

2017 级

《物联网数据存储与管理》课程

# 实 验 报 告

姓 名 胡晗

学 号 U201714518

班 号 物联网 1701 班

日 期 2020.06.01

# 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.....	2
4.2 对象存储性能分析.....	2
五、实验过程.....	2
六、实验总结.....	14
参考文献.....	15

## 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

物联网应用随着移动互联网的兴起而蓬勃发展，快速增长的物联网数据带来了存储挑战：

1. 庞大：以几何级数不断增长的数据量，对存储和处理带来挑战。
2. 复杂：数据内容多样性带来的数据异构，使得数据结构越来越复杂。

因此，需要一个高可用、可扩展的海量数据存储系统来满足物联网存储需求。面向对象存储技术提供了一种解决方案。对象存储同兼具 SAN 高速直接访问磁盘特点及 NAS 的分布式共享特点。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。

## 三、实验环境

**CPU：** Intel® Core™ i5-6200U CPU @ 2.30GHz × 4

**内存：** 7.7 GiB

**硬盘：** HDD 170GB

**操作系统：** ubuntu 16.04 LTS

**Go：** go1.14.2 linux/amd64

## 四、实验内容

根据实验给出的教程，选择对象存储服务端、对象存储客户端、对象存储评测工具。

#### 4.1 对象存储技术实践

1. 搭建对象存储服务端，选择使用 Minio 和 mock-s3。
2. 搭建对象存储客户端，选择使用 osm。
3. 测试对象存储基本功能是否正常。

#### 4.2 对象存储性能分析

1. 搭建对象评测工具环境，选择 S3 Bench，作为 Go 的一个模块进行安装。
2. 调整对象存储评测参数，包括客户端数量、对象数量、对象大小，观察数据存储性能。
3. 编写 shell 脚本实现批量测试，重定向 bash 终端输出结果到文本文件，以便进一步分析。
4. 整理和分析不同参数下存储性能数据。

### 五、实验过程

#### 5.1 搭建对象存储服务端

1. Minio 服务端是开箱即用的，只需下载二进制可执行文件到本地后修改执行权限即可，执行运行脚本，以指定的访问密钥和安全密钥运行服务端。

```
huhan@luhan: ~/i ot Storage/obs-tutorial $ ./run-minio.sh
master
Endpoint: http://192.168.1.102:9000 http://127.0.0.1:9000
AccessKey: hust
SecretKey: hust_obs
Browser Access:
http://192.168.1.102:9000 http://127.0.0.1:9000
Command Line Access: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc config host add myminio http://192.168.1.102:9000 hust hust_obs
Object API (Amazon S3 compatible):
Go: https://docs.min.io/docs/golang-client-quickstart-guide
Java: https://docs.min.io/docs/java-client-quickstart-guide
Python: https://docs.min.io/docs/python-client-quickstart-guide
JavaScript: https://docs.min.io/docs/javascript-client-quickstart-guide
.NET: https://docs.min.io/docs/dotnet-client-quickstart-guide
```

图 1 运行 minio 服务端

2. mock-s3 服务端是基于 Python 的，搭建好 Python 环境后，在 mock-s3 的程序包内运行 setup.py 即可成功安装。由于 mock-s3 是没有访问密钥的，所以只要指定服务端口，运行执行脚本，启动服务。

```
huhan@luhan: ~/i ot Storage/mock-s3 $ ./run-mock-s3.sh
Starting server, use <Ctrl-C> to stop
```

图 2 运行 mock-s3 服务端

## 5.2 搭建对象存储客户端

1. osm 作为一个 Go 模块加载，需要从 github 获取到代码包，放到 GOPATH 目录下的 src/github.com 文件夹中，之后使用 go install 命令进行模块安装。
2. 安装完成后在 GOPATH 目录的 bin 目录下会出现 osm 的可执行文件。
3. 运行 config-osm.sh 脚本配置 osm 参数。
4. 输入 osm -h 测试是否安装配置成功。

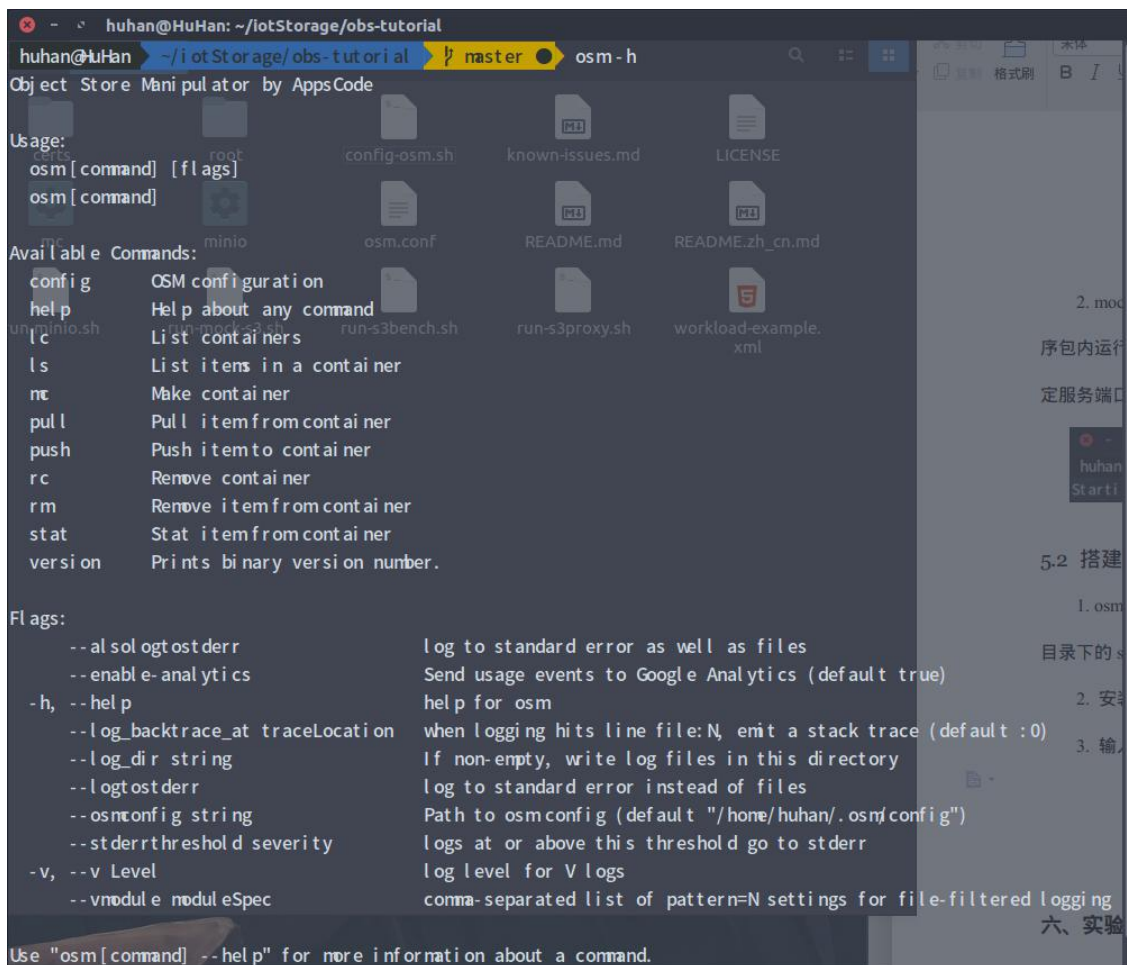


图 3 检查 osm 安装配置情况

## 5.3 测试对象存储功能

1. 运行 minio 服务端，在浏览器打开图形管理界面，访问 127.0.0.1:9000，输入密钥，即可查看服务端情况。

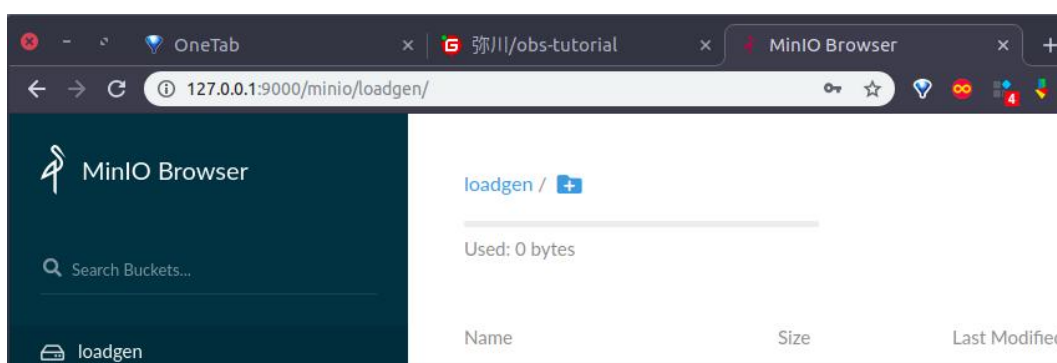


图 4 minio 服务端图形界面

2. 利用 osm 上传文件到 minio 服务端。



图 5 osm 上传文件

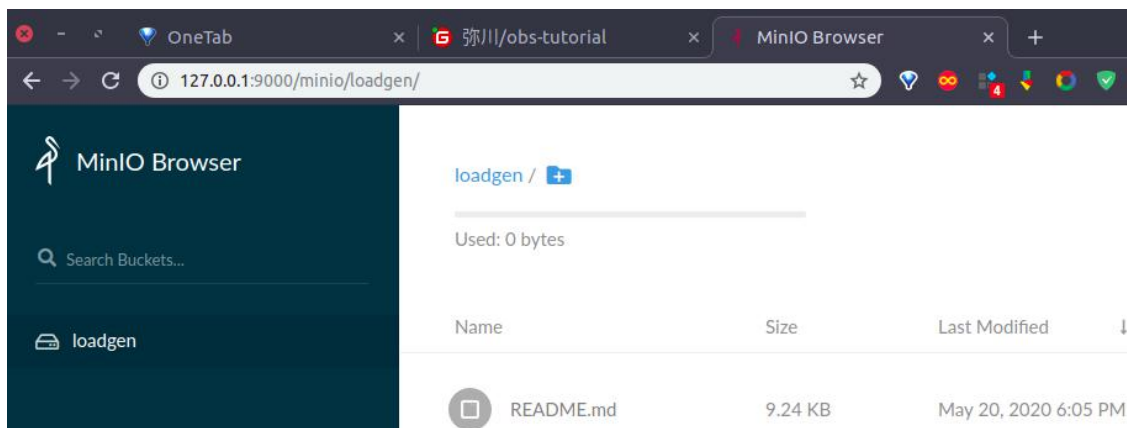


图 6 上传文件成功

3. 运行 mock-s3 服务端，在网页访问 127.0.0.1:9000.



图 7 mock-s3 服务端

## 5.4 对象存储性能分析

1. S3 Bench 作为 GO 模块加载，下载 aws-sdk-go 和 go-jmespath 依赖包进行安装，再安装 s3bench，安装完成在 GOPATH 的 bin 目录下会出现 s3bench 可执行文件。

2. S3 Bench 没有批量测试和输出数据为文件的功能，编写 shell 脚本进行批量测试，将 bash 的输出重定向到文件，测试脚本如下：

①对象存储大小从 1KB 增加到 1MB，并发客户端数量为 1，对象数量为 100，观察对象大小对存储性能的影响。

## my-s3bench1.sh

```
#!/bin/bash
#test shell-script 1 for NumClient=1 NumSample=100 ObjectSize=[1KB,1MB]

# Locate s3bench

s3bench=~/.gopath/bin/s3bench

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

# minio on port 9000
# mock-s3 on port 9000
endpoint="http://127.0.0.1:9000"
# endpoint="http://127.0.0.1:9000"
bucket="loadgen"
ObjectNamePrefix="loadgen"
AccessKey="hust"
AccessSecret="hust_obs"
filepath="minio_server_1.txt"
# filepath="mock_s3_server_1.txt"

declare -a NumClient
declare -a NumSample
declare -a ObjectSize

NumClient=(1 1 1 1 1 1 1 1 1 1)
NumSample=(100 100 100 100 100 100 100 100 100 100)
ObjectSize=(1024 2048 4096 10240 20480 40960 102400 204800 409600 1048576)

#display run progress
progress=9

for(( i=0;i<${#NumClient[@]};i++))
do
    # run sh
    $s3bench -accessKey=$AccessKey -accessSecret=$AccessSecret -bucket=$bucket \
-endpoint=$endpoint \
-numClients=${NumClient[i]} -numSamples=${NumSample[i]} \
-objectNamePrefix=$ObjectNamePrefix -objectSize=${ObjectSize[i]} >> $filepath

    echo -e "=====➤"
```



```
$i/$progress done\n" >> $filepath
```

```
    echo -e "=====➔
```

```
$i/$progress done\n"
```

```
done
```

②并发客户端从 1 增加到 100，对象数量为 100，对象大小为 1KB，观察并发客户端数量对存储性能的影响。脚本其他部分与测试 1 相同，具体见附件。

my-s3bench2.sh

```
#!/bin/bash
```

```
#test shell-script 2 for NumClient=[1,100] NumSample=100 ObjectSize=1KB
```

```
.....
```

```
filepath="minio_server_2.txt"
```

```
# filepath="mock_s3_server_2.txt"
```

```
.....
```

```
NumClient=( 1      2      4      8      16     32     64     70     80     90     100)
```

```
NumSample=( 100    100    100    100    100    100    100    100    100    100    100)
```

```
ObjectSize=(1024 1024 1024 1024 1024 1024 1024 1024 1024 1024 1024)
```

```
#display run progress
```

```
progress=10
```

```
.....
```

③并发客户端从 1 增加到 100，对象数量为 100，对象大小为 100KB，与测试 2 内容做对比。

my-s3bench3.sh

```
#!/bin/bash
```

```
#test shell-script 3 for NumClient=[1,100] NumSample=100 ObjectSize=100KB
```

```
.....
```

```
filepath="minio_server_3.txt"
```

```
# filepath="mock_s3_server_3.txt"
```

```
.....
```

```
NumClient=( 1      2      4      8      16     32     64     70     80     90     100)
```

```
NumSample=( 100    100    100    100    100    100    100    100    100    100    100)
```

```
ObjectSize=(102400 102400 102400 102400 102400 102400 102400 102400 102400 102400 102400 102400)
```

```
#display run progress
```

```
progress=10
```

.....

④并发客户端数量为 1，对象数量从 5 增加到 1280，对象大小为 1KB，观察对象数量对存储性能的影响。

my-s3bench4.sh

```
#!/bin/bash
#test shell-script 4 for NumClient=1 NumSample=[5,1280] ObjectSize=1KB

.....

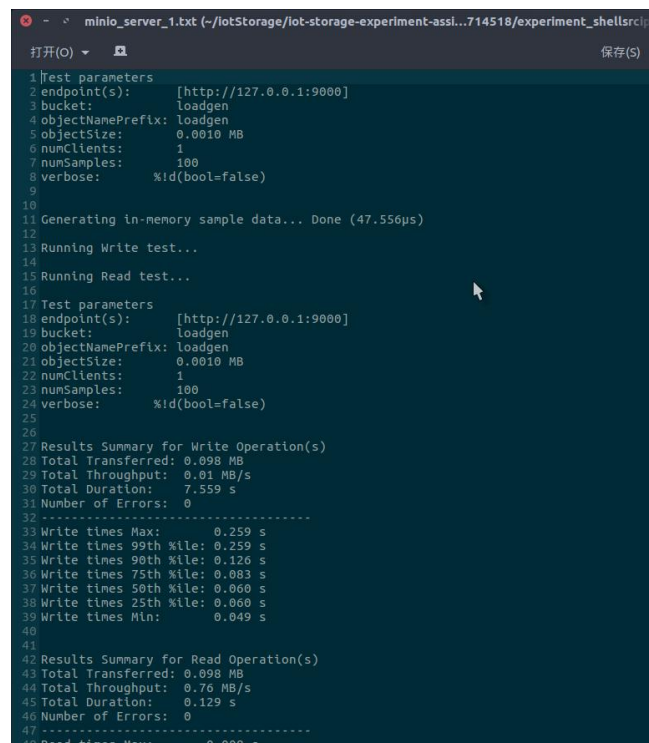
filepath="minio_server_4.txt"
# filepath="mock_s3_server_4.txt"

.....

NumClient=( 1      1      1      1      1      1      1      1      1)
NumSample=( 5      10     20     40     80     160    320    640   1280)
ObjectSize=(1024 1024 1024 1024 1024 1024 1024 1024 1024)
#display run progress
progress=8

.....
```

3. 执行测试脚本，输出重定向文件，文件内容如图所示。



```
minio_server_1.txt (~/.lotStorage/lot-storage-experiment-assl...714518/experiment_shellscrip
打开(O) 保存(S)

1[Test parameters
2 endpoint(s):      [http://127.0.0.1:9000]
3 bucket:          loadgen
4 objectNamePrefix: loadgen
5 objectSize:       0.0010 MB
6 numClients:       1
7 numSamples:       100
8 verbose:          %d(bool=false)
9
10
11 Generating in-memory sample data... Done (47.556µs)
12
13 Running Write test...
14
15 Running Read test...
16
17 Test parameters
18 endpoint(s):      [http://127.0.0.1:9000]
19 bucket:          loadgen
20 objectNamePrefix: loadgen
21 objectSize:       0.0010 MB
22 numClients:       1
23 numSamples:       100
24 verbose:          %d(bool=false)
25
26
27 Results Summary for Write Operation(s)
28 Total Transferred: 0.098 MB
29 Total Throughput:  0.01 MB/s
30 Total Duration:    7.559 s
31 Number of Errors:  0
32 -----
33 Write times Max:   0.259 s
34 Write times 99th %ile: 0.259 s
35 Write times 90th %ile: 0.126 s
36 Write times 75th %ile: 0.063 s
37 Write times 50th %ile: 0.060 s
38 Write times 25th %ile: 0.060 s
39 Write times Min:   0.049 s
40
41
42 Results Summary for Read Operation(s)
43 Total Transferred: 0.098 MB
44 Total Throughput:  0.76 MB/s
45 Total Duration:    0.129 s
46 Number of Errors:  0
47 -----
48 Read times Max:    0.008 s
```

图 8 重定向文件内容

## 4. 整理数据

#	Clients-Samples-Size	W_Throughput(MB/s)	W_Duration(s)	W_99th(s)	W_90th(s)	R_Throughput(MB/s)	R_Duration(s)	R_99th(s)	R_90th(s)
1	1-100-0.0010MB	0.01	7.559	0.259	0.126	0.76	0.129	0.008	0.002
	1-100-0.0020MB	0.03	7.678	0.138	0.116	1.33	0.146	0.005	0.004
	1-100-0.0039MB	0.05	7.412	0.182	0.127	3.06	0.127	0.005	0.003
	1-100-0.0098MB	0.12	8.158	0.198	0.127	7.42	0.132	0.009	0.003
	1-100-0.0195MB	0.23	8.585	0.186	0.160	15.08	0.130	0.005	0.004
	1-100-0.0391MB	0.56	7.031	0.138	0.125	27.83	0.140	0.006	0.004
	1-100-0.0977MB	1.59	6.136	0.137	0.061	76.33	0.128	0.005	0.004
	1-100-0.1953MB	3.19	6.116	0.118	0.061	137.73	0.142	0.005	0.004
	1-100-0.3906MB	5.65	6.908	0.160	0.074	257.23	0.152	0.006	0.004
2	1-100-1.0000MB	12.21	8.190	0.181	0.083	561.38	0.178	0.007	0.003
	1-100-0.0010MB	0.02	6.278	0.138	0.061	0.70	0.140	0.008	0.003
	2-100-0.0010MB	0.02	4.976	0.180	0.099	1.08	0.091	0.006	0.005
	4-100-0.0010MB	0.04	2.544	0.169	0.100	0.71	0.138	0.026	0.011
	8-100-0.0010MB	0.07	1.317	0.142	0.107	0.78	0.125	0.040	0.210
	16-100-0.0010MB	0.14	0.695	0.138	0.113	0.62	0.159	0.143	0.078
	32-100-0.0010MB	0.21	0.469	0.193	0.167	0.66	0.148	0.120	0.088
	64-100-0.0010MB	0.39	0.252	0.158	0.151	1.59	0.061	0.057	0.051
	70-100-0.0010MB	0.38	0.255	0.181	0.171	1.72	0.057	0.053	0.051
	80-100-0.0010MB	0.32	0.305	0.221	0.207	0.91	0.108	0.089	0.069
	90-100-0.0010MB	0.39	0.251	0.187	0.170	0.72	0.136	0.114	0.111
	100-100-0.0010MB	0.29	0.339	0.331	0.314	0.96	0.102	0.087	0.076
3	1-100-0.0977MB	1.16	8.437	0.240	0.160	67.81	0.144	0.006	0.004
	2-100-0.0977MB	1.33	7.349	0.372	0.199	85.38	0.114	0.008	0.005
	4-100-0.0977MB	2.05	4.758	0.349	0.265	71.58	0.136	0.017	0.011
	8-100-0.0977MB	3.38	2.893	0.375	0.261	76.04	0.128	0.046	0.027
	16-100-0.0977MB	6.22	1.570	0.331	0.312	63.04	0.155	0.109	0.052
	32-100-0.0977MB	9.51	1.027	0.500	0.457	64.66	0.151	0.139	0.103
	64-100-0.0977MB	14.86	0.657	0.446	0.440	70.00	0.140	0.128	0.124
	70-100-0.0977MB	17.03	0.573	0.350	0.335	66.17	0.148	0.144	0.133
	80-100-0.0977MB	15.40	0.634	0.547	0.532	63.42	0.154	0.144	0.128
	90-100-0.0977MB	20.29	0.481	0.342	0.311	62.56	0.156	0.144	0.129
4	100-100-0.0977MB	29.09	0.336	0.331	0.312	74.14	0.132	0.123	0.111
	1-5-0.0010MB	0.01	0.626	0.209	0.209	0.21	0.023	0.005	0.005
	1-10-0.0010MB	0.01	0.793	0.160	0.160	0.22	0.045	0.005	0.005
	1-20-0.0010MB	0.01	1.762	0.160	0.156	0.28	0.069	0.005	0.005
	1-40-0.0010MB	0.01	3.152	0.163	0.158	0.43	0.090	0.009	0.006
	1-80-0.0010MB	0.01	6.137	0.166	0.127	0.59	0.133	0.005	0.004
	1-160-0.0010MB	0.01	11.456	0.154	0.138	0.89	0.176	0.010	0.006
	1-320-0.0010MB	0.02	19.808	0.152	0.110	0.91	0.342	0.004	0.001
	1-640-0.0010MB	0.02	39.981	0.141	0.061	0.96	0.653	0.004	0.001
4	1-1280-0.0010MB	0.01	88.724	0.249	0.072	1.06	1.181	0.003	0.001

图 9 minio 性能评测数据

#	Clients-Samples-Size	W_Throughput(MB/s)	W_Duration(s)	W_99th(s)	W_90th(s)	R_Throughput(MB/s)	R_Duration(s)	R_99th(s)	R_90th(s)
1	1-100-0.0010MB	0.34	0.290	0.042	0.005	0.91	0.107	0.011	0.001
	1-100-0.0020MB	1.76	0.111	0.002	0.001	1.89	0.103	0.004	0.001
	1-100-0.0039MB	3.42	0.114	0.003	0.001	3.75	0.104	0.005	0.001
	1-100-0.0098MB	8.20	0.119	0.003	0.001	9.33	0.105	0.003	0.001
	1-100-0.0195MB	13.91	0.140	0.018	0.002	18.66	0.105	0.005	0.001
	1-100-0.0391MB	26.90	0.145	0.005	0.002	36.35	0.107	0.003	0.001
	1-100-0.0977MB	56.36	0.173	0.004	0.002	85.83	0.114	0.002	0.001
	1-100-0.1953MB	75.40	0.259	0.005	0.003	161.47	0.121	0.002	0.001
	1-100-0.3906MB	77.86	0.502	0.063	0.005	291.13	0.134	0.002	0.002
	1-100-1.0000MB	82.52	1.212	0.157	0.022	549.01	0.182	0.003	0.002
2	1-100-0.0010MB	0.32	0.307	0.157	0.003	0.96	0.102	0.004	0.001
	2-100-0.0010MB	0.98	0.099	0.004	0.002	1.04	0.094	0.003	0.002
	4-100-0.0010MB	0.91	0.107	0.007	0.005	0.86	0.113	0.010	0.006
	8-100-0.0010MB	0.34	0.288	0.057	0.035	0.10	1.023	1.023	0.015
	16-100-0.0010MB	0.09	1.081	1.032	0.066	0.08	1.240	1.237	1.027
	32-100-0.0010MB	0.07	1.264	1.251	1.034	0.08	1.280	1.278	1.065
	64-100-0.0010MB	0.05	1.244	1.226	1.055	0.03	3.301	3.296	3.107
	70-100-0.0010MB	0.05	1.299	1.286	1.222	0.01	7.822	7.812	4.348
	80-100-0.0010MB	0.03	1.259	1.236	1.041	0.01	6.789	6.777	4.358
	90-100-0.0010MB	0.03	1.276	1.239	1.053	0.02	4.485	4.474	3.287
3	100-100-0.0010MB	0.03	1.251	1.236	1.050	0.01	19.355	19.346	10.891
	1-100-0.0977MB	53.32	0.183	0.009	0.002	85.03	0.115	0.005	0.001
	2-100-0.0977MB	75.59	0.129	0.005	0.003	94.49	0.103	0.004	0.003
	4-100-0.0977MB	41.01	0.238	0.062	0.021	42.60	0.229	0.045	0.020
	8-100-0.0977MB	9.66	1.011	1.011	0.016	9.51	1.027	1.025	0.043
	16-100-0.0977MB	8.32	1.174	1.060	0.069	8.52	1.146	1.047	0.076
	32-100-0.0977MB	7.69	1.270	1.245	1.060	7.64	1.279	1.274	1.233
	64-100-0.0977MB	2.68	3.347	3.264	2.121	5.12	1.907	1.884	1.508
	70-100-0.0977MB	4.51	2.166	2.124	1.258	2.98	3.278	3.244	2.681
	80-100-0.0977MB	3.11	3.140	3.087	2.231	1.23	7.970	7.962	2.775
4	90-100-0.0977MB	4.16	2.347	2.328	2.108	3.48	2.805	2.782	1.951
	100-100-0.0977MB	2.10	3.350	3.137	3.118	2.21	4.416	4.402	2.784
	1-5-0.0010MB	0.57	0.009	0.003	0.003	0.65	0.007	0.003	0.003
	1-10-0.0010MB	0.87	0.011	0.002	0.002	0.95	0.010	0.001	0.001
	1-20-0.0010MB	0.80	0.024	0.003	0.002	0.72	0.027	0.004	0.003
	1-40-0.0010MB	0.88	0.044	0.002	0.002	0.94	0.041	0.002	0.001
	1-80-0.0010MB	0.33	0.236	0.046	0.005	0.98	0.080	0.001	0.001
	1-160-0.0010MB	0.49	0.319	0.034	0.004	0.73	0.215	0.003	0.002
	1-320-0.0010MB	0.39	0.811	0.016	0.005	0.73	0.429	0.005	0.002
	1-640-0.0010MB	0.43	1.437	0.006	0.004	0.59	1.062	0.004	0.002
	1-1280-0.0010MB	0.73	1.708	0.004	0.002	0.98	1.274	0.002	0.001

图 10 mock-s3 性能评测数据

## 5. 分析结果

### (1) 对象大小对性能的影响

#### ①minio

如图 9 中编号 1 的测试用例。并发终端数为 1,对象数为 100,对象大小从 1KB 增长到 1MB。随着对象大小的增大,吞吐率也增大。将百分位延迟用散点图表示。从图中可以观察到读取延迟无较大变化,写入延迟随对象大小增大呈现先增大再减小再增大的变化。

根据上述分析,可知对象大小越大,吞吐率越大,延迟在对象较小和较大时比较高。

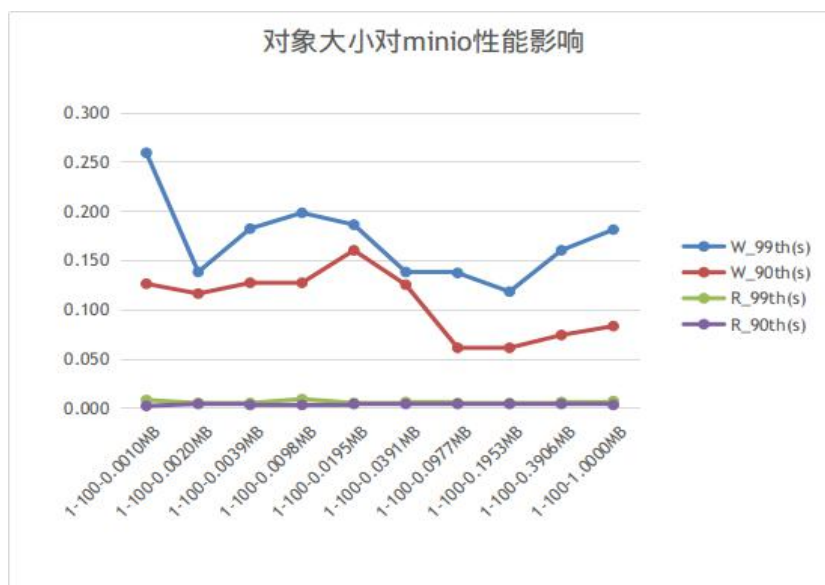


图 11 minio 改变对象大小

## ②mock-s3

图 10 中编号 1 的测试用例显示，随对象大小的增大，吞吐率在增大。作出百分位延迟散点图。观察图像可知，读取延迟无较大变化，写入延迟随对象大小增大先减小、后增大。

根据以上分析，随对象大小增大，mock-s3 吞吐率增大，写入延迟在对象较小和较大时较大，读取延迟无较大变化。对比 minio，mock-s3 的写入延迟和读取延迟更小，吞吐量更大，性能更好。

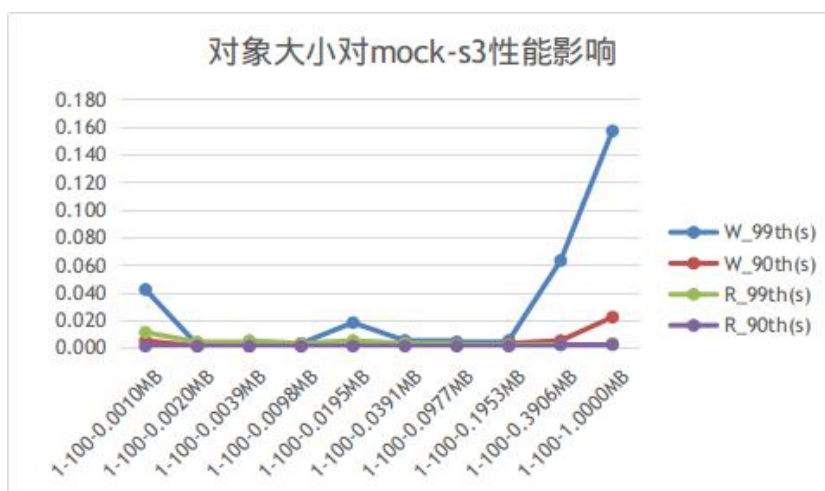


图 12 mock-s3 改变对象大小

## ③对于熟悉的某类应用，根据其数据访问特性，怎样适配对象存储最合适？



根据以上分析,在使用要求低延迟的应用时,例如搜索引擎,可以选择 mock-s3 且工作的对象大小应控制较小。在使用追求吞吐率和访问速度的应用时,例如网盘,选择 mock-s3 且工作在对象大小较大时。

## (2) 并发客户端数对性能的影响

### ①minio

如图 9 中编号 2 的测试用例。并发终端数从 1 增加到 100,对象数为 100,对象大小 1KB。编号 3 的测试用例,仅仅将对象大小改为 100KB,其他条件一致。可以观察到两种条件下,随着并发客户端增加,吞吐率在增加,且相比 1KB 大小的测试对象,100KB 大小的测试对象有更大的吞吐率。将延迟折线图画出。



图 13 并发客户端数量影响性能-1KB



图 14 并发客户端数量影响性能-100KB

可以观察到，随着并发客户端增加，延迟也呈现增加的趋势。总结下来，minio 在并发客户端增加的情况下，吞吐率在提高，但延迟也提高了，因此总体性能是在下降的。

#### ②mock-s3

在图 10 中，编号 2、3 的测试用例中标红的部分，写入发生错误，可以观察到错误均发生在 32 个并发客户端之后，说明在 32 个并发客户端后 mock-s3 的并发数量达到上限，继续增加会发生丢失数据的严重问题。

③综合上述分析，并发数量增加时，吞吐率增大，延迟增大，当达到并发极限时，可能出现丢失数据问题。minio 的并发负载能力比 mock-s3 更好。

#### （3）对象数量对性能影响

观察图 9 和图 10，在并发客户端数量为 1，对象数量从 5 增加到 1280，对象大小为 1KB 的条件下，minio 和 mock-s3 的吞吐率和延迟变化不大，因为此时只有一个客户端，所以数据是串行到达，对象数量只是影响处理时间，对吞吐率和延迟无作用。

#### （4）I/O 延迟背后的关键影响要素

根据以上三个测试，可知 I/O 延迟主要的影响因素：

①对象大小，对象大小越大，延迟越大。是因为对象越大需要从磁盘读取的数据越大，耗时越长。

②并发数，并发客户端越多，延迟越大。是因为过多的连接请求产生拥塞，需要排队处理请求，而超过负荷的请求会造成请求失败，数据丢失。

## 六、实验总结

本次实验是一次比较新颖和有趣的实验，通过实验接触了面向对象存储这一新兴的存储技术，虽然实验内容不算难，但是能够从中学习到较多的技能，由于之前已经学习过 Git、Python、Java，所以开始搭建环境的时候比较轻松，因为之前已经具备了对应的开发环境，由于想尝试新的技术，所以选择了 Go 语言实现的客户端 osm 和评测工具 S3 Bench，搭建 Go 环境时，对环境变量的设置有所疑问，上网查资料后知道了应该怎么设置环境变量。由于无法连接到外网，所以安装 Go 模块成了一个问题，老师教了一个从 Gitee 上查找镜像库 clone 到本地进行加载的办法，还有一个办法是更换 get 的源，这个方法还没有尝试过，因为安装较大的模块时，比如本次用到的依赖 aws-sdk-go，有 100 多 MB，采用离线到本地进行加载的方式更加合适。

在配置服务端、客户端和评测工具时，老师给的脚本节省了很多学习的时间，而且直接阅读脚本内容，也能进一步理解是怎么配置的，具体涉及到哪些参数，参数具体作用是什么。由于 S3 Bench 没有批量测试和输出结果文件功能，所以利用 shell 脚本实现了批量测试和终端输出的重定向。

实验过程中老师和同学对我的疑问和困难进行了解答，同时也参考了往届学长的实验，实验过程中虽有挫折，但是收获满满。



## 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.
- [4] URL: [https://gitee.com/shi\\_zhan/obs-tutorial](https://gitee.com/shi_zhan/obs-tutorial)
- [5] URL: <https://github.com/cs-course/iot-storage-experiment-assignment-2019>