对象存储实验

姓名：洪涛

学号：M202173712

1 实验目的

- 熟悉对象存储技术，代表性系统及其特性
- 实践对象存储系统，部署实验环境，进行初步测试
- 基于对象存储系统，架设实际应用，示范主要功能

2 实验背景

随着互联网的不断扩张和云计算技术的进一步推广，海量的数据在个人、企 业、研究机构等源源不断地产生，如何有效、快速、可靠地存取这些日益增长的 海量数据成了关键的问题。

传统的存储解决方案面临越来越多的困难，比如数据量的指数级增长对不断扩容的存储空间提出要求、实时分析海量的数据对存储计算能力提出要求等。因此，需要能处理海量数据、高性能、易扩展、可伸缩、高可用的新型存储方案。对象存储系统（Object-Based Storage System）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。

对象存储架构由对象、对象存储设备、元数据服务器、对象存储系统的客户端四部分组成，其核心是将数据通路（数据读或写）和控制通路（元数据）分离， 并且基于对象存储设备构建存储系统，每个对象存储设备具有一定的智能，能够自动管理其上的数据分布。

3 实验环境

实验环境	信息
硬件环境	处理器：Intel® Core™ i5-7300HQ CPU @ 2.50GHz × 4
	内存：4GB
软件环境	OS：Ubuntu 21.04
	Virtualization：VMware
	Python：Python 3.9.5
	GO：go version go1.11.5 linux/amd64
	服务器端：mock-s3
	客户端：s3cmd
	测试工具：s3bench

4 实验内容

本次实验的主要内容有搭建对象存储服务器；搭建对象存储客户端；使用对象存储测评工具进行评测。

4.1 对象存储技术实践

1. 搭建实验基本环境，如Python、Go等
2. 搭建对象存储服务器端，mock-s3
3. 搭建对象存储客户端，s3cmd

4.2 对象存储性能分析

1. 搭建性能评测工具，s3bench
2. 测试并观察记录对象尺寸对吞吐率和延迟的影响。
3. 测试并观察记录样本数对吞吐率和延迟的影响。
4. 测试并观察记录并发数对吞吐率和延迟的影响。

4.3 尾延迟处理

尝试应对对冲请求、关联请求

5 实验过程

1. 对象存储技术实践

安装并配置Java、Python、Go

```
taosupr@ubuntu:~/Programs$ java --version
openjdk 17.0.1 2021-10-19
OpenJDK Runtime Environment (build 17.0.1+12-39)
OpenJDK 64-Bit Server VM (build 17.0.1+12-39, mixed mode, sharing)
taosupr@ubuntu:~/Programs$ python3 --version
Python 3.9.5
taosupr@ubuntu:~/Programs$ go version
go version go1.11.5 linux/amd64
taosupr@ubuntu:~/Programs$
```

安装mock-s3

1. 从https://github.com/ShiZhan/mock-s3下载mock-s3-master
2. 运行mock-s3

```
cd mock_s3
python3 main.py --hostname 0.0.0.0 --port 9000 --root ./root
```

```
taosupr@ubuntu:~/Programs/mock-s3-master/mock_s3$ python3 main.py --hostname 0.0.0.0 --port 9000 --
root ./root
Starting server, use <Ctrl-C> to stop
127.0.0.1 - - [03/Jan/2022 06:23:09] "GET / HTTP/1.1" 200 -
```

安装s3cmd

1. 安装

```
pip install s3cmd
```

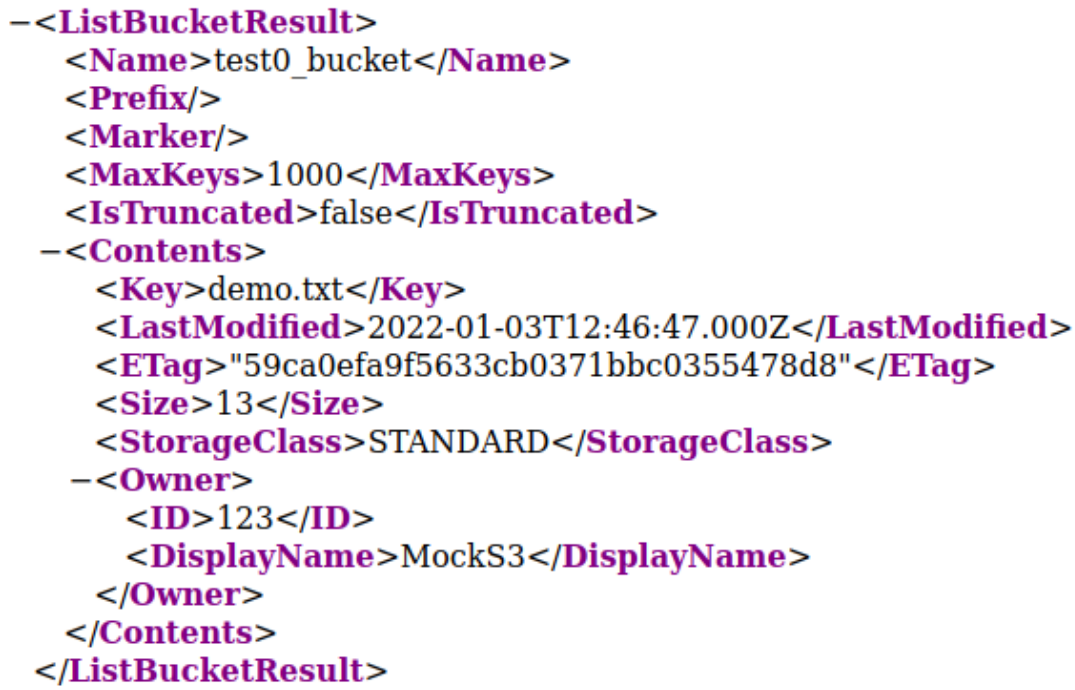
2. 配置

```
s3cmd --configure
```

```
HTTP Proxy server name:

New settings:
  Access Key: hust
  Secret Key: hust_obs
  Default Region: US
  S3 Endpoint: 127.0.0.1:9000
  DNS-style bucket+hostname:port template for accessing a bucket: 9000
  Encryption password: hust_obs
  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...
WARNING: Could not refresh role
ERROR: Test failed: [Errno 111] Connection refused

Retry configuration? [Y/n] n
```

测试服务端与客户端

```
s3cmd mb test0bucket #创建新bucket
s3cmd put ~/demo.txt s3://test0_bucket #上传测试
```



2. 对象存储性能分析

安装s3bench

1. 安装

```
go get -u github.com/igneous-systems/s3bench
```

2. 命令行范例

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

测试结果

```
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:  6.98 MB/s
Total Duration:    1.147 s
Number of Errors:  0
-----------------------------------
Write times Max:       1.029 s
Write times 99th %ile: 0.151 s
Write times 90th %ile: 0.042 s
Write times 75th %ile: 0.018 s
Write times 50th %ile: 0.012 s
Write times 25th %ile: 0.011 s
Write times Min:       0.003 s


Results Summary for Read Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  7.54 MB/s
Total Duration:    1.060 s
Number of Errors:  0
-----------------------------------
Read times Max:       1.033 s
Read times 99th %ile: 0.108 s
Read times 90th %ile: 0.042 s
Read times 75th %ile: 0.018 s
Read times 50th %ile: 0.011 s
Read times 25th %ile: 0.010 s
Read times Min:       0.007 s


Cleaning up 256 objects...
Deleting a batch of 256 objects in range {0, 255}... Succeeded
Successfully deleted 256/256 objects in 60.027437ms
```

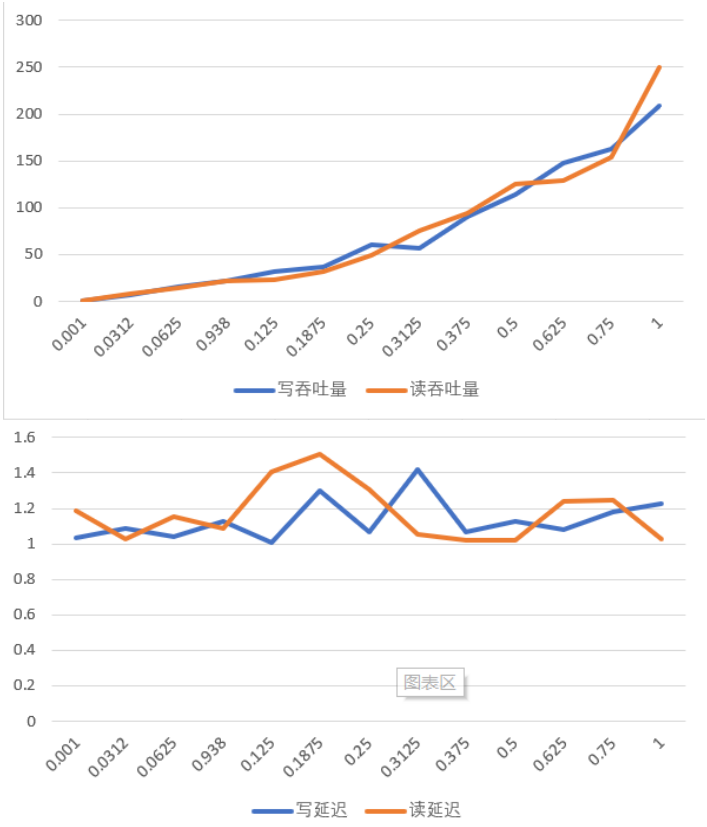
测试并观察记录对象尺寸对吞吐率和延迟的影响

numSample设为256，numClients设置为8，

改变ObjectSize的值分别进行实验，测试脚本与实验结果如下：

```
#!/bin/sh

# Locate s3bench

s3bench=~/goProject/bin/s3bench
testresult=~/goProject/bin/testresult

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

# mk testresult dir if it doesn't exist
if [ ! -d $testresult ]; then
    mkdir  $testresult
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.

# set objectSize as (element in testArr * 1024)
testArr=(1 32 64 96 128 192 256 320 384 512 640 768 1024)

for i in ${testArr[*]}
do
        $s3bench \
        -accessKey=hust \
        -accessSecret=hust_obs \
        -bucket=loadgen \
        -endpoint=http://127.0.0.1:9000 \^S
        -numClients=8 \
        -numSamples=256 \
        -objectNamePrefix=loadgen \
        -objectSize=$(( 1024*$i )) >> $testresult/result1
done

# 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.
```

ObjectSize(MB)	Total Throughput of Write(MB/s)	Total Duration of Write(s)	Total Throughput of Read(MB/s)	Total Duration of Read(s)
0.0010	0.24	1.033	0.21	1.184
0.0312	7.34	1.090	7.80	1.026
0.0625	15.38	1.040	13.90	1.151
0.9380	21.27	1.128	22.07	1.088
0.1250	31.74	1.008	22.76	1.406
0.1875	36.91	1.300	31.94	1.503
0.2500	60.00	1.067	48.91	1.309
0.3125	56.40	1.419	75.98	1.053
0.3750	90.12	1.065	94.08	1.020
0.5000	113.89	1.124	125.18	1.023
0.6250	147.85	1.082	128.72	1.243
0.7500	163.14	1.177	154.17	1.245
1.0000	208.28	1.229	249.73	1.025



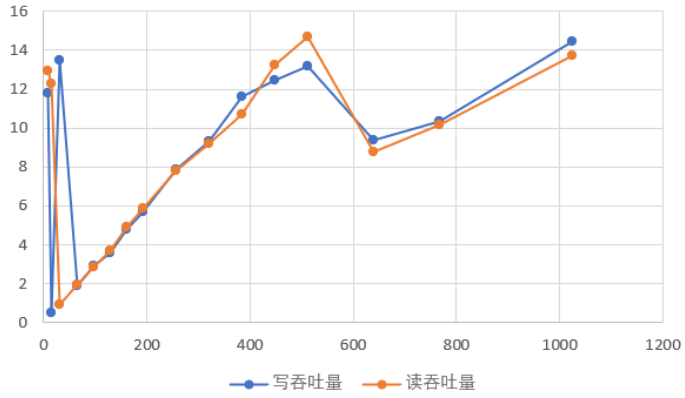
一定范围内，随着对象大小的增加，读写的吞吐量都在逐渐增加，读的吞吐量在后期增加的更快，其对于读写的延迟影响不大，呈现波动状态

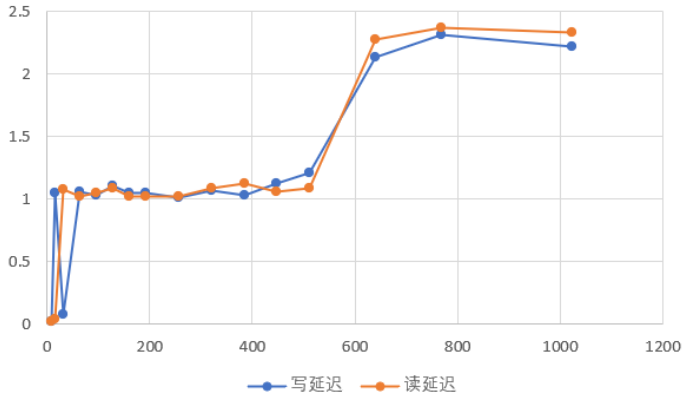
测试并观察记录样本数对吞吐率和延迟的影响

设置ObjectSize为1024*32，numClients为8

改变numSamples值，试验结果如下（测试代码与上例相似，不再赘述）：

numSamples	Total Throughput of Write(MB/s)	Total Duration of Write(s)	Total Throughput of Read(MB/s)	Total Duration of Read(s)
8	11.81	0.021	12.94	0.019
16	0.48	1.048	12.25	0.041
32	13.51	0.074	0.93	1.076
64	1.89	1.061	1.96	1.022
96	2.92	1.029	2.85	1.054
128	3.60	1.110	3.68	1.088
160	4.76	1.051	4.89	1.023
192	5.72	1.049	5.86	1.024
256	7.88	1.015	7.82	1.023
320	9.32	1.073	9.20	1.087
384	11.61	1.034	10.71	1.121
448	12.44	1.125	13.27	1.055
512	13.18	1.214	14.70	1.089
640	9.35	2.138	8.80	2.272
768	10.37	2.315	10.14	2.367
1024	14.44	2.216	13.70	2.336





样本数在前期8到64的增加过程中，会导致读写吞吐量急剧下降，后随着样本数的继续增加，读写吞吐量也呈增加趋势

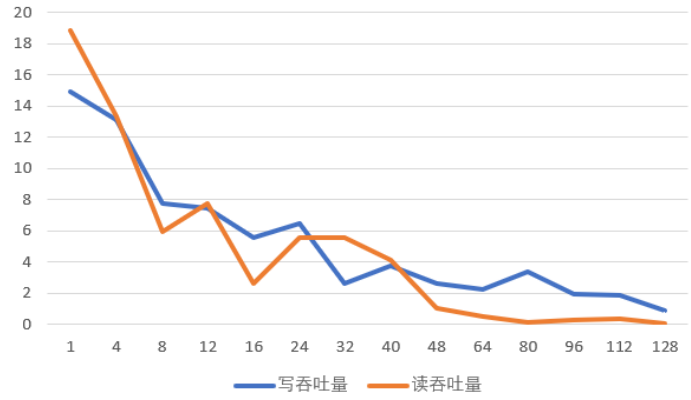
随着样本数的增加，读写延迟也随之增加，增加到一定程度后区域平稳。

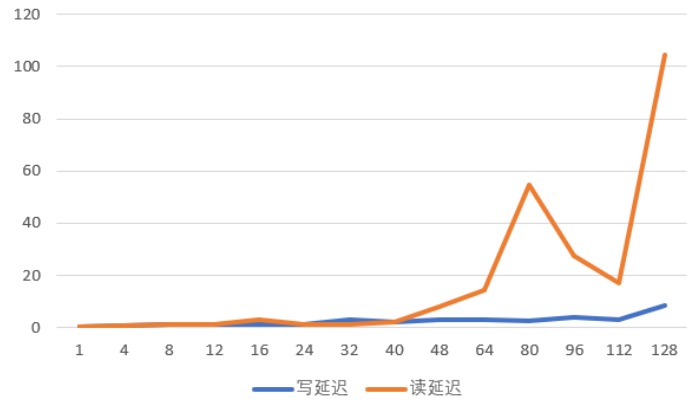
测试并观察记录并发数对吞吐率和延迟的影响

设置ObjectSize为1024*32，numSamples为256

改变numClients值，试验结果如下（测试代码与上例相似，不再赘述）：

numClients	Total Throughput of Write(MB/s)	Total Duration of Write(s)	Total Throughput of Read(MB/s)	Total Duration of Read(s)
1	14.96	0.535	18.87	0.424
4	13.12	0.610	13.36	0.599
8	7.74	1.033	5.95	1.344
12	7.46	1.072	7.73	1.035
16	5.60	1.429	2.63	3.042
24	6.45	1.239	5.56	1.440
32	2.62	3.049	5.53	1.447
40	3.75	2.134	4.13	1.935
48	2.62	3.058	1.02	7.840
64	2.22	3.253	0.50	14.523
80	3.37	2.373	0.15	54.533
96	1.94	4.098	0.29	27.690
112	1.88	3.250	0.35	17.280
128	0.88	8.337	0.06	104.496





随着并发客户端的增加，读写吞吐量逐渐降低，当并发数达到一定程度，吞吐量会接近于0；

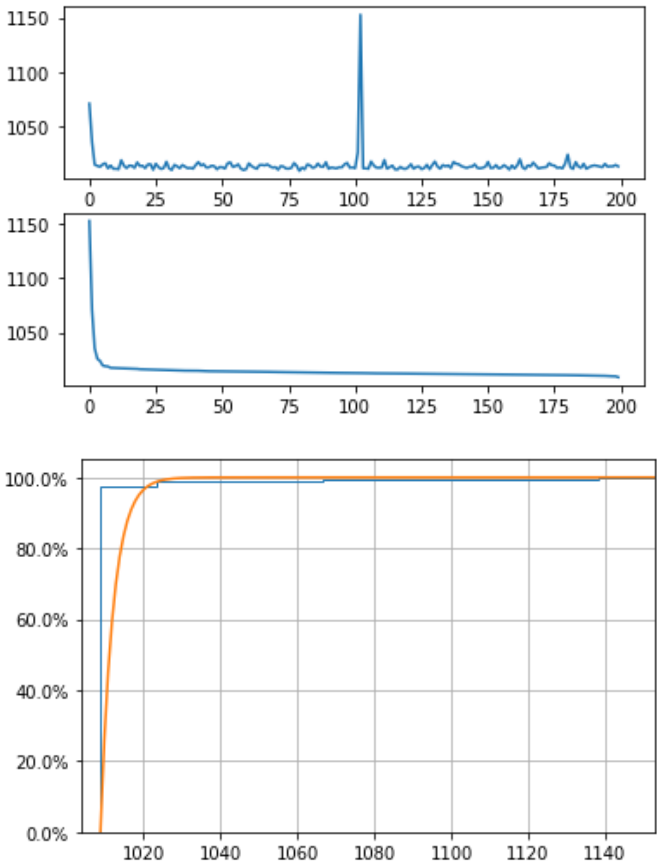
并发数在40以下时，读写延迟都比较稳定，当并发数继续增加，读延迟显著升高，在并发数超过112之后，写延迟也逐渐显著提升。

3. 尾延迟挑战

云存储系统的尾响应延迟是指系统中最慢的少量数据访问请求对应的响应延迟。

- 尾部延迟总是用百分数来表示；
- 长尾延迟指的是与平均延迟时间相比延迟的更高百分位数(例如 第98、第99)。

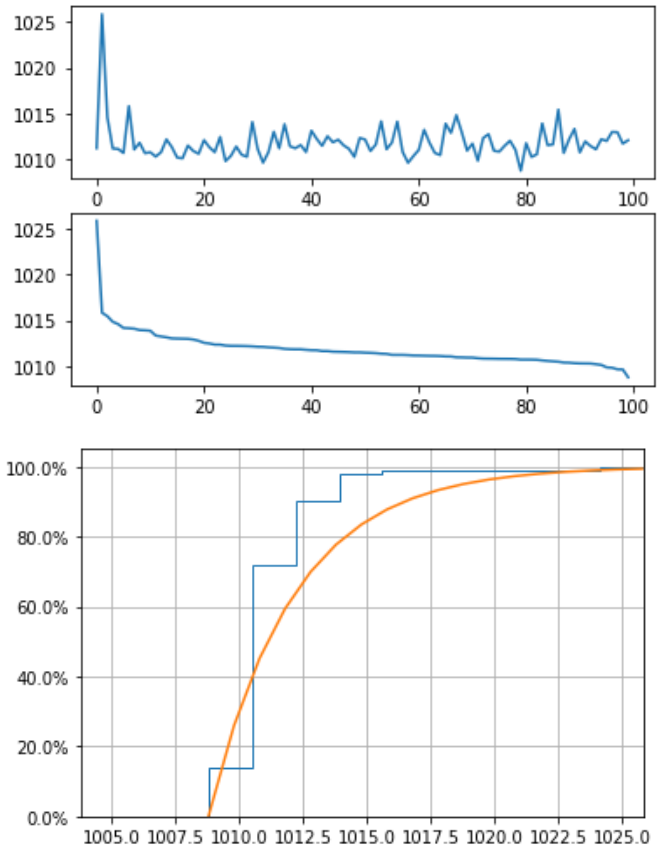
请求延迟分布情况与排队论模型拟合实测数据如下：



对冲请求(Hedged requests)：一种用来抑制延迟抖动的简单方法就是将相同请求发送给多个副本，然后使用最先返回的那个作为结果。我们将这些请求称为“对冲请求”，因为客户端会首先发送一个请求到被认为是最合适的那个副本，随后在短暂的延迟后会再发送一个请求。一旦接收到响应结果客户端会忽略剩余未完成的请求。虽然该技术的这种简单实现版本会引入不可接受的额外负载，但是它的一些变种实现却可以在对负载略有影响的情况下得到很好的延迟降低效果。

实验设计：先后向同一副本发送两次请求，选择延时较短的作为结果

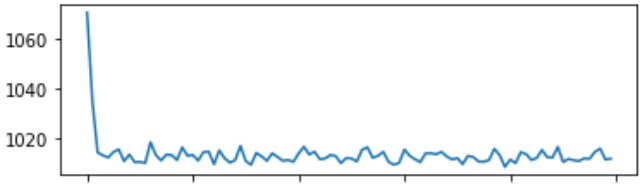
实验结果：将99th延迟降到了从1138ms降到了1023ms

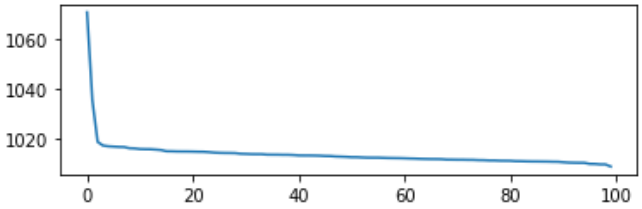


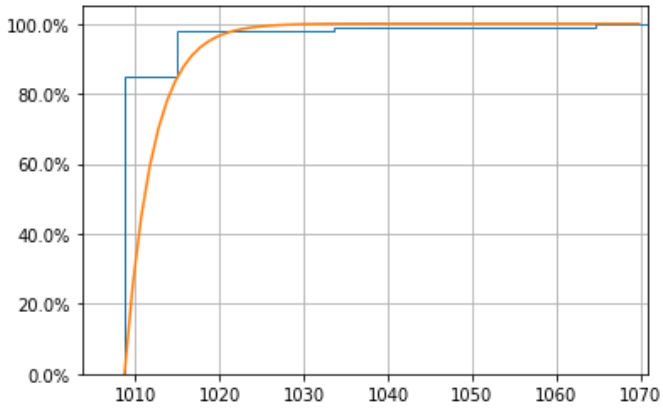
关联请求（Tied requests）：对冲请求技术存在一个脆弱窗口，在该窗口内多个服务器可能会不必要地对同一个请求进行处理。这些额外的工作量可以通过在发送对冲请求之 前等待所有延迟的第 95 个的百分位数代表的时间进行限制，但是这就将收益限 制在了一少部分请求之上。如果要在合理的资源消耗下更积极的使用对冲请求， 则需要能对请求进行更快速的取消。

实验设计：发送两个请求，99th的延迟为1137ms，对超过1138ms的请求重启发送一次

实验结果：将99th延迟从1138ms降到了1063ms







6 实验总结

本次实验从环境配置到对象存储技术实践再到对象存储性能分析、尾延迟处理，一步步逐渐深入，将实践者引入到对象存储这门技术中。前两步文档提供的非常全面、可选择性也非常的高，过程还是比较顺利的，最后的两部分难度逐渐提升，文档也少很多，更考验对对象存储系统的理解。这次试验提高了我对象存储系统的理解水平以及Linux平台的实验操作水平，总体而言收获颇丰。