



2017 级

《物联网数据存储与管理》课程

实 验 报 告

姓 名 库尔夏提·亚森

学 号 U201714621

班 号 计算机 1804 班

日 期 2021.06.28

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	1
4.1 对象存储技术实践	1
4.2 对象存储性能分析	1
五、实验过程	1
六、实验总结	8
参考文献	8

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

随着万维网的持续进化和物联网的持续发展，当今数据存储存在着巨大的挑战。一方面数据庞大，规模持续扩张；另一方面，数据的内容结构非常丰富。传统模式的存储技术效率低且管理复杂，而新产生的对象存储技术具有性能高、管理简单的特点。它具有存储“数据”和“属性”的特点，使它更可以满足未来的扩展操作。

Mock- S3，是用 Python 重写 fake- S3 实现的，沙盒环境中测试非常有用，无需实际调用 Amazon，其目标是最小化运行时依赖关系，并更像是一个开发工具来测试代码中的 S3 调用。

S3cmd 是一款免费的命令行工具和客户端，用于在 Amazon S3 和其他使用 S3 协议的云存储服务提供商中上传，检索和管理数据。

三、实验环境

表 1 实验环境说明

操作系统	Ubuntu 16.04 LTS
系统内核版本	Linux version 4.15.0
CPU	Intel®Core™ i7- 6700HQ CPU @ 2.60GHz × 2
Python 版本	2.7.12
Golang 版本	1.14.3

四、实验内容

- 1.熟悉 Git 与 linux 环境，安装 Python，Go 以及 Java 基础环境。
- 2.实践对象存储，配置服务端与客户端，并进行基础的操作尝试。
- 3.对配置好的对象存储系统进行测试，并分析性能。

4.1 对象存储技术实践

- 1.在 linux 下配置 Python，Go 等运行环境。
- 2.安装 mock- s3 作为服务器端，安装 s3cmd 作为客户端。
- 3.使用例如新建 bucket、上传删除文件等指令简单测试系统。

4.2 对象存储性能分析

- 1.安装测试用工具 s3- benchmark。
- 2.改变各种不同的参数，记录每次的测试结果。
- 3.根据测试结果，分析不同数据的影响。

五、实验过程

(编号说明实验过程及观测效果、数据、图表)

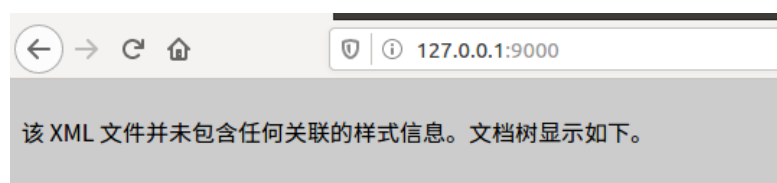
1.首先配置实验环境，包括 Python 和 Go，配置完成好的环境如下图所示。

```
zhan@ubuntu:~$ go version
go version go1.14.3 linux/amd64
zhan@ubuntu:~$ python --version
Python 2.7.12
```

图 1 Python 及 Go 所用版本

2.安装搭建 mock- s3 服务端，直接从 github 上获取项目源码，根据教程安装启动即可。

3.安装配置 s3cmd，直接输入 pip install s3cmd 运行。待安装完成后，配置 access_key,secret_key,host_base 以及 host_bucket 等信息。之后通过 s3cmd，创建 bucket 并查看。



该 XML 文件并未包含任何关联的样式信息。文档树显示如下。

```
-<ListAllMyBucketsResult>
  -<Owner>
    <ID>123</ID>
    <DisplayName>MockS3</DisplayName>
  </Owner>
  -<Buckets>
    -<Bucket>
      <Name>one</Name>
      <CreationDate>2020-05-13T15:34:18.000Z</CreationDate>
    </Bucket>
  </Buckets>
</ListAllMyBucketsResult>
```

图 2 创建的一个名为 one 的 bucket

```
zhan@ubuntu:~$ s3cmd ls
2020-05-13 15:34 s3://one
```

图 3 使用 s3cmd 命令查看所有 bucket

随后向新建的 bucket 上传一个测试文件。

```
zhan@ubuntu:~$ s3cmd put file.txt s3://one
upload: 'file.txt' -> 's3://one/file.txt' [1 of 1]
12 of 12 100% in 0s 2.30 KB/s done
```

图 4 向 bucket 上传测试文件

之后在浏览器中查看，发现文件已经上传成功。



图 5 在浏览器中查看文件

再使用命令删除这个文件。

```
zhan@ubuntu:~$ s3cmd del s3://one/file.txt
delete: 's3://one/file.txt'
```

图 6 使用 s3cmd 删除文件

至此，对于服务端与客户端的安装和基本操作测试完成。下面使用 s3- benchmark 进行性能测试和分析。

使用命令 `go get -u github.com/chinglinwen/s3- benchmark` 安装测试工具，安装完毕后，使用 `./s3- benchmark -a 123 -s 123 -u http://127.0.0.1:9000 -b one -d 3 -t 1 -z 1k` 先进行测试，看安装是否成功。

测试时服务端在不断返回数据，而测试工具也给出了最后的结果，由此可见测试成功。

```
zhan@ubuntu:~/mock-s3-master
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-923 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-924 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-925 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-926 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-927 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-928 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-929 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-930 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-931 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-932 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-933 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-934 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-935 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-936 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-937 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-938 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-939 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-940 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-941 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-942 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-943 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-944 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-945 HTTP/1.1" 204 -
127.0.0.1 - - [25/May/2020 16:34:44] "DELETE /one/Object-946 HTTP/1.1" 204 -
```

图 7 服务端在测试时返回数据

```

zhan@ubuntu:~/go/src/github.com/chinglinwen/s3-benchmark$ ./s3-benchmark -a 123
-s 123 -u http://127.0.0.1:9000 -b one -d 3 -t 1 -z 1k
Wasabi benchmark program v2.0
Parameters: url=http://127.0.0.1:9000, bucket=one, region=us-east-1, duration=3,
threads=1, loops=1, size=1k
Loop 1: PUT time 3.0 secs, objects = 1038, speed = 346KB/sec, 346.0 operations/s
ec. Slowdowns = 0
Loop 1: GET time 2.9 secs, objects = 1038, speed = 357.4KB/sec, 357.4 operations
/sec. Slowdowns = 0
Loop 1: DELETE time 2.0 secs, 527.9 deletes/sec. Slowdowns = 0
result title: name-concurrency-size, uploadspeed, downloadspeed
result csv: 127-1-1k,0.34,0.35

```

图 8 s3- benchmark 的命令界面

之后每次改变一个参数，分析各参数对于性能的影响。

在参数 t 为 1,z 为 1k 的情况下，改变 d 的数据，得到的数据如下所示。

表 2 只改变参数 d 时各项数据的测试表

d	put time	objects	speed kb/s	operations/s	get time	obj	speed	operations/s	delete time	delete/s
1	1	330	329.7	329.7	0.9	330	361.1	361.1	0.7	473.9
2	2	721	360.1	360.1	2	589	294.3	294.3	1.8	405.1
3	3	919	306.1	306.1	3	889	296.3	296.3	2	468.5
4	4	906	226.4	226.4	3.2	906	282.5	282.5	2.1	427.4
5	5	1454	290.6	290.6	5	1290	257.9	257.9	4.1	392.5
6	6	1573	262	262	5.1	1573	305.8	305.8	3.6	431.5
7	7	2266	323.7	323.7	7	2130	304.3	304.3	5.2	438.2
8	8	2777	347.1	347.1	8	1778	222.2	222.2	6.8	405.5
9	9	2848	316.4	316.4	9	2506	278.4	278.4	6.9	411.3
20	20	5946	297.3	297.3	20	5443	272.1	272.1	13.4	444.4

根据数据做出统计图

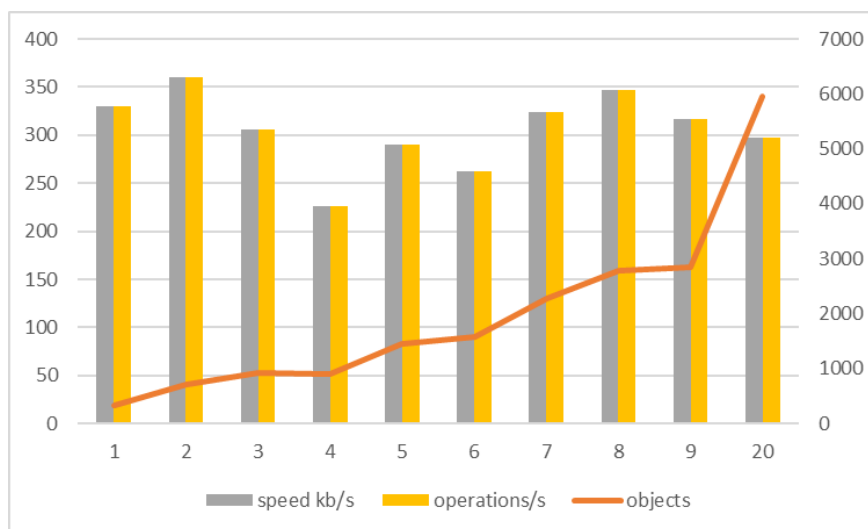


图 9 PUT 操作时各项数据随 d 的变化图

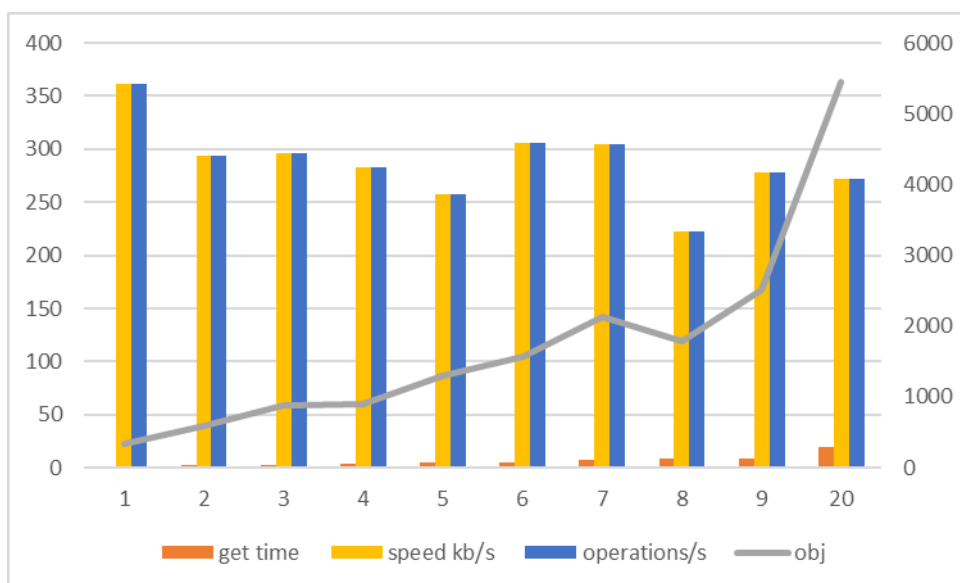


图 10 GET 操作时各项数据随 d 的变化图

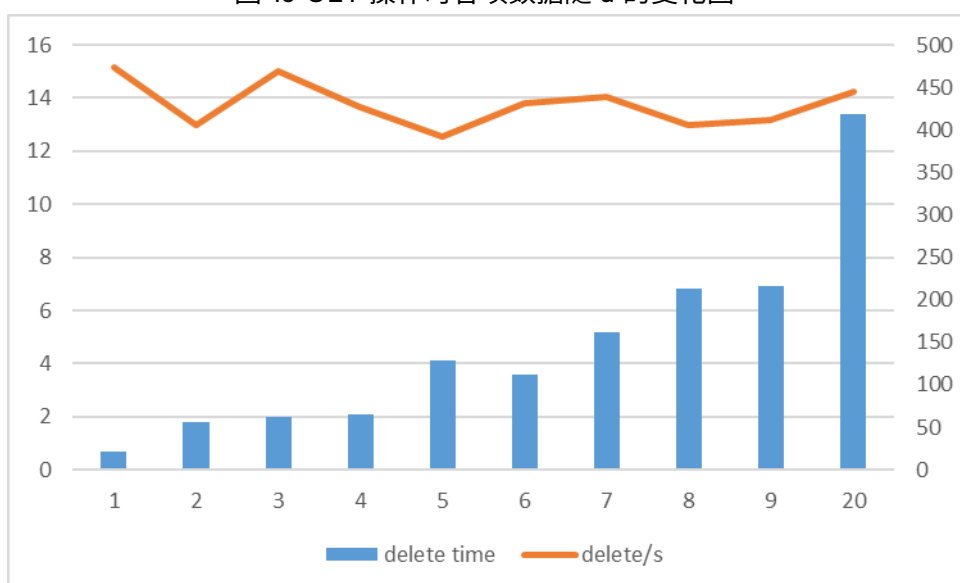


图 11 delete 操作时各项数据随 d 的变化图

从以上的数据图表中可以看出，随着操作时间的增长，所能进行的总的操作数是在不断增加的，但是每秒钟所能进行的操作数目大致相同。总体来看，put 与 get 操作的速度大致相同，而进行删除操作时的速度快于其他操作。

之后令 $d=3, z$ 为 1k 的情况下，改变参数 t 的值，即线程数。得到的数据统计表如下所示

表 3 只改变参数 t 时各项数据的测试表

t	put time	objects	speed kb/s	operations/s	get time	obj	speed	operations/s	delete time	delete/s
1	3	1104	366.6	366.6	3	948	315.8	315.8	2.3	484.3
2	3	1117	372.5	372.5	3	1033	343.2	343.2	2.2	491.6
3	3	1170	389.5	389.5	3	1073	356.5	356.5	2.2	540
4	3	1168	388.5	388.5	3	1105	367.7	367.7	2.1	562.2
5	3	1174	390.3	390.3	3	1086	361.1	361.1	2.1	554.9

根据统计表，做出统计图如下所示。



图 12 PUT 操作时各项数据随 t 的变化图

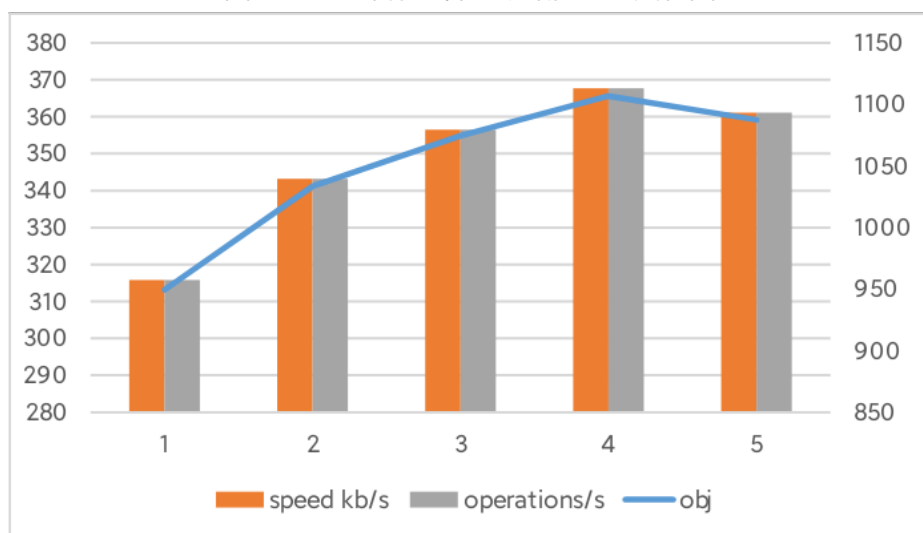


图 13 GET 操作时各项数据随 t 的变化图

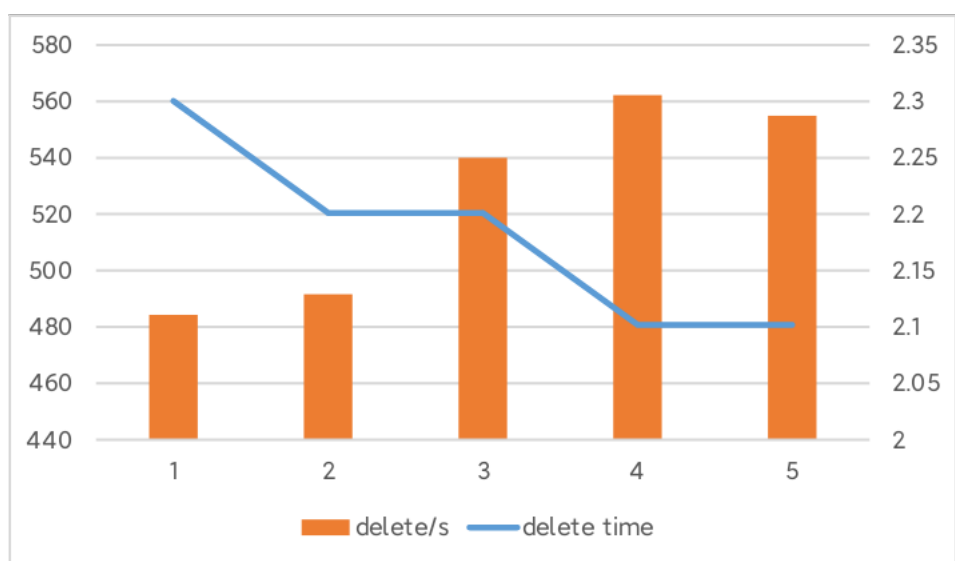


图 14 delete 操作时各项数据随 t 的变化图

从以上的数据分析，可以得出，随着线程数的增加，进行各项操作时的速度都有所增加，但是在 t 大于 4 之后，这种增长的趋势开始变得不明显，甚至有下降的趋势，因此不是线程数越大越好。

之后令 $d=3, t$ 为 1 的情况下，改变参数 z 的值，即每个测试文件的大小。得到的数据统计表如下所示。

表 4 只改变参数 z 时各项数据的测试表

z	put time	objects	speed mb/s	operations/s	get time	obj	speed	operations/s	delete time	delete/s
1k	3	1100	0.36	366.4	3	1036	344.9	344.9	2.4	455
3k	3	1151	1.1	383.4	3	1036	1	345.1	2.3	490.4
10k	3	1133	3.69	377.6	3	1047	3.4	348.8	2.2	504.9
100k	3	868	28.25	283.3	2.7	868	30.85	315.9	1.9	445.9
1m	3	331	110.1	110.1	1.3	331	251.9	251.9	0.9	375.7

根据统计表，做出统计图如下所示。

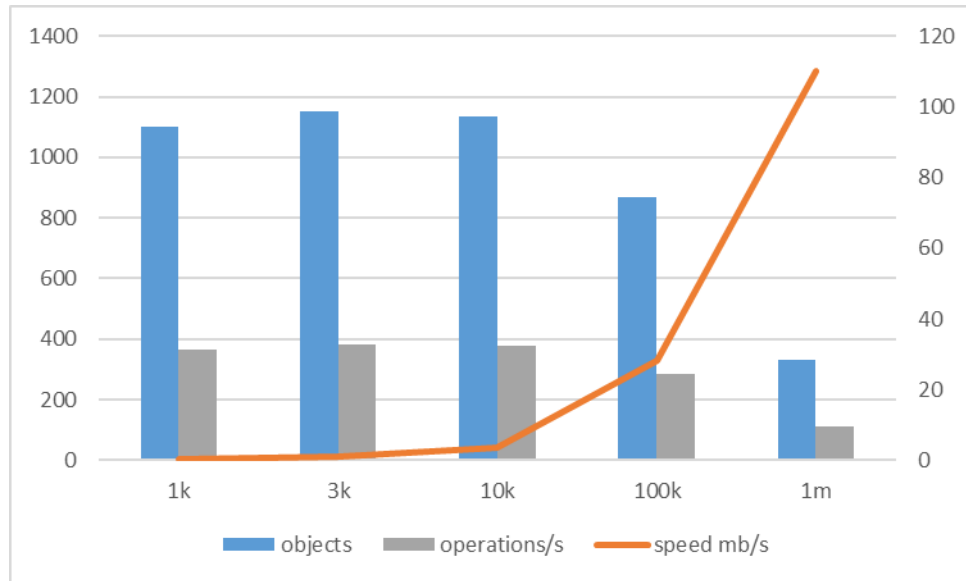


图 15 PUT 操作时各项数据随 z 的变化图

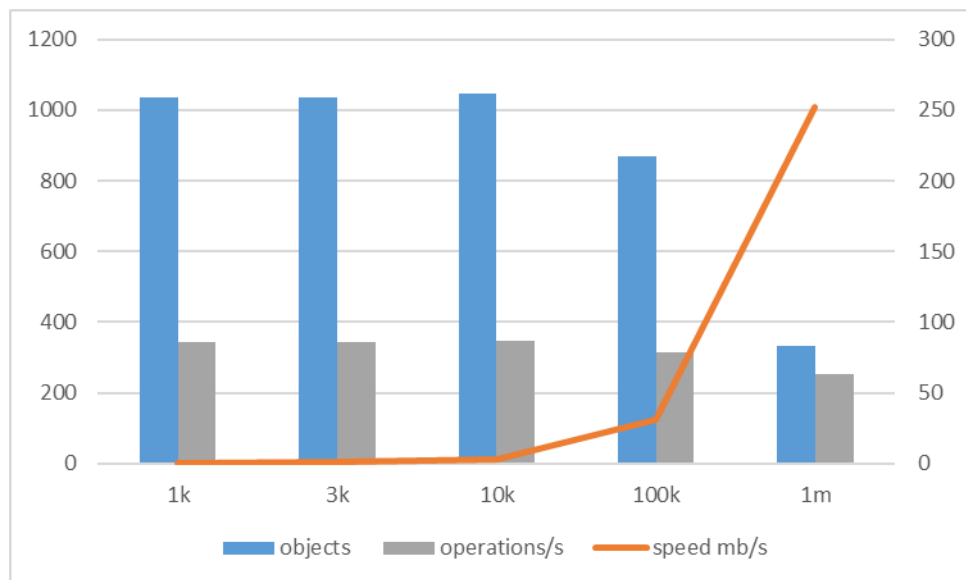


图 16 GET 操作时各项数据随 z 的变化图

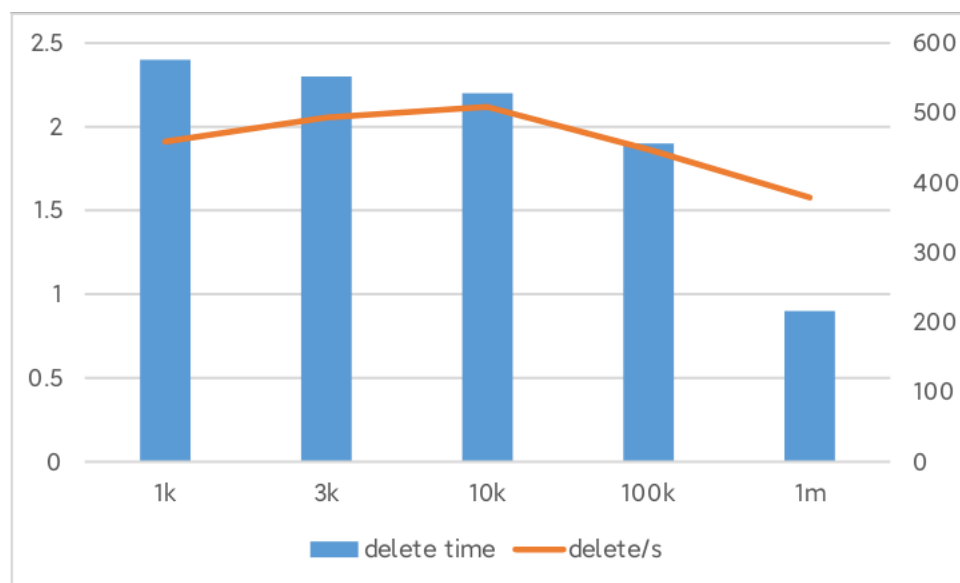


图 17 delete 操作时各项数据随 z 的变化图

从以上的统计图表中分析发现，随着对象大小的增加，一定时间内能够操作的对象数目明显下降，但是速度显著提升。get 与 delete 操作所需时间都明显下降。

六、实验总结

通过这次的实验课程，让我学习到了对象存储技术的相关知识，并动手实践搭建了对象存储的服务端和客户端，并通过 s3- benchmark 评测工具，通过改变参数的方式，对存储中的各项数据进行了测试和分析。

在这次实验的过程中，也遇到了一些问题，主要是在环境配置方面的，通过查阅网上的资料，最终还是得到了解决，并且在这个过程中学习到了很多，这些收获也会在今后的学习中对我有很大的帮助。

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.