



---

2016 级

《物联网数据存储与管理》课程  
实 验 报 告

姓 名 许鸿皓

学 号 U201614908

班 号 物联网 1601 班

日 期 2019.05.29

---

## 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 .....	2
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 .....	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

对象存储，也叫做基于对象的存储，是用来描述解决和处理离散单元的方法的通用术语，这些离散单元被称作为对象。

对象存储系统是综合了 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。可以在一个持久稳固且高度可用的系统中存储任意的对象，且独立于虚拟机实例之外。应用和用户可以在对象存储中使用简单的 API 访问数据；这些通常都基于表属性状态转移（REST）架构，但是也有面向编程语言的界面。

Minio Cloud Storage 是一款轻量级对象存储服务，兼容 Amazon S3。只需要简单的命令，就可以搭建服务器，能实现可以通过浏览器访问的简易网盘功能。利用 Minio 客户端 MC 可以在命令行中对云中的内容进行管理。

COSBench 是衡量云对象存储服务性能的基准测试工具。

S3 Bench 提供了在与 S3 兼容的端点上运行非常基本的吞吐量基准测试的能力。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。

S3 Benchmark 是由 Wasabi 提供的一种性能测试工具，用于执行对象的 S3 操作（Put、Get 和 Delete）。对于不同的测试，除了桶配置之外，对象大小和线程数也会有所不同。

## 三、实验环境

实验环境和工具如表 1 所示。

表 1 实验环境表

操作系统	Windows 10 家庭中文版
处理器	Inter(R) Core(TM) i7-7500 CPU@2.70GHz 2.90GHz

Java 版本	java version "1.8.0_131"
Python 版本	Python 3.7.3
GO 版本	go version go1.12.5 windows/amd64
服务器端	Minio
客户端	mc

## 四、实验内容

本次实验内容大致如下。

1. 熟悉 git 的使用方法，搭建自己的库进行上传文件。
2. 搭建 Java, python, go 的环境。
3. 对对象存储系统进行实践，搭建服务器端，并利用客户端进行操作。
4. 对对象存储系统进行测试，并进行一定的性能分析。

### 4.1 对象存储技术实践

1. 配置 minio server 端，建立本地服务器，并在可视化界面中进行简单的上传、删除文件等操作。
2. 下载 mc 客户端，可创建 bucket。
3. 配置 cosbench，对 minio 进行性能分析。
4. 配置 s3bench 和 s3benchmark，观察其运行结果。

### 4.2 对象存储性能分析

1. 测试读写性能的优劣。
2. 测试 work 值对各项指标的影响，包含吞吐率，带宽等。
3. 测试不同块大小对各项指标的影响。

## 五、实验过程

1. 配置 minio server 端。

下载服务器端 minio.exe 文件，新建一个文件夹 minioserver 作为本地的服务器，用来存储各客户端发送的文件。利用 E:\minio\minio.exe server E:\minio\minioserver 命令设置该文件夹，可以看到 AccessKey 为 INP1BWF9UASLA6ZW9E7W，SerectKey 为 EuKUeXSUvc6EXDFf+THWj +4aD+BQSZIlocNTcG++，公钥和私钥的值均可以在.minio.sys 文件夹下 config.json 文件中进行修改，本实验采用默认密

钥。操作结果如图 1 所示。

```
C:\Users\dell>E:\minio\minio.exe server E:\minio\minioserver

Endpoint: http://169.254.34.249:9000 http://169.254.138.248:9000 http://10.11.55.235:9000 http://169.254.235.88:9000
http://169.254.38.21:9000 http://192.168.1.2:9000 http://127.0.0.1:9000
AccessKey: INP1BWF9UASLA6ZW9E7W
SecretKey: EuKUeXSUvc6EXDFf+THWj+4aD+BQSZIlocNTcG++

Browser Access:
http://169.254.34.249:9000 http://169.254.138.248:9000 http://10.11.55.235:9000 http://169.254.235.88:9000 http://169.254.38.21:9000 http://192.168.1.2:9000 http://127.0.0.1:9000

Command-line Access: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc.exe config host add myminio http://169.254.34.249:9000 INP1BWF9UASLA6ZW9E7W EuKUeXSUvc6EXDFf+THWj+4aD+BQSZIlocNTcG++

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 打开 minio server

在浏览器中打开 `http://127.0.0.1:9000`，为服务器的可视化管理界面，输入 `AccessKey` 和 `SecretKey` 登陆。界面右下角红色标记可用来添加 bucket 和上传文件，创建名为 `test` 的 bucket，并上传一个 doc 文件，结果如图 2 所示。

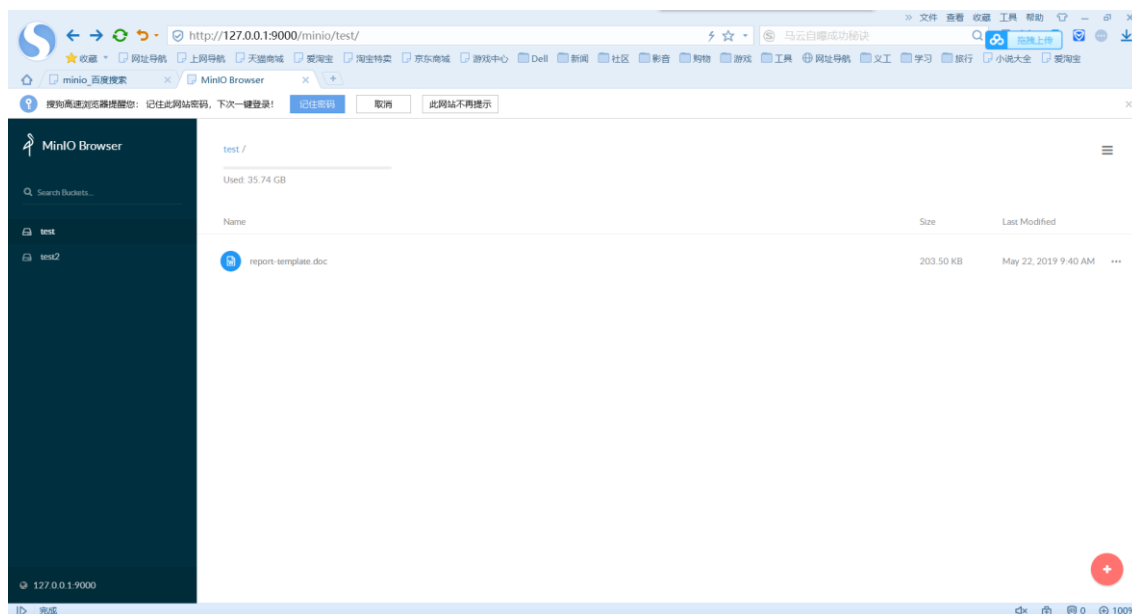


图 2 可视化服务器界面

此时打开 `minioserver`，可以看到文件被传到了此位置。如图 3 所示。

 .minio.sys	2019/5/22 9:40	文件夹
 test	2019/5/22 9:40	文件夹

图 3 minioserver 上传成功

## 2. 配置 minio client 端

下载客户端 `mc.exe`，通过命令行添加服务器 `minioserver`，之后可利用客户端

的命令行对服务器进行操作。如图 4 所示。其中第一次输入有误，所以完成初始化后出现报错情况。

```
C:\Users\dell>E:\minio\mc.exe config host add E:\minio\minioserver http://127.0.0.1:9000 INP1BWF9UASLA6ZW9E7W EuKUeXSUvc6EXDff+THWj+4aD+BQSZIllocNTcG++ S3v4
mc.exe: Configuration written to 'C:\Users\dell\mc\config.json'. Please update your access credentials.
mc.exe: Successfully created 'C:\Users\dell\mc\share'.
mc.exe: Initialized share uploads 'C:\Users\dell\mc\share\uploads.json' file.
mc.exe: Initialized share downloads 'C:\Users\dell\mc\share\downloads.json' file.
mc.exe: <ERROR> Invalid alias. Alias 'E:\minio\minioserver' should have alphanumeric characters such as [helloWorld0, hello_World0, ...].

C:\Users\dell>E:\minio\mc.exe config host add minioserver http://127.0.0.1:9000 INP1BWF9UASLA6ZW9E7W EuKUeXSUvc6EXDff+THWj+4aD+BQSZIllocNTcG++ S3v4
Added 'minioserver' successfully.
```

图 4 添加服务器

利用命令 `mc.exe mb minioserver/test2`，为服务器创建新的 bucket 名为 test2，之后通过 `ls` 命令查看服务器，结果如图 5 所示。

```
C:\Users\dell>E:\minio\mc.exe mb minioserver/test2
Bucket created successfully minioserver/test2.

C:\Users\dell>E:\minio\mc.exe ls minioserver
[2019-05-22 09:40:31 CST]      0B test\
[2019-05-22 09:58:20 CST]      0B test2\
```

图 5 mc 命令行操作

此时打开 minioserver，可以看到新创建的 test2，如图 6 所示。




 .minio.sys	2019/5/22 9:40	文件夹
 test	2019/5/22 9:40	文件夹
 test2	2019/5/22 9:58	文件夹

图 6 mc 操作成功

3. 配置 cosbench 进行测评。

在维持服务器打开状态下进行。

下载 cosbench 并解压，点击批处理文件 `start_all.bat` 执行，结果如图 7 所示。

```
COSBench Driver - start-driver.bat
E:\cosbench\0.4.2.c4>java -Dcosbench.tomcat.config=conf/driver-tomcat-server.xml -server -cp main/* org.eclipse.equinox.launcher.Main -configuration conf/.driver -console 18089
Listening on port 0.0.0.0/0.0.0.0:18089 ...
Persistence bundle starting...
Persistence bundle started.

!!! Service will listen on web port: 18088 !!!

COSBench Controller - start-controller.bat
E:\cosbench\0.4.2.c4>java -Dcosbench.tomcat.config=conf/controller-tomcat-server.xml -server -cp main/* org.eclipse.equinox.launcher.Main -configuration conf/.controller -console 19089
Listening on port 0.0.0.0/0.0.0.0:19089 ...
Persistence bundle starting...
Persistence bundle started.

!!! Service will listen on web port: 19088 !!!
```

图 7 运行 start-all

双击 web.bat 进入浏览器界面，修改 workload-example.xml 里的 accessKey 和 secretKey 并提交，点击 info 获得运行信息，运行结束后得到分析报告，如图 8 所示。

ID	Name	Works	Workers	Op-Info	State	Link
w1-s1-init	init	1 wks	1 wkrs	init	completed	<a href="#">view details</a>
w1-s2-prepare	prepare	8 wks	64 wkrs	prepare	completed	<a href="#">view details</a>
w1-s3-8kb	8kb	1 wks	8 wkrs	read, write	completed	<a href="#">view details</a>
w1-s4-16kb	16kb	1 wks	8 wkrs	read, write	completed	<a href="#">view details</a>
w1-s5-32kb	32kb	1 wks	4 wkrs	read, write	completed	<a href="#">view details</a>
w1-s6-64kb	64kb	1 wks	4 wkrs	read, write	completed	<a href="#">view details</a>
w1-s7-128kb	128kb	1 wks	1 wkrs	read, write	completed	<a href="#">view details</a>
w1-s8-256kb	256kb	1 wks	1 wkrs	read, write	completed	<a href="#">view details</a>
w1-s9-512kb	512kb	1 wks	1 wkrs	read, write	completed	<a href="#">view details</a>
w1-s10-1mb	1mb	1 wks	1 wkrs	read, write	completed	<a href="#">view details</a>
w1-s11-cleanup	cleanup	1 wks	1 wkrs	cleanup	completed	<a href="#">view details</a>
w1-s12-dispose	dispose	1 wks	1 wkrs	dispose	completed	<a href="#">view details</a>

图 8 workload-example 运行结果

#### 4. 读写性能比较

3 中运行的报告如图 9 所示。可以看出，每个任务的读操作与写操作相比，吞吐量 and 带宽都要高出很多，而处理时延则相对较短，说明其读性能要优于写性能，由于此样例均成功，所以无法看出读写操作成功率的不同。

op1: read	3.61 kops	28.91 MB	9.23 ms	9.09 ms	120.86 op/s	966.89 KB/S	100%
op2: write	907 ops	7.26 MB	226.39 ms	226.32 ms	30.33 op/s	242.68 KB/S	100%
op1: read	2.94 kops	47.12 MB	9.29 ms	9.12 ms	99.54 op/s	1.59 MB/S	100%
op2: write	768 ops	12.29 MB	272.32 ms	272.05 ms	25.96 op/s	415.38 KB/S	100%
op1: read	3.1 kops	99.1 MB	8.19 ms	7.97 ms	103.39 op/s	3.31 MB/S	100%
op2: write	757 ops	24.22 MB	124.37 ms	123.49 ms	25.27 op/s	808.67 KB/S	100%
op1: read	3.11 kops	199.3 MB	7.78 ms	7.16 ms	103.83 op/s	6.64 MB/S	100%
op2: write	782 ops	50.05 MB	122 ms	120.05 ms	26.07 op/s	1.67 MB/S	100%
op1: read	1.4 kops	179.46 MB	7.77 ms	6.37 ms	46.79 op/s	5.99 MB/S	100%
op2: write	382 ops	48.9 MB	49.58 ms	45.58 ms	12.75 op/s	1.63 MB/S	100%
op1: read	1.18 kops	301.31 MB	9.28 ms	6.62 ms	39.24 op/s	10.04 MB/S	100%
op2: write	301 ops	77.06 MB	63.02 ms	55.31 ms	10.03 op/s	2.57 MB/S	100%
op1: read	964 ops	493.57 MB	12.73 ms	7.29 ms	32.14 op/s	16.45 MB/S	100%
op2: write	234 ops	119.81 MB	75.32 ms	60.06 ms	7.8 op/s	3.99 MB/S	100%
op1: read	564 ops	564 MB	21.6 ms	7.49 ms	18.81 op/s	18.81 MB/S	100%
op2: write	115 ops	115 MB	154.5 ms	122.24 ms	3.83 op/s	3.83 MB/S	100%

图 9 cosbench 报告

#### 5. 测试 work 值对各项指标的影响

op1: read	1.28 kops	163.2 MB	9.67 ms	7.94 ms	42.56 op/s	5.45 MB/S	100%
op2: write	307 ops	39.3 MB	57.07 ms	53.16 ms	10.25 op/s	1.31 MB/S	100%
op1: read	2.14 kops	273.41 MB	9 ms	7.41 ms	71.26 op/s	9.12 MB/S	100%
op2: write	505 ops	64.64 MB	80.31 ms	76.32 ms	16.85 op/s	2.16 MB/S	100%
op1: read	2.89 kops	370.18 MB	9.31 ms	7.73 ms	96.67 op/s	12.37 MB/S	100%
op2: write	705 ops	90.24 MB	131.25 ms	127.66 ms	23.56 op/s	3.02 MB/S	100%
op1: read	3.08 kops	394.75 MB	10.13 ms	8.66 ms	103.66 op/s	13.27 MB/S	100%
op2: write	761 ops	97.41 MB	271.4 ms	267.78 ms	25.58 op/s	3.27 MB/S	100%
op1: read	3.11 kops	397.57 MB	12.84 ms	11.24 ms	104.64 op/s	13.39 MB/S	100%
op2: write	735 ops	94.08 MB	591.15 ms	587.53 ms	24.79 op/s	3.17 MB/S	100%
op1: read	3.48 kops	445.31 MB	23.1 ms	21.54 ms	121.36 op/s	15.53 MB/S	100%
op2: write	808 ops	103.42 MB	1038 ms	1034.39 ms	28.21 op/s	3.61 MB/S	100%
op1: read	3.72 kops	476.29 MB	16.24 ms	14.77 ms	141.52 op/s	18.11 MB/S	100%
op2: write	828 ops	105.98 MB	1972.86 ms	1969.66 ms	31.5 op/s	4.03 MB/S	100%
op1: read	4.25 kops	543.62 MB	37.11 ms	35.56 ms	165.42 op/s	21.17 MB/S	100%
op2: write	890 ops	113.92 MB	3619.2 ms	3616.02 ms	34.19 op/s	4.38 MB/S	100%

图 10 worker 数量的影响

将 workload-example 中的块大小保持 128K 不变，workers 的数量从 1 依次翻倍到 128，保存并重新提交，运行报告如图 10 所示。

从报告中可以看出随着 worker 的增加，其吞吐量，时延和带宽均呈现增加趋势，而增加速度之间减慢，说明 worker 数量尚未到服务器能力上限，性能随 worker 数量增加而提升，若到达饱和状态后，服务器出现拥挤，造成堵塞，就会出现性能降低的情况。

## 6. 测试块大小对各项指标的影响

将 workload-example 中的 workers 数量保持 4 不变，块大小从 8kb 依次翻倍到 1mb，保存并重新提交，运行报告如图 11 所示。

op1: read	2.96 kops	23.7 MB	7.81 ms	7.68 ms	98.91 op/s	791.3 KB/S	100%
op2: write	743 ops	5.94 MB	129.75 ms	129.71 ms	24.8 op/s	198.43 KB/S	100%
op1: read	2.83 kops	45.22 MB	8.27 ms	8.1 ms	94.41 op/s	1.51 MB/S	100%
op2: write	723 ops	11.57 MB	132.7 ms	132.32 ms	24.15 op/s	386.45 KB/S	100%
op1: read	2.92 kops	93.44 MB	7.68 ms	7.48 ms	97.64 op/s	3.12 MB/S	100%
op2: write	732 ops	23.42 MB	132.35 ms	131.3 ms	24.48 op/s	783.33 KB/S	100%
op1: read	2.99 kops	191.42 MB	8.42 ms	7.78 ms	99.92 op/s	6.39 MB/S	100%
op2: write	729 ops	46.66 MB	129.32 ms	127.12 ms	24.35 op/s	1.56 MB/S	100%
op1: read	2.67 kops	341.63 MB	9.99 ms	8.26 ms	89.23 op/s	11.42 MB/S	100%
op2: write	655 ops	83.84 MB	141.61 ms	137.88 ms	21.9 op/s	2.8 MB/S	100%
op1: read	2.22 kops	567.04 MB	10.68 ms	7.47 ms	74.48 op/s	19.07 MB/S	100%
op2: write	544 ops	139.26 MB	174.91 ms	167.47 ms	18.29 op/s	4.68 MB/S	100%
op1: read	1.87 kops	958.98 MB	13.97 ms	7.67 ms	62.5 op/s	32 MB/S	100%
op2: write	449 ops	229.89 MB	208.35 ms	194.16 ms	14.98 op/s	7.67 MB/S	100%
op1: read	1.13 kops	1.13 GB	21.59 ms	9.09 ms	37.96 op/s	37.96 MB/S	100%
op2: write	298 ops	298 MB	317.79 ms	285.89 ms	10 op/s	10 MB/S	100%

图 11 块大小的影响

从报告中可以看出随着块增大，其读写的处理时延成增大趋势，且增加速度越来越快，说明一开始块较小，处理时延增加不大，而之后超出其处理能力后时延会迅速增加。带宽一直成增加趋势，而吞吐量则先稳定在一个值左右，随后当超出处理能力后快速下降，吞吐量与时延的变化相对应，说明当块较小时，服务器处理能力和带宽相对充足，因此吞吐率基本不变，时延增加不多，而带宽值几乎也是翻倍地增加，但当块增加到一定程度，服务器能力开始饱和甚至不足，吞吐率严重下降。

## 7. S3bench 和 S3-benchmark 测试

下载 S3bench，输入命令将 numClients 设为 10，numSamples 设为 100，objectSize 设为 1024，在 2 中创建的 test2 中进行测试，命令如图 12 所示，结果如图 13 所示。



```

C:\Users\dell\go\bin>C:\Users\dell\go\bin\s3bench -accessKey=INP1BWF9UASLA6ZW9E7W -accessSecret=EuKUeXSUvc6EXDFf+THWj+4a
D+BQSZIlocNTcG++ -endpoint=http://127.0.0.1:9000 -bucket=test2 -objectNamePrefix=test2 -numClients=10 -numSamples=100 -o
bjectSize=1024
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:          test2
objectNamePrefix: test2
objectSize:      0.0010 MB
numClients:      10
numSamples:      100
verbose:         %!d(bool=false)

Generating in-memory sample data... Done (5.9779ms)

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:          test2
objectNamePrefix: test2
objectSize:      0.0010 MB
numClients:      10
numSamples:      100
verbose:         %!d(bool=false)

```

图 12 S3bench 测试命令

```

Generating in-memory sample data... Done (5.9779ms)

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:          test2
objectNamePrefix: test2
objectSize:      0.0010 MB
numClients:      10
numSamples:      100
verbose:         %!d(bool=false)

Results Summary for Write Operation(s)
Total Transferred: 0.098 MB
Total Throughput:  0.04 MB/s
Total Duration:   2.704 s
Number of Errors: 0
-----
Write times Max:    0.766 s
Write times 99th %ile: 0.766 s
Write times 90th %ile: 0.482 s
Write times 75th %ile: 0.284 s
Write times 50th %ile: 0.225 s
Write times 25th %ile: 0.175 s
Write times Min:    0.125 s

Results Summary for Read Operation(s)
Total Transferred: 0.098 MB
Total Throughput:  0.28 MB/s
Total Duration:   0.354 s
Number of Errors: 0
-----
Read times Max:     0.190 s
Read times 99th %ile: 0.190 s
Read times 90th %ile: 0.175 s
Read times 75th %ile: 0.027 s
Read times 50th %ile: 0.018 s
Read times 25th %ile: 0.012 s
Read times Min:     0.004 s

Cleaning up 100 objects...
Deleting a batch of 100 objects in range {0, 99}... Succeeded
Successfully deleted 100/100 objects in 427.8547ms

```

图 13 S3bench 测试结果

下载 S3-benchmark，输入命令进行测试，结果如图 14 所示。

```
C:\Users\dell\go\bin>C:\Users\dell\go\bin\s3-benchmark -a INP1BWF9UASLA6ZW9E7W -s EuKUeXSUvc6EXDFf+THWj+4aD+BQSZ1locNTcG
++ -u http://127.0.0.1:9000 -b test3 -d 3 -t 1 -z 1K
Wasabi benchmark program v2.0
Parameters: url=http://127.0.0.1:9000, bucket=test3, region=us-east-1, duration=3, threads=1, loops=1, size=1K
Loop 1: PUT time 3.0 secs, objects = 55, speed = 18.2KB/sec, 18.2 operations/sec. Slowdowns = 0
Loop 1: GET time 0.3 secs, objects = 55, speed = 162.2KB/sec, 162.2 operations/sec. Slowdowns = 0
Loop 1: DELETE time 0.4 secs, 144.4 deletes/sec. Slowdowns = 0
result title: name-concurrency-size, uploadspeed, downloadspeed
result csv: 127-1-1K,0.02,0.16
```

图 14 S3-benchmark 测试结果

## 六、实验总结

在本次实验中，主要是搭建了一个简单的分布式对象存储系统 minio，并通过工具对其性能进行了测试分析。Minio 感觉使用起来非常的方便，只要服务器运行着，就可以随时向其中添加文件，由此实现文件的汇总，操作也很简单。而通过测试工具，我也更加直观的了解到时延，吞吐量，带宽这些指标的意义，修改不同的数据，对这些指标的影响也是很明显的。

而这个实验较为简单，过程中遇到最多的问题还是概念和工具的不熟悉，尤其是命令行操作时，如果不清楚某个参数的含义就容易出现错误。比如使用客户端 mc 时，服务器名 minioserver 我一开始意味要输入绝对路径，然后就爆出了如图 4 所示的错误，后来发现只要输入服务器的名称。而使用 s3bench 时也遇到同样的问题，我一开始以为参数中的 bucket 是随便填的数值，命令会自己创建，然后运行中就报错了，发现其实需要自己想创建一个文件夹，再进行调用。当然相对的，这些错误解决起来也比较容易。所以本次实验的过程还是相对愉快的。

而通过自己动手从头开始搭建了一次对象存储系统，我对其工作原理和概念有了更加深刻和具体的了解，而不再是仅仅局限于课堂上的知识了。总体来说收获还是很大的。

## 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 Sy

---

mposium on Operating Systems Design and Implementation. Berkeley,  
CA, USA: USENIX Association, 2006: 307 – 320.