 的安装和运行



```
wrp@wrp-Lenovo-XiaoXin-RUI7000: ~/obs-tutorial
(base) wrp@wrp-Lenovo-XiaoXin-RUI7000:~/obs-tutorial$ ./run-minio.sh

You are running an older version of MinIO released 1 month ago
Update: Run `mc admin update`

API: http://10.11.74.164:9000 http://127.0.0.1:9000
RootUser: hust
RootPass: hust_obs

Console: http://10.11.74.164:9090 http://127.0.0.1:9090
RootUser: hust
RootPass: hust_obs

Command-line: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc alias set myminio http://10.11.74.164:9000 hust hust_obs

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

图 5-1 Minio server 的启动 1

打开 <http://127.0.0.1:9090>:

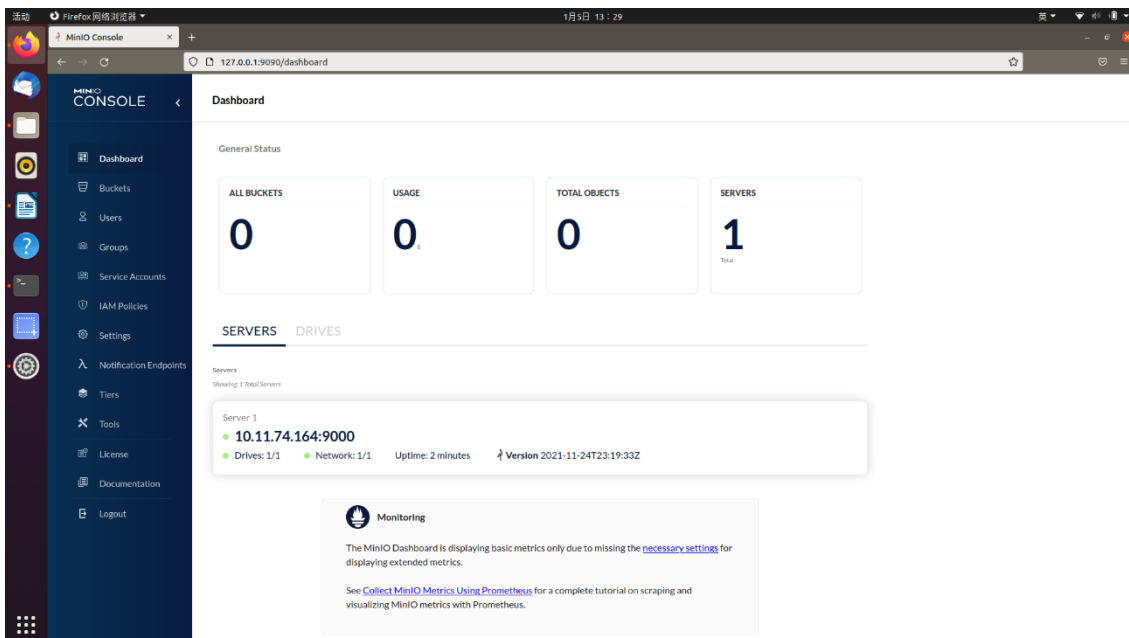


图 5-2 Minio server 的启动 2

2. 运行 s3-bench 对 Minio 的 Server 进行测试

```
(base) wrp@wrp-Lenovo-XiaoXin-RUI7000:~/obs-tutorial$ ./run-s3bench.sh
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 (2.866096ms)

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: 0.000 MB
Total Throughput:  0.00 MB/s
Total Duration:    0.150 s
Number of Errors:  256

Results Summary for Read Operation(s)
Total Transferred: 0.000 MB
Total Throughput:  0.00 MB/s
Total Duration:    0.035 s
Number of Errors:  256

Cleaning up 256 objects...
Deleting a batch of 256 objects in range [0, 255]... Failed (NoSuchBucket: The specified bucket does not exist
status code: 404, request id: 16C748F317654EFA, host id: )
Successfully deleted 0/256 objects in 2.920705ms
(base) wrp@wrp-Lenovo-XiaoXin-RUI7000:~/obs-tutorial$
```

图 5-3 s3-bench 测评

3. 学习并理解 s3-bench 测试程序的构成

```
#!/bin/sh

# Locate s3bench
s3bench=~/go/bin/s3bench

if [ -n "$GOPATH" ]; then
    s3bench=$GOPATH/bin/s3bench
fi

# -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.

$s3bench \
    -accessKey=hust \
    -accessSecret=hust_obs \
    -bucket=loadgen \
    -endpoint=http://127.0.0.1:9000 \
    -numClients=8 \
    -numSamples=256 \
    -objectNamePrefix=loadgen \
    -objectSize=$(( 1024*32 ))

# build your own test script with designated '-numClients', '-numSamples' and '-objectSize'
# 1. Use loop structure to generate test batch (E.g.: to re-evaluate multiple s3 servers under the same configuration, or to gather data from a range of parameters);
# 2. Use redirection (the '>' operator) for storing program output to text files;
# 3. Observe and analyse the underlying relation between configuration parameters and performance metrics.
```

图 5-4 s3-bench.sh

可以看出，s3-bench 需要运行在 go 环境下，首先需要输入用户名和密码，然后设置用于存放所有测试对象的 bucket。接下来设置了模拟 8 个客户端同时运行的情况，共有 256 个测试对象，测试对象的名称前缀均为 loadgen，每个测试对象的大小为 1024 字节。

4. 安装 boto3

Boto3 在 github 上下载安装或直接使用 pip 安装均可。如图所示 boto3 已经安装好了。

```
(base) wrp@wrp-Lenovo-XiaoXin-RUI7000:~/obs-tutorial$ python
Python 3.8.8 (default, Apr 13 2021, 19:58:26)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import boto3
>>> 
```

图 5-5 boto3 安装情况

5. 使用 latency-collect.py 测试

实验环境初始化

```
In [1]: import os
import time
from concurrent.futures import ThreadPoolExecutor, as_completed
from boto3 import Session
import botocore
from tqdm import tqdm
import throttle

# 准备密钥
aws_access_key_id = 'hust'
#hust
aws_secret_access_key = 'hust-obs'
#hust_obs

# 本地S3服务地址
local_s3 = 'http://10.11.74.164:9000'
# 'http://192.168.33.1:9000'

# 建立会话
session = Session(aws_access_key_id=aws_access_key_id, aws_secret_access_

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

图 5-6 实验环境初始化

查看所有bucket

```
In [29]: for bucket in s3.buckets.all():
print('bucket name:%s' % bucket.name)
```

图 5-7 查看所有 bucket

新建一个实验用 bucket (注意: "bucket name" 中不能有下列线)

```
In [30]: bucket_name = 'test100objs'
if s3.Bucket(bucket_name) not in s3.buckets.all():
    s3.create_bucket(Bucket=bucket_name)
```

图 5-8 新建一个实验用 bucket

查看此 bucket 下的所有 object (若之前实验没有正常结束, 则不为空)

```
In [31]: bucket = s3.Bucket(bucket_name)
for obj in bucket.objects.all():
    print('obj name:%s' % obj.key)
```

图 5-9 查看此 bucket 下所有 object

准备负载, 可以按照几种不同请求到达率 (Inter-Arrival Time, IAT) 设置。

```
In [32]: # 初始化本地数据文件
local_file = "_test 4K.bin"
test_bytes = [0xFF for i in range(1024*4)] # 填充至所需大小

with open(local_file, "wb") as lf:
    lf.write(bytearray(test_bytes))

# 发起请求和计算系统停留时间
def request_timing(s3res, i): # 使用独立 session.resource 以保证线程安全
    obj_name = "testObj%08d"%(i,) # 所建对象名
    # temp_file = '.tempfile'
    service_time = 0 # 系统滞留时间
    start = time.time()
    s3res.Object(bucket_name, obj_name).upload_file(local_file) # 将本地文件上传为对象
    # 或
    # bucket.put_object(Key=obj_name, Body=open(local_file, 'rb'))
    # 下载obj
    # s3res.Object(bucket_name, obj_name).download_file(temp_file)
    end = time.time()
    system_time = end - start
    return system_time * 1000 # 换算为毫秒

# 按照请求到达率限制来执行和跟踪请求
def arrival_rate_max(s3res, i): # 不进行限速
    return request_timing(s3res, i)

@throttle.wrap(0.1, 2) # 100ms 内不超过 2 个请求, 下同.....
def arrival_rate_2(s3res, i):
    return request_timing(s3res, i)

@throttle.wrap(0.1, 4)
def arrival_rate_4(s3res, i):
    return request_timing(s3res, i)

@throttle.wrap(0.1, 8)
def arrival_rate_8(s3res, i):
    return request_timing(s3res, i)
```

图 5-10 准备负载

按照预设IAT发起请求

```
In [33]: latency = []
failed_requests = []

with tqdm(desc="Accessing S3", total=100) as pbar:
    with ThreadPoolExecutor(max_workers=1) as executor:
        futures = [
            executor.submit(
                arrival_rate_max,
                session.resource('s3', endpoint_url=local_s3), i) for i in range(100) # 为保证线程安全，应给每个任务申请一
        ]
        for future in as_completed(futures):
            if future.exception():
                failed_requests.append(futures[future])
            else:
                latency.append(future.result()) # 正确完成的请求，采集延迟
        pbar.update(1)

Accessing S3: 100%|██████████| 100/100 [00:09<00:00, 10.64it/s]
```

图 5-11 按照预设 IAT 发起请求

清理实验环境

```
In [34]: try:
        # 删除bucket下所有object
        bucket.objects.filter().delete()

        # 删除bucket下某个object
        # bucket.objects.filter(Prefix=obj_name).delete()

        bucket.delete()
    except botocore.exceptions.ClientError as e:
        print('error in bucket removal')
```

图 5-12 清理实验环境

删除本地测试文件

```
In [35]: os.remove(local_file)
```

图 5-13 删除本地测试文件

记录延迟到CSV文件

```
In [36]: with open("latency.csv", "w+") as tracefile:
        tracefile.write("latency\n")
        tracefile.writelines([str(l) + '\n' for l in latency])
```

图 5-14 记录延迟到 csv 文件

6. 使用 latency-plot.py 观察延迟情况

请求延迟分布情况

```
In [5]: plt.subplot(211)
plt.plot(latency)
plt.subplot(212)
plt.plot(sorted(latency, reverse=True))
plt.show()
```

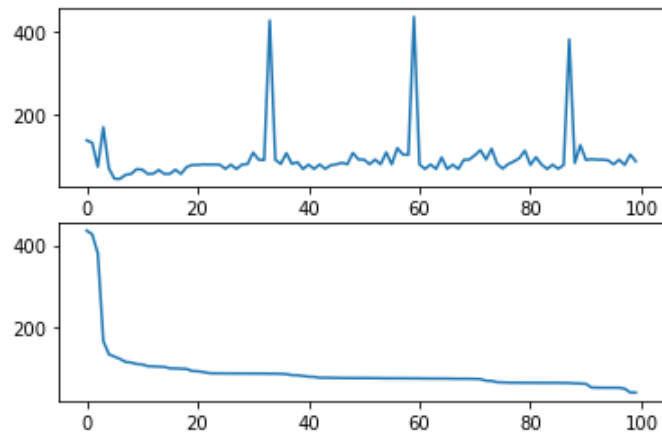


图 5-15 请求延迟分布情况

用排队论模型来拟合实测数据

```
In [6]: # 百分比换算
def to_percent(y, position):
    return str(100 * round(y, 2)) + "%"

# 设置纵轴为百分比
formatter = FuncFormatter(to_percent)
ax = plt.gca()
# ax.xaxis.set_major_locator(MultipleLocator(5))
ax.yaxis.set_major_formatter(formatter)
# 避免横轴数据起始位置与纵轴重合, 调整合适座标范围
x_min = max(min(latency) * 0.8, min(latency) - 5)
x_max = max(latency)
plt.xlim(x_min, x_max)
# 绘制实际百分位延迟
plt.hist(latency, cumulative=True, histtype='step', weights=[1./ len(latency)] * len(latency))

# 排队论模型
#  $F(t)=1-e^{(-1*a*t)}$ 
alpha = 0.3
X_qt = np.arange(min(latency), max(latency), 1.)
Y_qt = 1 - np.exp(alpha * (min(latency) - X_qt))
# 绘制排队论模型拟合
plt.plot(X_qt, Y_qt)

plt.grid()
plt.show()
```

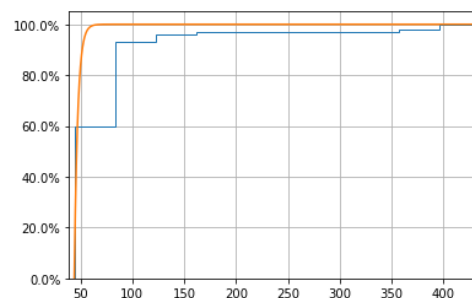


图 5-16 用排队论模型来拟合实测数据

六、实验总结

对象存储的概念我第一接触，本次实验目前为之前所有关于对象存储的知识都来源于老师课堂上的讲授，并没有更多地额外接触对象存储的理论知识，本次实验主要围绕着对象存储服务器的部署、配置以及测试来展开。

实验过程相对简单，没有特别复杂的内容，老师给的实验指导详细而完整，照着老师的指导一步步来做，基本上不会遇到太大困难，由于各个机器环境不同而遇到的困难，都可以通过上网找到解决方案。

经过本次实验，我接触到一个较为方便的工具 minio，Minio 是个基于 Golang 编写的开源对象存储套件，基于 Apache License v2.0 开源协议，虽然轻量，却拥有着不错的性能。它兼容亚马逊 S3 云存储服务接口。可以很简单的和其他应用结合使用，例如 NodeJS、Redis、MySQL 等。minio 简单易上手的特性在实验过程中体现得淋漓尽致

参考文献

- [1] https://gitee.com/shi_zhan/obs-tutorial
- [2] <https://minio.io/downloads.html>
- [3] <https://github.com/boto/boto3>
- [4] <https://github.com/igneous-systems/s3bench>