2016 级

《物联网数据存储与管理》课程

实 验 报 告

姓　名　倪鸿

学　号　U201614889

班　号　物联网 1601 班

日　期　2019.05.28

目　　录

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

对象存储根本上改变了存储蓝图。它处理和解决了曾经被认为是棘手的存储问题：不间断可扩展性、弹性下降、限制数据持久性、无限技术更新和成本失控。

存储领域的专家对其潜在的优势感到兴奋，尤其是他们的绝大多数数据都碰巧是被动的或者是冷数据。

三、实验环境

实验平台为 Windows10 操作系统：

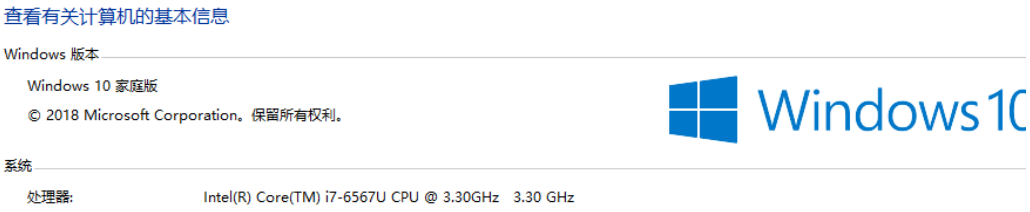


图 1 Win10 系统

配置 Python3.7.3 环境：

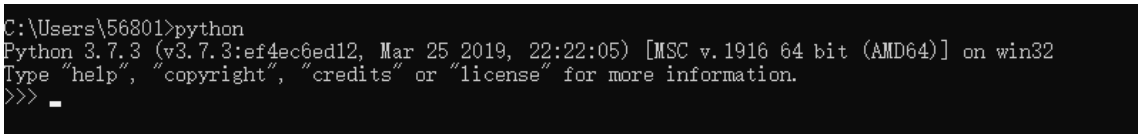


图 2 Python 环境配置成功

配置 Go 语言环境：

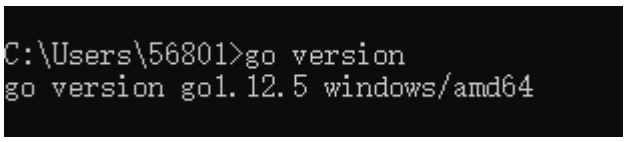


图 3 Go 语言环境配置成功

四、实验内容

4.1 对象存储技术实践

1. 采用 Minio 原生 Server 及 Client，配置成功后在 COSBench 上进行测试；
2. 采用 Minio 原生 Server 及 Client，配置成功后在 s3-benchmark 上进行测试。

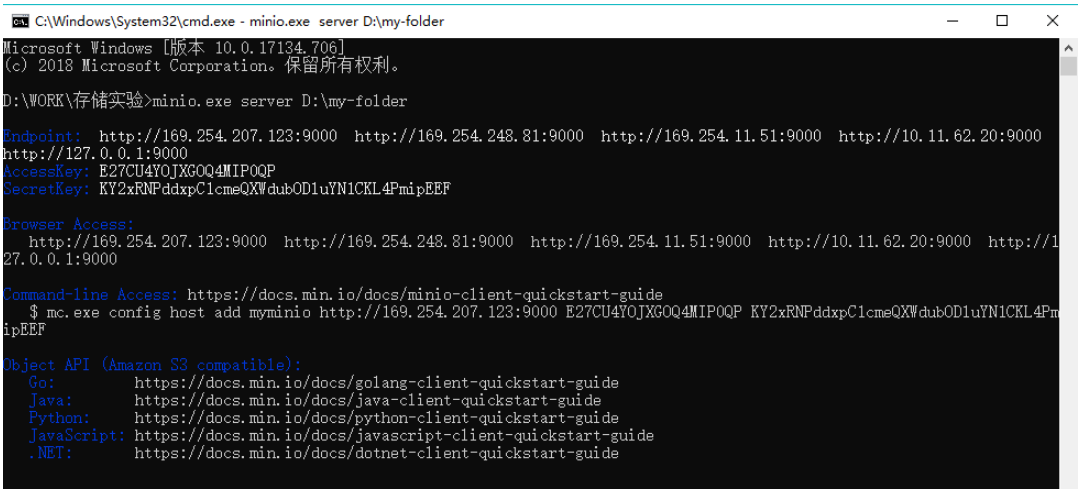
4.2 对象存储性能分析

1. 进行 s3-benchmark 上的测试时，通过改变输入参数，观察存储性能的变化；
2. 测试结果分析。

五、实验过程

以下为一些前期的安装及配置流程，较为简单就不再赘述。

Server 用的是 Minio 自带的 Server：



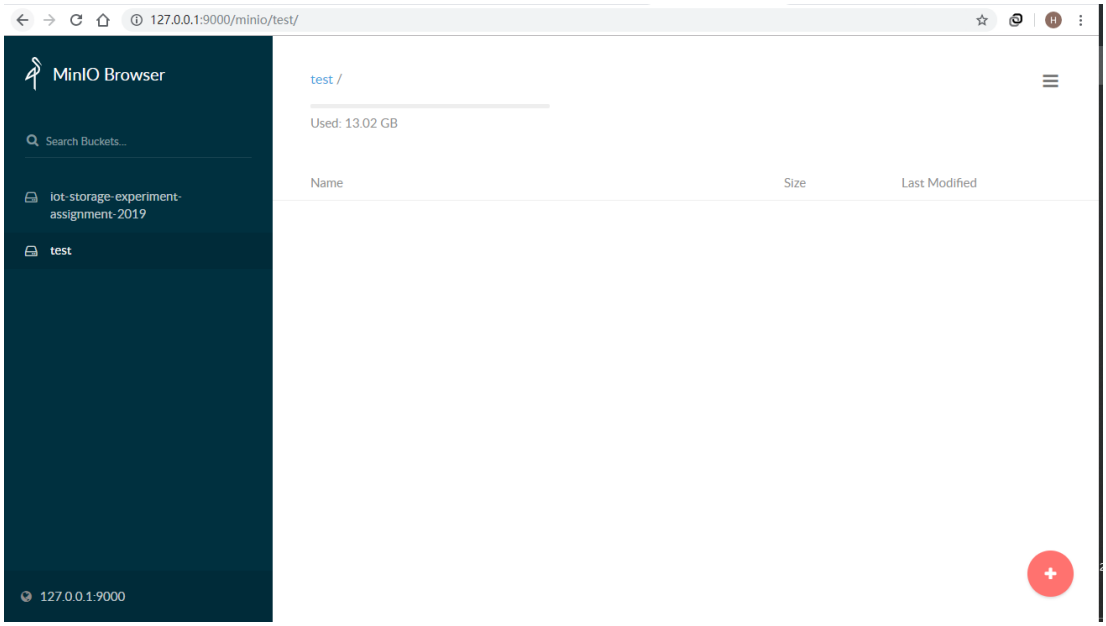


图 4 Minio-Server 配置成功

Client 是 Minio 自带的 mc.exe 以及 aws-shell：

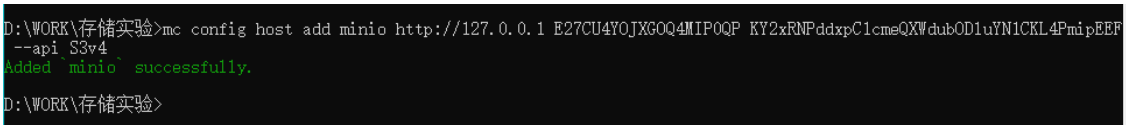


图 5 MC 配置成功

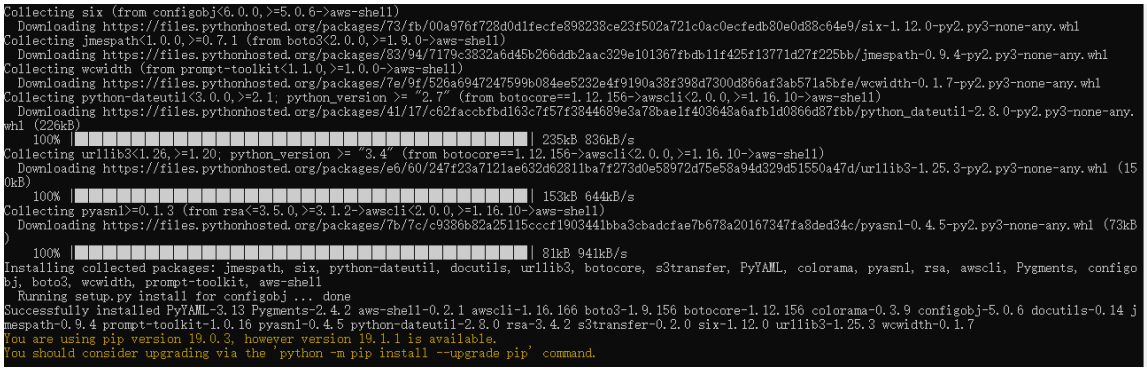


图 6 aws-shell 安装成功

最后是测试部分，我采用了 COSBench 及 s3-benchmark：

1. COSBench 测试：

先在 COSBench 载入包中点击 start-all.bat，打开控制和驱动服务端口，如下图所示：

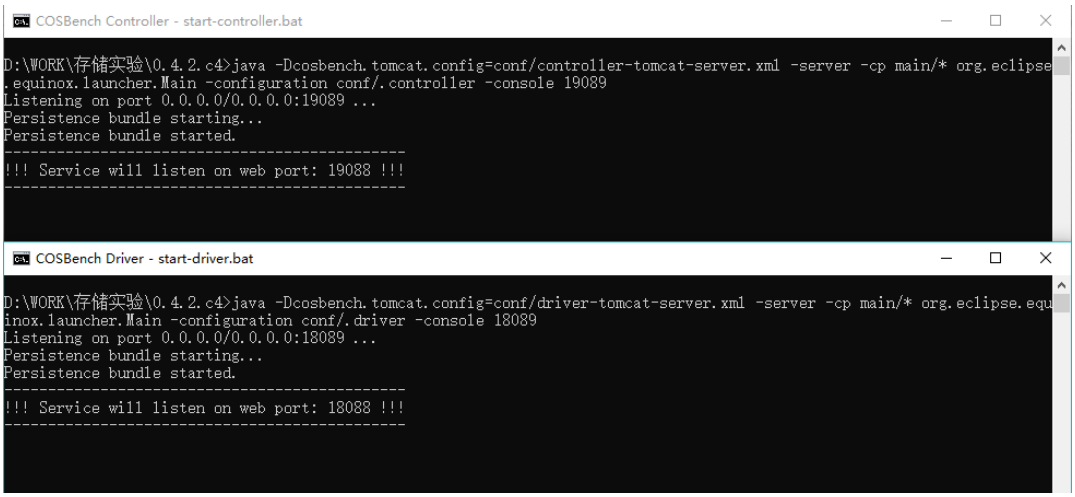
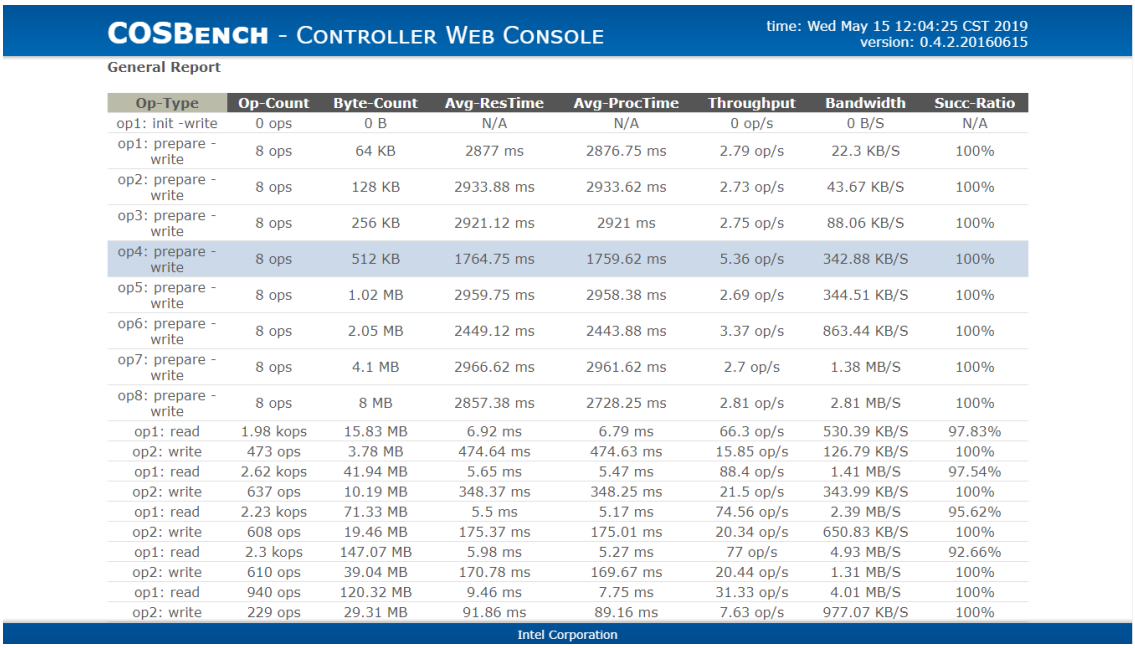


图 7 打开 COSBench 服务端口

然后点击 web.bat 即可打开 COSBench 站点：http://127.0.0.1:19088/controller/index.html，如下图所示：

图 8 COSBench 配置成功

接着点击“submit new workloads”即可上传测试文件进行测试，测试部分结果如下：



Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
op1: init -write	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
op1: prepare - write	8 ops	64 KB	2877 ms	2876.75 ms	2.79 op/s	22.3 KB/S	100%
op2: prepare - write	8 ops	128 KB	2933.88 ms	2933.62 ms	2.73 op/s	43.67 KB/S	100%
op3: prepare - write	8 ops	256 KB	2921.12 ms	2921 ms	2.75 op/s	88.06 KB/S	100%
op4: prepare - write	8 ops	512 KB	1764.75 ms	1759.62 ms	5.36 op/s	342.88 KB/S	100%
op5: prepare - write	8 ops	1.02 MB	2959.75 ms	2958.38 ms	2.69 op/s	344.51 KB/S	100%
op6: prepare - write	8 ops	2.05 MB	2449.12 ms	2443.88 ms	3.37 op/s	863.44 KB/S	100%
op7: prepare - write	8 ops	4.1 MB	2966.62 ms	2961.62 ms	2.7 op/s	1.38 MB/S	100%
op8: prepare - write	8 ops	8 MB	2857.38 ms	2728.25 ms	2.81 op/s	2.81 MB/S	100%
op1: read	1.98 kops	15.83 MB	6.92 ms	6.79 ms	66.3 op/s	530.39 KB/S	97.83%
op2: write	473 ops	3.78 MB	474.64 ms	474.63 ms	15.85 op/s	126.79 KB/S	100%
op1: read	2.62 kops	41.94 MB	5.65 ms	5.47 ms	88.4 op/s	1.41 MB/S	97.54%
op2: write	637 ops	10.19 MB	348.37 ms	348.25 ms	21.5 op/s	343.99 KB/S	100%
op1: read	2.23 kops	71.33 MB	5.5 ms	5.17 ms	74.56 op/s	2.39 MB/S	95.62%
op2: write	608 ops	19.46 MB	175.37 ms	175.01 ms	20.34 op/s	650.83 KB/S	100%
op1: read	2.3 kops	147.07 MB	5.98 ms	5.27 ms	77 op/s	4.93 MB/S	92.66%
op2: write	610 ops	39.04 MB	170.78 ms	169.67 ms	20.44 op/s	1.31 MB/S	100%
op1: read	940 ops	120.32 MB	9.46 ms	7.75 ms	31.33 op/s	4.01 MB/S	100%
op2: write	229 ops	29.31 MB	91.86 ms	89.16 ms	7.63 op/s	977.07 KB/S	100%

图 9 COSBench 测试结果

可以看到，COSBench 的测试结果十分齐全，包含了诸如平均处理时间、读写速率和吞吐率等信息。

2. s3-benchmark 测试：

因为第一个实验已经配置好了服务器和客户端，所以用 s3-benchmark 测试时只需保证 Go 语言的环境，然后从 Github 上 go get 得到 s3-benchmark 即可。

```
C:\Users\56801>go get -u github.com/chinglinwen/s3-benchmark
```

图 10 go get 获取 s3-benchmark

然后设置参数 bucket 为 test，上传时间 duration 为 3s，线程数 threads 为 1，数据大小 object 为 1K，进行测试。如图 11 所示，测试结果为：put 速率 24KB/sec、吞吐率 24.0；get 速率 379.2KB/sec、吞吐率 379.2。

```
C:\Users\56801>s3-benchmark -a E27CU4YOJXGOQ4MIP0QP -s KY2xRNPddxpC1cmeQXWdubOD1uYN1CKL4PmipEEF -u http://127.0.0.1:9000 -b test -d 3 -t 1 -z 1K
Wasabi benchmark program v2.0
Parameters: url=http://127.0.0.1:9000, bucket=test, region=us-east-1, duration=3, threads=1, loops=1, size=1K
Loop 1: PUT time 3.0 secs, objects = 73, speed = 24KB/sec, 24.0 operations/sec. Slowdowns = 0
Loop 1: GET time 0.2 secs, objects = 73, speed = 379.2KB/sec, 379.2 operations/sec. Slowdowns = 0
Loop 1: DELETE time 0.1 secs, 522.8 deletes/sec. Slowdowns = 0
result title: name-concurrency-size, uloadspeed, downloadspeed
result csv: 127-1-1K,0.02,0.37
```

图 11 第一次测试结果

3. 变参数测性能：

接着我们可以通过固定 object 大小观察结果随 threads 数变化而变化、固定 threads 数观察结果随 object 大小变化而变化：

1) 固定 threads 数为 4，object 大小分别为 1K、10K、20K、30K、60K、120K，得到如下结果：

```
C:\Users\56801>s3-benchmark -a E27CU4YOJXGOQ4MIP0QP -s KY2xRNPddxpC1cmeQXWdubOD1uYN1CKL4PmipEEF -u http://127.0.0.1:9000 -b test -d 3 -t 4 -z 1K
Wasabi benchmark program v2.0
Parameters: url=http://127.0.0.1:9000, bucket=test, region=us-east-1, duration=3, threads=4, loops=1, size=1K
Loop 1: PUT time 3.1 secs, objects = 95, speed = 30.9KB/sec, 30.9 operations/sec. Slowdowns = 0
Loop 1: GET time 0.3 secs, objects = 380, speed = 1.5MB/sec, 1500.1 operations/sec. Slowdowns = 0
Loop 1: DELETE time 0.1 secs, 1323.0 deletes/sec. Slowdowns = 0
result title: name-concurrency-size, uloadspeed, downloadspeed
result csv: 127-4-1K,0.03,1.46

C:\Users\56801>s3-benchmark -a E27CU4YOJXGOQ4MIP0QP -s KY2xRNPddxpC1cmeQXWdubOD1uYN1CKL4PmipEEF -u http://127.0.0.1:9000 -b test -d 3 -t 4 -z 10K
Wasabi benchmark program v2.0
Parameters: url=http://127.0.0.1:9000, bucket=test, region=us-east-1, duration=3, threads=4, loops=1, size=10K
Loop 1: PUT time 3.0 secs, objects = 113, speed = 370.9KB/sec, 37.1 operations/sec. Slowdowns = 0
Loop 1: GET time 0.3 secs, objects = 452, speed = 15.1MB/sec, 1546.8 operations/sec. Slowdowns = 0
Loop 1: DELETE time 0.2 secs, 703.7 deletes/sec. Slowdowns = 0
result title: name-concurrency-size, uloadspeed, downloadspeed
result csv: 127-4-10K,0.36,15.11

C:\Users\56801>s3-benchmark -a E27CU4YOJXGOQ4MIP0QP -s KY2xRNPddxpC1cmeQXWdubOD1uYN1CKL4PmipEEF -u http://127.0.0.1:9000 -b test -d 3 -t 4 -z 20K
Wasabi benchmark program v2.0
Parameters: url=http://127.0.0.1:9000, bucket=test, region=us-east-1, duration=3, threads=4, loops=1, size=20K
Loop 1: PUT time 3.1 secs, objects = 111, speed = 714.9KB/sec, 35.7 operations/sec. Slowdowns = 0
Loop 1: GET time 0.3 secs, objects = 444, speed = 26.9MB/sec, 1378.3 operations/sec. Slowdowns = 0
Loop 1: DELETE time 0.1 secs, 849.6 deletes/sec. Slowdowns = 0
result title: name-concurrency-size, uloadspeed, downloadspeed
result csv: 127-4-20K,0.70,26.92

C:\Users\56801>s3-benchmark -a E27CU4YOJXGOQ4MIP0QP -s KY2xRNPddxpC1cmeQXWdubOD1uYN1CKL4PmipEEF -u http://127.0.0.1:9000 -b test -d 3 -t 4 -z 30K
Wasabi benchmark program v2.0
Parameters: url=http://127.0.0.1:9000, bucket=test, region=us-east-1, duration=3, threads=4, loops=1, size=30K
Loop 1: PUT time 3.1 secs, objects = 116, speed = 1.1MB/sec, 37.2 operations/sec. Slowdowns = 0
Loop 1: GET time 0.3 secs, objects = 464, speed = 41.4MB/sec, 1414.1 operations/sec. Slowdowns = 0
Loop 1: DELETE time 0.1 secs, 1038.5 deletes/sec. Slowdowns = 0
result title: name-concurrency-size, uloadspeed, downloadspeed
result csv: 127-4-30K,1.09,41.43
```

图 12 固定 threads 数目，改变 object 大小（部分结果）

put		
object大小（K）	put速率（KB/S）	put吞吐率
1	30.9	30.9
10	370.9	37.1
20	714.9	35.7
30	1126.4	37.2
60	2150.4	36.4
120	3891.2	32.6
get		
object大小（K）	get速率（KB/S）	get吞吐率
1	1536	1500.1
10	15462.4	1546.8
20	27545.6	1378.3
30	42393.6	1414.1
60	75264	1255
120	128716.8	1072.9

图 13 excel 总结

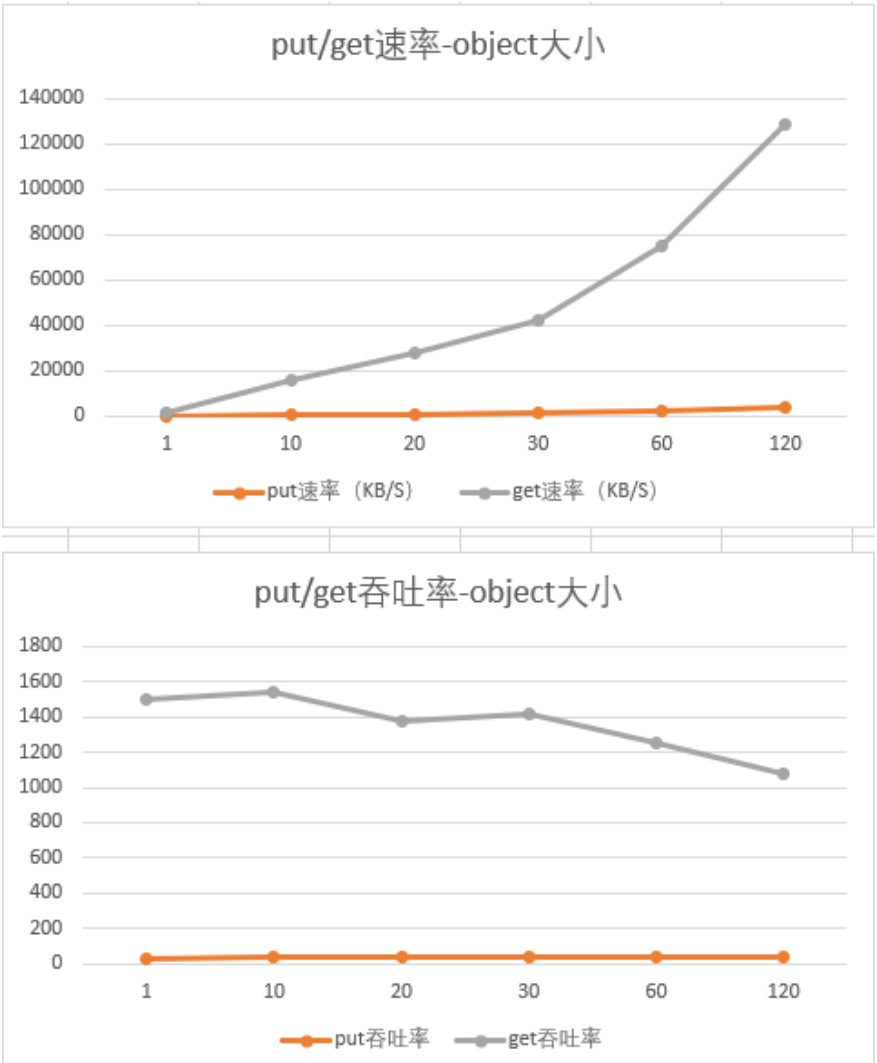


图 14 对应折线图

由图 13 和图 14 不难得出结论，固定线程数目，随着 object 的增大，put/get 的速率确实都是在增大的，但吞吐率却呈现出一种先增后降再增再降的过程。由于最后设置的 object 大小为 120K 相比其他数据已经足够大了，而其对应的吞吐率又确实更小了，可见“先增后降再增再降”的现象是比较可信的，而这也表明了吞吐率并不是随 object 增大就增大的，而是有一上界，且对应一个特殊的 object 大小。

2) 固定 object 大小为 10K，threads 数量分别为 1、5、10、15、30、60，得到如下结果：

```
C:\Users\56801>s3-benchmark -a E27CU4YOJXGOQ4MIP0QP -s KY2xRNPddxpC1cmeQXWdubOD1uYN1CKL4PmipEEF -u http://127.0.0.1:9000 -b test -d 3 -t 1 -z 10K
Wasabi benchmark program v2.0
Parameters: url=http://127.0.0.1:9000, bucket=test, region=us-east-1, duration=3, threads=1, loops=1, size=10K
Loop 1: PUT time 3.0 secs, objects = 45, speed = 148.5KB/sec, 14.9 operations/sec. Slowdowns = 0
Loop 1: GET time 0.2 secs, objects = 45, speed = 2.9MB/sec, 298.8 operations/sec. Slowdowns = 0
Loop 1: DELETE time 0.1 secs, 392.3 deletes/sec. Slowdowns = 0
result title: name-concurrency-size, uloadspeed, downloadspeed
result csv: 127-1-10K,0.15,2.92

C:\Users\56801>s3-benchmark -a E27CU4YOJXGOQ4MIP0QP -s KY2xRNPddxpC1cmeQXWdubOD1uYN1CKL4PmipEEF -u http://127.0.0.1:9000 -b test -d 3 -t 5 -z 10K
Wasabi benchmark program v2.0
Parameters: url=http://127.0.0.1:9000, bucket=test, region=us-east-1, duration=3, threads=5, loops=1, size=10K
Loop 1: PUT time 3.1 secs, objects = 108, speed = 347.5KB/sec, 34.7 operations/sec. Slowdowns = 0
Loop 1: GET time 0.4 secs, objects = 540, speed = 13.1MB/sec, 1340.7 operations/sec. Slowdowns = 0
Loop 1: DELETE time 0.1 secs, 887.6 deletes/sec. Slowdowns = 0
result title: name-concurrency-size, uloadspeed, downloadspeed
result csv: 127-5-10K,0.34,13.09

C:\Users\56801>s3-benchmark -a E27CU4YOJXGOQ4MIP0QP -s KY2xRNPddxpC1cmeQXWdubOD1uYN1CKL4PmipEEF -u http://127.0.0.1:9000 -b test -d 3 -t 10 -z 10K
Wasabi benchmark program v2.0
Parameters: url=http://127.0.0.1:9000, bucket=test, region=us-east-1, duration=3, threads=10, loops=1, size=10K
Loop 1: PUT time 3.2 secs, objects = 123, speed = 381.1KB/sec, 38.1 operations/sec. Slowdowns = 0
Loop 1: GET time 0.8 secs, objects = 1230, speed = 14.7MB/sec, 1502.2 operations/sec. Slowdowns = 0
Loop 1: DELETE time 0.3 secs, 489.4 deletes/sec. Slowdowns = 0
result title: name-concurrency-size, uloadspeed, downloadspeed
result csv: 127-10-10K,0.37,14.67

C:\Users\56801>s3-benchmark -a E27CU4YOJXGOQ4MIP0QP -s KY2xRNPddxpC1cmeQXWdubOD1uYN1CKL4PmipEEF -u http://127.0.0.1:9000 -b test -d 3 -t 15 -z 10K
Wasabi benchmark program v2.0
Parameters: url=http://127.0.0.1:9000, bucket=test, region=us-east-1, duration=3, threads=15, loops=1, size=10K
Loop 1: PUT time 3.3 secs, objects = 110, speed = 338KB/sec, 33.8 operations/sec. Slowdowns = 0
Loop 1: GET time 1.1 secs, objects = 1650, speed = 15.2MB/sec, 1560.3 operations/sec. Slowdowns = 0
Loop 1: DELETE time 0.1 secs, 934.7 deletes/sec. Slowdowns = 0
result title: name-concurrency-size, uloadspeed, downloadspeed
result csv: 127-15-10K,0.33,15.24
```

图 15 固定 object 大小，改变 threads 数目（部分结果）

put		
threads数目	put速率（KB/S）	put吞吐率
1	148.5	14.9
5	347.5	34.7
10	381.1	38.1
15	338	33.8
30	368.2	36.8
60	360.1	36
get		
threads数目	get速率（KB/S）	get吞吐率
1	2969.6	298.8
5	13414.4	1340.7
10	15052.8	1502.2
15	15564.8	1560.3
30	15052.8	1506.8
60	15462.4	1544.9

图 16 excel 总结（2）

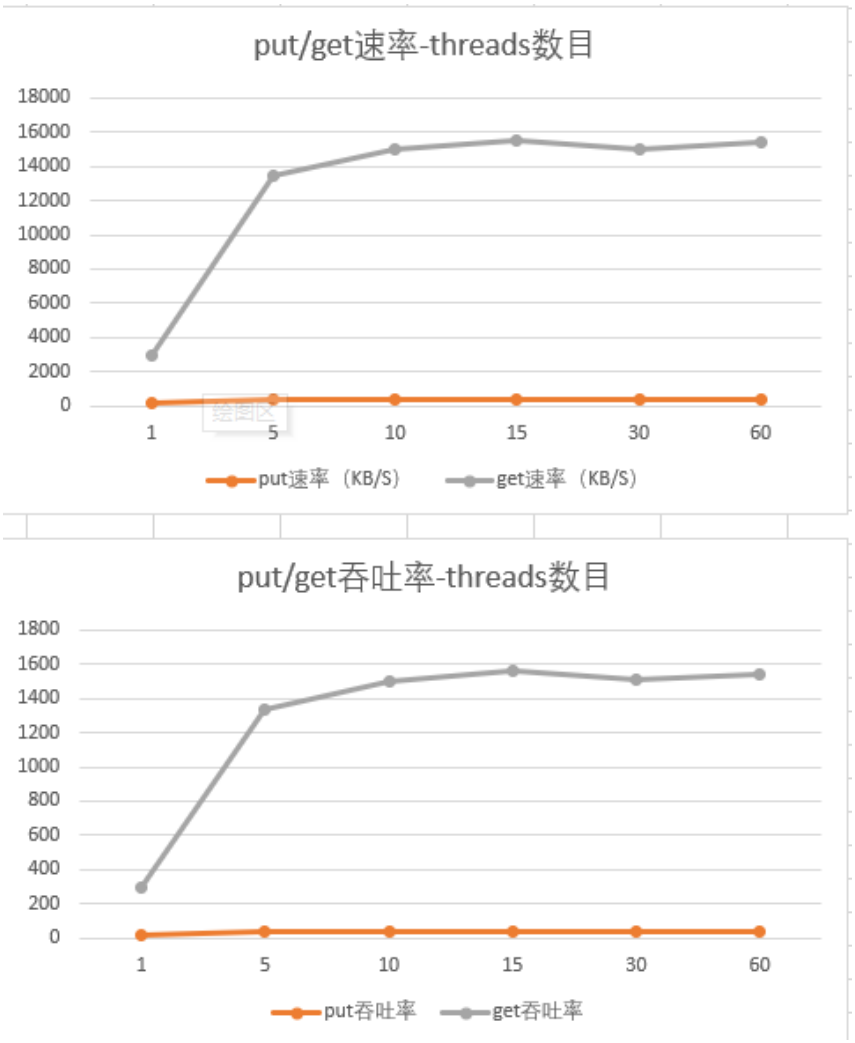


图 17 对应折线图（2）

由图 16 和图 17 不难发现，put/get 的吞吐率均随着线程的数量增加而增加，且具体表现为一开始的迅速增加和之后的逐渐平稳，最后稳定在一定的数值范围内上下波动。

4. 其他：

另外还配置了 mock-s3 作为服务器，配置过程无非是从 Github 上下载源码并依照 README 中的提示进行操作，最后的到图 19 所示结果即为配置成功。

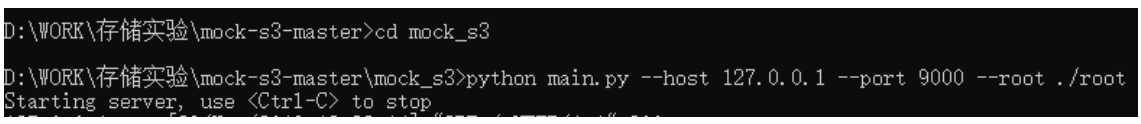


图 18 mock-s3 配置成功

接着测试服务器 127.0.0.1:9000，如下图所示：

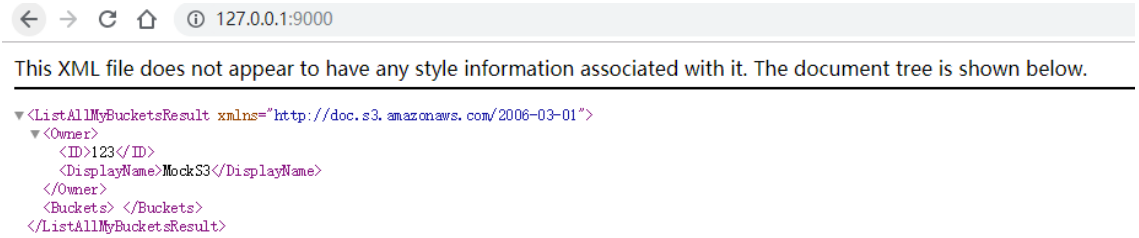


图 19 测试 127.0.0.1:9000

```
C:\Users\56801>s3-benchmark -a 123 -s MockS3 -u http://127.0.0.1:9000 -b  -d 3 -t 1 -z 1K
Wasabi benchmark program v2.0
Parameters: url=http://127.0.0.1:9000, bucket=-d, region=us-east-1, duration=60, threads=1, loops=1, size=1M
Loop 1: PUT time 60.0 secs, objects = 1354, speed = 22.6MB/sec, 22.6 operations/sec. Slowdowns = 0
Loop 1: GET time 31.1 secs, objects = 1354, speed = 43.6MB/sec, 43.6 operations/sec. Slowdowns = 0
Loop 1: DELETE time 4.4 secs, 310.0 deletes/sec. Slowdowns = 0
result title: name-concurrency-size, uloadspeed, downloadspeed
result csv: 127-1-1M,22.55,43.59
```

图 20 测试 s3-benchmark

虽然好像是测试成功了（花了大约 1 分钟，远远长于之前用 Minio 的 Server 做测试的时候），不过不太明白 mock-s3 的密钥在哪以及 bucket 如何创建。

六、实验总结

本次实验通过 Github 上的各种教程，详细全面地指引笔者探索基于对象存储的全新领域。实验从 minio 入手，先熟悉其自带的服务器、客户端，再给出诸如 s3proxy、aws-shell 等“高级”替代版本，由浅入深，帮助我们更好地掌握对象存储的内容要领。可以说是一次非常好学习体验。

实验中遇到的问题不多，因为详细的配置样例网上均已给出（除了配 MC 时 MINIO 的翻译网站给出的配置代码有误，少了-api）。试着尝试在 ubuntu 上跑 s3bench 进行测试，但由于 ubuntu 的版本太低，相应的 Go 语言环境版本也才 1.2，所以又回到了 win 系统。Win 系统的 go get 及其缓慢，尝试了更改 go 的环境配置也不能解决问题，后来发现可能是 DNS 相关的问题，所以用管理员权限修改了 hosts.txt 内的内容就没问题了。

最后，感谢老师在本次实验中给予的帮助和指导。

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.