



2015 级

《物联网数据存储与管理》课程

# 实 验 报 告

姓 名 许万全

学 号 U201514882

班 号 物联网 1501 班

日 期 2018. 04. 23

# 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
五、实验过程.....	3
六、实验总结.....	14
参考文献.....	16

## 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

**Minio:**

一个基于 Apache License v2.0 开源协议的对象存储服务。它兼容亚马逊 S3 云存储服务接口，非常适合于存储大容量非结构化的数据，例如图片、视频、日志文件、备份数据和容器/虚拟机镜像等，而一个对象文件可以是任意大小，从几 kb 到最大 5T 不等。Minio 是一个非常轻量的服务,可以很简单的和其他应用的结合，类似 NodeJS, Redis 或者 MySQL。

**Minio Client:**

mc 提供包括 ls、cat、cp、mirror、diff 等 UNIX 命令。它提供文件系统以及亚马逊 S3 兼容性云存储服务。

**COSBench:**

有效测试各种服务端的表现（effectively measure the performance of these services）。

## 三、实验环境

操作系统版本为 Ubuntu14.04.5，如图 3.1:

```
xwq@xwq-Aspire-4749:~$ cat /etc/issue
Ubuntu 14.04.5 LTS \n \l
```

图 3.1 Ubuntu 版本

内核版本号为 4.4.0-31，如图 3.2:

```
xwq@xwq-Aspire-4749:~$ uname -a
Linux xwq-Aspire-4749 4.4.0-31-generic
```

图 3.2 内核版本号

python 版本为 2.7.6，如图 3. 3:

```
xwq@xwq-Aspire-4749:~$ python  
Python 2.7.6 (default, Nov 23 2017, 15:49:48)
```

图 3. 3 python 版本

Java 版本为 1.8，如图 3. 4

```
xwq@xwq-Aspire-4749:~$ java -version  
java version "1.8.0_161"
```

图 3. 4 Java 版本

## 四、实验内容

### 4.1 对象存储技术实践

- 1.配置 minio 服务端。
- 2.使用 mc 作为客户端。
- 3.配置 s3proxy 服务端。
- 4.使用 s3cmd 作为客户端。
- 5.安装使用 COSBench。

### 4.2 对象存储性能分析

- 1.测试读写性能与哪些参数有关。
- 2.测试当 workers 增大时，读写性能的变化程度。
- 3.测试当 workers 增大时，服务端的处理延迟。
- 4.模拟“百度搜索”“上传文件到云端”两个应用，查看 COSBench 的各项指标。
- 5.比较 minio 服务端和 s3proxy 服务端的优劣。

## 五、实验过程

1. 在 minio 所在目录下运行 `sudo ./minio server /data` 打开 minio 客户端，打开 `http://127.0.0.1:9000/minio/`，得到如图 5. 1 结果。

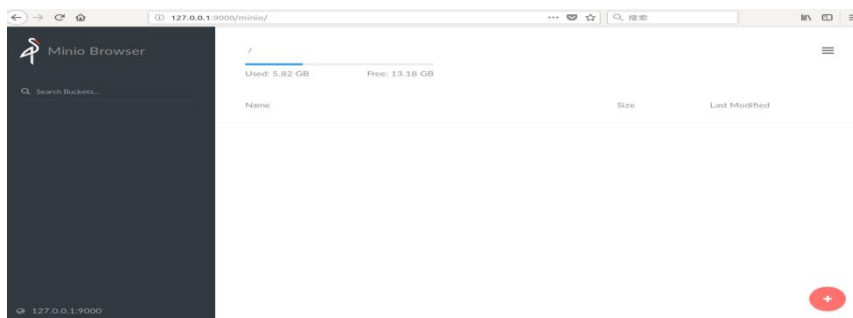


图 5. 1 打开 minio 服务器

2. 打开新终端，使用 minio 服务端提供的 mc 客户端命令行，如图 5. 2，打开 mc 客户端，如果成功打开客户端，结果会如图 5. 3 所示。

```
Command-line Access: https://docs.minio.io/docs/minio-client-quickstart-guide
$ mc config host add myminio http://10.10.95.186:9000 30ZBVBjH2ZD20UG9GLB9 iQ
hUxUlIh13DefVcTmfBCPzYZi1JUUs0arVrjIMm
```

图 5. 2 命令行打开 mc 客户端

```
Added `myminio` successfully.
```

图 5. 3 成功打开 mc

3. 切换到 COSBench 所在目录，使用 `start-all.sh` 启动 driver 和 controller。如图 5. 4，图 5. 5 所示。


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

图 5. 4 打开 driver

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

图 5. 5 打开 controller

4. 上传示例负载，使用 `sudo sh cli.sh submit conf/workload-config.xml`，打开 `http://127.0.0.1:19088/controller/index.html`，如图 5. 6，观察到负载成功上传。

Active Workloads 


	ID	Name	Submitted-At	State	Order	Link
<input type="checkbox"/>	w2	demo	Apr 16, 2018 8:54:59 AM	processing		<a href="#">view details</a>

图 5. 6 上传范例负载

由第 4 点可以得到结论，minio、mc、COSBench 成功安装并运行。

5. 上传实验所用负载，使用 `sudo sh cli.sh submit conf/workload-example.xml`

实验结果如图 5. 7

ID	Name	Works	Workers	Op-Info	State	Link
w4-s1-init	init	1 wks	1 wkrs	init	completed	<a href="#">view details</a>
w4-s2-prepare	prepare	8 wks	64 wkrs	prepare	completed	<a href="#">view details</a>
w4-s3-8kb	8kb	1 wks	8 wkrs	read, write	completed	<a href="#">view details</a>
w4-s4-16kb	16kb	1 wks	8 wkrs	read, write	completed	<a href="#">view details</a>
w4-s5-32kb	32kb	1 wks	4 wkrs	read, write	completed	<a href="#">view details</a>
w4-s6-64kb	64kb	1 wks	4 wkrs	read, write	completed	<a href="#">view details</a>
w4-s7-128kb	128kb	1 wks	1 wkrs	read, write	completed	<a href="#">view details</a>
w4-s8-256kb	256kb	1 wks	1 wkrs	read, write	completed	<a href="#">view details</a>
w4-s9-512kb	512kb	1 wks	1 wkrs	read, write	completed	<a href="#">view details</a>
w4-s10-1mb	1mb	1 wks	1 wkrs	read, write	completed	<a href="#">view details</a>
w4-s11-cleanup	cleanup	1 wks	1 wkrs	cleanup	completed	<a href="#">view details</a>
w4-s12-dispose	dispose	1 wks	1 wkrs	dispose	completed	<a href="#">view details</a>

图 5. 7 上传实验负载

具体的数值如图 5. 8:

op1: read	18.57 kops	18.57 MB	10.32 ms	10.32 ms	619.3 op/s	619.3 KB/S	100%
op2: write	4.61 kops	36.9 MB	10.32 ms	10.32 ms	153.77 op/s	1.23 MB/S	100%
op1: read	18.69 kops	18.69 MB	10.29 ms	10.29 ms	623.18 op/s	623.18 KB/S	100%
op2: write	4.56 kops	72.99 MB	10.29 ms	10.29 ms	152.09 op/s	2.43 MB/S	100%
op1: read	9.27 kops	9.27 MB	10.3 ms	10.3 ms	309.17 op/s	309.17 KB/S	100%
op2: write	2.34 kops	74.88 MB	10.3 ms	10.3 ms	78.01 op/s	2.5 MB/S	100%
op1: read	9.29 kops	9.29 MB	10.32 ms	10.32 ms	309.54 op/s	309.54 KB/S	100%
op2: write	2.3 kops	147.39 MB	10.34 ms	10.33 ms	76.77 op/s	4.91 MB/S	100%
op1: read	2.32 kops	2.32 MB	10.41 ms	10.41 ms	77.17 op/s	77.17 KB/S	100%
op2: write	551 ops	70.53 MB	10.43 ms	10.42 ms	18.37 op/s	2.35 MB/S	100%
op1: read	2.3 kops	2.3 MB	10.47 ms	10.47 ms	76.64 op/s	76.64 KB/S	100%
op2: write	550 ops	140.8 MB	10.47 ms	10.45 ms	18.34 op/s	4.69 MB/S	100%
op1: read	2.28 kops	2.28 MB	10.43 ms	10.43 ms	75.92 op/s	75.92 KB/S	100%
op2: write	573 ops	293.38 MB	10.56 ms	10.5 ms	19.11 op/s	9.78 MB/S	100%
op1: read	2.26 kops	2.26 MB	10.49 ms	10.48 ms	75.4 op/s	75.4 KB/S	100%
op2: write	577 ops	577 MB	10.56 ms	10.45 ms	19.23 op/s	19.23 MB/S	100%

图 5. 8 实验负载具体结果

1) 分析读性能:

比较 s3 (size=8kb、workers=8) 和 s4 (size=16kb、workers=8) 的读性能:

s3 的吞吐率为 619.3 op/s, 带宽为 619.3 KB/S

s4 的吞吐率为 623.18 op/s, 带宽为 623.18 KB/S

两者的差距非常小, 可以近似看做相同。

可以得到结果: 读性能与块大小无关。

我们得出以上结果后, 分析读性能时就可以不用再考虑 size 大小, 排除了 size 大小这个可能影响因素, 只剩下 workers 这个变量。

我们再比较 s3、s4、s5、s6。忽略掉 size 大小, s3 和 s4 的 workers 都是 8, s5、s6 的 workers 都是 4。

s5 的吞吐率为 309.17 op/s, 带宽为 309.17 KB/S

s6 的吞吐率为 309.54 op/s, 带宽为 309.54 KB/S

我们得到“读性能  $s3=s4>s5=s6$ ”的结果。

同时, 我们可以发现, s3 的读速率 619 差不多是 s5 的读速率 309 的两倍, 刚好 s3 的 workers 数是 s5 的 workers 数的两倍。

我们做出假设, workers 以一定倍数增大, 读性能也以相同倍数提高。

为了验证该假设, 我们分析 s6, workers=1, s6 的带宽为 77.17 KB/S。

即 s6 的 workers 数是 s5 的四分之一, 带宽也是 s5 的四分之一。

进一步验证, 我们将 s3 的 workers 改为 1024, 得到如图 5.9 结果:

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
read	196.25 kops	196.25 MB	125.11 ms	125.11 ms	6569.5 op/s	6.57 MB/S	100%

图 5.9 worker 改为 1024

发现 s3 的 workers 数虽然提高了 128 倍（与之前的 workers=8 的 s3 比较），但是读性能只提高了  $6.57 \times 1024 / 619 = 10.8$  倍。

所以我们的假设还存在问题。

我们将 8 个 workstage 的 size 保持不变，workers 依次改为 128、64、32、16、8、4、2、1。

得到如图 5.10 结果

op1: read	216.82 kops	216.82 MB	14.13 ms	14.13 ms	7230.34 op/s	→ 7.23 MB/S	100%
op2: write	54.29 kops	434.34 MB	14.14 ms	14.13 ms	1810.53 op/s	14.48 MB/S	100%
op1: read	146.12 kops	146.12 MB	10.49 ms	10.49 ms	4872.31 op/s	→ 4.87 MB/S	100%
op2: write	36.33 kops	581.33 MB	10.49 ms	10.49 ms	1211.5 op/s	19.38 MB/S	100%
op1: read	73.95 kops	73.95 MB	10.4 ms	10.4 ms	2466.37 op/s	→ 2.47 MB/S	100%
op2: write	18 kops	576.13 MB	10.4 ms	10.4 ms	600.43 op/s	19.21 MB/S	100%
op1: read	36.47 kops	36.47 MB	10.44 ms	10.44 ms	1215.88 op/s	→ 1.22 MB/S	100%
op2: write	9.32 kops	596.42 MB	10.44 ms	10.44 ms	310.71 op/s	19.89 MB/S	100%
op1: read	18.3 kops	18.3 MB	10.46 ms	10.46 ms	609.94 op/s	→ 609.94 KB/S	100%
op2: write	4.55 kops	582.14 MB	10.47 ms	10.45 ms	151.63 op/s	19.41 MB/S	100%
op1: read	9.09 kops	9.09 MB	10.47 ms	10.47 ms	302.91 op/s	→ 302.91 KB/S	100%
op2: write	2.33 kops	596.22 MB	10.46 ms	10.43 ms	77.64 op/s	19.87 MB/S	100%
op1: read	4.61 kops	4.61 MB	10.45 ms	10.45 ms	153.76 op/s	→ 153.76 KB/S	100%
op2: write	1.1 kops	563.2 MB	10.48 ms	10.43 ms	36.67 op/s	18.78 MB/S	100%
op1: read	2.32 kops	2.32 MB	10.42 ms	10.42 ms	77.34 op/s	→ 77.34 KB/S	100%
op2: write	540 ops	540 MB	10.53 ms	10.43 ms	18 op/s	18 MB/S	100%

图 5.10 workers 依次改为 128、64、32、16、8、4、2、1

我们发现，在 s4 到 s10 这 7 个 workstage 之间，读速率是以 2 倍的关系递减的。而 s3（workers=128）的读速率（7.23MB/S）却不是 s4（4.87MB/S）的两倍。

我们再联系之前 workers=1024、读速率=6.57MB 的情况，可以得出如下结论：

在 workers 数比较小时（小于 100），服务端不拥挤，没有阻塞情况出现，workers（模拟客户端的数量）翻倍，读的性能也成倍增加。但是当 workers 过多，服务器出现拥挤，造成堵塞，此时读性能可能不增反减。比如当 workers 为 1024 的 workstage 比 workers 为 128 的 workstage 读性能更加差。

我们联系实际情况：

读操作就相当于用户从服务端下载数据，下载速率与下载的东西大小无关，只与客户端的连接数有关。



当用户的请求数非常少时，服务端不会出现延迟、排队等情况，用户的请求都能够及时相应并完成。

当用户的请求数非常大时，连接数过多，服务端来不及相应，会出现延迟的情况，服务端出现拥塞，在处理连接等方面花费更大的开销，就会出现读速率增长不多甚至负增长的情况出现。

## 2) 分析写性能:

使用原样例(s3到s10的size大小从8kb到1mb,workers为8,8,4,4,1,1,1,1),测试结果如图，每个workstage的写速率在图 5. 11 中标出。

op1: read	18.57 kops	18.57 MB	10.32 ms	3	10.32 ms	619.3 op/s	619.3 KB/S	100%
op2: write	4.61 kops	36.9 MB	10.32 ms		10.32 ms	153.77 op/s	1.23 MB/S	100%
op1: read	18.69 kops	18.69 MB	10.29 ms	4	10.29 ms	623.18 op/s	623.18 KB/S	100%
op2: write	4.56 kops	72.99 MB	10.29 ms		10.29 ms	152.09 op/s	2.43 MB/S	100%
op1: read	9.27 kops	9.27 MB	10.3 ms	5	10.3 ms	309.17 op/s	309.17 KB/S	100%
op2: write	2.34 kops	74.88 MB	10.3 ms		10.3 ms	78.01 op/s	2.5 MB/S	100%
op1: read	9.29 kops	9.29 MB	10.32 ms	6	10.32 ms	309.54 op/s	309.54 KB/S	100%
op2: write	2.3 kops	147.39 MB	10.34 ms		10.33 ms	76.77 op/s	4.91 MB/S	100%
op1: read	2.32 kops	2.32 MB	10.41 ms	7	10.41 ms	77.17 op/s	77.17 KB/S	100%
op2: write	551 ops	70.53 MB	10.43 ms		10.42 ms	18.37 op/s	2.35 MB/S	100%
op1: read	2.3 kops	2.3 MB	10.47 ms	8	10.47 ms	76.64 op/s	76.64 KB/S	100%
op2: write	550 ops	140.8 MB	10.47 ms		10.45 ms	18.34 op/s	4.69 MB/S	100%
op1: read	2.28 kops	2.28 MB	10.43 ms	9	10.43 ms	75.92 op/s	75.92 KB/S	100%
op2: write	573 ops	293.38 MB	10.56 ms		10.5 ms	19.11 op/s	9.78 MB/S	100%
op1: read	2.26 kops	2.26 MB	10.49 ms	10	10.48 ms	75.4 op/s	75.4 KB/S	100%
op2: write	577 ops	577 MB	10.56 ms		10.45 ms	19.23 op/s	19.23 MB/S	100%

图 5. 11 原实验负载

分析 s3、s4，两者的 workers 相同，size 翻倍，写速率翻倍。

分析 s4，s5，s4 的块大小是 s5 的一半，workers 数是 s5 的两倍。但是两者的读速率持平（2.43MB/S 和 2.5MB/S）。按照 s3，s4 的分析结果，size 翻倍，写速率也应该翻倍。由此得出结论：写速率不仅与 size 大小有关，还与 workers 有关。

我们再分析 s6，s7。s7 的块大小是 s6 的两倍，workers 是 s6 的 1/4，写速率是 s6 的一半。即块大小翻倍，workers 变为 1/4，写速率减半。

所以我们可以得出结论，写速率与块大小、workers 都有关，且都是正比关系。

当 workers 不变，块大小翻倍，写速率也翻倍。当块大小 size 不变，workers 翻倍，写速率也翻倍。

我们再来分析一下写性能会不会也遇到与读性能一样的瓶颈。

我们用一个新的 workload，s3 到 s10 的大小从 8kb 到 1mb，workers 从 1 成倍递增到 128。结果如图 5. 12。

op1: read	2.29 kops	2.29 MB	10.49 ms	3	10.49 ms	76.28 op/s	76.28 KB/S	100%
op2: write	553 ops	4.42 MB	10.51 ms		10.51 ms	18.44 op/s	147.5 KB/S	100%
op1: read	4.51 kops	4.51 MB	10.52 ms	4	10.52 ms	150.29 op/s	150.29 KB/S	100%
op2: write	1.16 kops	18.62 MB	10.52 ms		10.51 ms	38.81 op/s	620.89 KB/S	100%
op1: read	9 kops	9 MB	10.61 ms	5	10.61 ms	300.09 op/s	300.09 KB/S	100%
op2: write	2.28 kops	72.86 MB	10.5 ms		10.5 ms	75.91 op/s	2.43 MB/S	100%
op1: read	18.2 kops	18.2 MB	10.52 ms	6	10.52 ms	606.81 op/s	606.81 KB/S	100%
op2: write	4.52 kops	289.09 MB	10.49 ms		10.48 ms	150.59 op/s	9.64 MB/S	100%
op1: read	36.47 kops	36.47 MB	10.49 ms	7	10.48 ms	1215.85 op/s	1.22 MB/S	100%
op2: write	9.09 kops	1.16 GB	10.51 ms		10.49 ms	303.07 op/s	38.79 MB/S	100%
op1: read	73.38 kops	73.38 MB	10.41 ms	8	10.41 ms	2446.68 op/s	2.45 MB/S	100%
op2: write	18.52 kops	4.74 GB	10.42 ms		10.4 ms	617.52 op/s	158.09 MB/S	100%
op1: read	145.92 kops	145.92 MB	10.47 ms	9	10.47 ms	4864.92 op/s	4.86 MB/S	100%
op2: write	36.73 kops	18.8 GB	10.52 ms		10.47 ms	1224.43 op/s	626.91 MB/S	100%
op1: read	205.29 kops	205.29 MB	14.92 ms	10	14.92 ms	6844.96 op/s	6.84 MB/S	100%
op2: write	51.57 kops	51.57 GB	14.94 ms		14.86 ms	1719.38 op/s	1.72 GB/S	100%

图 5. 12 workers 从 1 成倍递增到 128

我们知道，写速率与 workers、size 都是成正比，所以当块大小和 workers 都翻倍时，写速率应当翻 3 倍（变为原来的 4 倍）。如图，结果与我们的推测是一致的。

## 6. 架设实际应用

1) 模拟百度搜索（用户将关键词提交给服务器，服务器返回大量的搜索结果）。

修改 workload-config.xml，将读（下载）的比率修改为 95%，将写（上传）的比例修改为 5%。如图 5. 13 所示

```
<workstage name="main">
  <work name="main" workers="8" runtime="300">
    <operation type="read" ratio="95" config="containers=u(1,32);objects=u(1,50)" />
    <operation type="write" ratio="5" config="containers=u(1,32);objects=u(51,100);sizes=c(64)KB" />
  </work>
</workstage>
```

图 5. 13 将读的比率大幅提高

结果如图 5. 14

op1: read	218.41 kops	13.98 GB	10.41 ms	10.41 ms	728.04 op/s	46.59 MB/S	100%
op2: write	11.58 kops	740.93 MB	10.43 ms	10.42 ms	38.59 op/s	2.47 MB/S	100%

图 5. 14 模拟“百度搜索”结果

发现 op1 的各项指标（操作数、吞吐量、带宽）远大于 op2。

op1 是读操作，相当于服务器给用户的数据。

op2 是写操作，相当于用户上传给服务器的数据。

由图可以得到，服务端的带宽（46.59）远大于用户端的带宽（2.47），这与我们的常识相符。

用户只是将一个很小的关键词上传给百度，百度计算得到大量的搜索结果返回给用户，所以服务端给用户的数据就会远远大于用户上传的数据。

## 2) 上传文件到百度云

用户上传数据给云服务器（写操作），服务器将一些传输控制信息返回（读操作），修改 workload-config.xml，将读（下载）的比率修改为 5%，将写（上传）的比例修改为 95%。如图 5. 15 所示

```
<workstage name="main">
  <work name="main" workers="8" runtime="300">
    <operation type="read" ratio="5" config="containers=u(1,32);objects=u(1,50)" />
    <operation type="write" ratio="95" config="containers=u(1,32);objects=u(51,100);sizes=c(64)KB" />
  </work>
</workstage>
```

图 5. 15 将写的比率大幅提高

结果如图 5. 16 所示

op1: read	11.4 kops	729.92 MB	10.48 ms	10.48 ms	38.02 op/s	2.43 MB/S	100%
op2: write	215.16 kops	13.77 GB	10.46 ms	10.45 ms	717.2 op/s	45.9 MB/S	100%

图 5. 16 模拟“上传文件到百度云”结果

可以发现，op2 的各项参数远远大于 op1，因为当用户进行上传操作时，用户给服务器的数据要远远大于服务器返回的数据。

## 6. 搭建 S3proxy+S3cmd

使用 Docker 运行 S3proxy。

使用 `docker pull andrewgaul/s3proxy` 获取镜像。

再使用 `docker run --publish 80:80 --env S3PROXY_AUTHORIZATION=none andrewgaul/s3proxy` 运行镜像。

使用 `curl --request PUT http://127.0.0.1:80/testbucket` 向服务端添加一个 bucket，打开 127.0.0.1:80，结果如图 5. 17

```
-<ListAllMyBucketsResult>
  -<Owner>
    -<ID>
      75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a
    </ID>
    <DisplayName>CustomersName@amazon.com</DisplayName>
  </Owner>
  -<Buckets>
    -<Bucket>
      <Name>testbucket</Name>
      <CreationDate>2018-04-16T08:11:14.000Z</CreationDate>
    </Bucket>
  </Buckets>
</ListAllMyBucketsResult>
```

图 5. 17 s3proxy 服务端

使用 pip 下载 s3cmd。

使用命令 `s3cmd --configure` 进行配置。

配置结果如下图 5. 18 所示

```
New settings:
Access Key: hust
Secret Key: hust2018
Default Region: US
S3 Endpoint: 127.0.0.1:80
DNS-style bucket+hostname:port template for accessing a bucket: 80
Encryption password:
Path to GPG program: /usr/bin/gpg
Use HTTPS protocol: False
HTTP Proxy server name:
HTTP Proxy server port: 0
```

图 5. 18 s3cmd 配置

使用 `s3cmd mb s3://xwq_bucket_test1` 向服务端添加名为 `xwq_bucket_test1` 的桶，打开 `127.0.0.1:80`，结果如图 5. 19 所示

```

-<Bucket>
  <Name>xwq_bucket_test1</Name>
  <CreationDate>2018-04-16T09:16:16.000Z</CreationDate>
</Bucket>

```

图 5. 19 使用客户端向服务端发送请求

说明客户端和服务端连接成功。

将 `workload-example.xml` 上传，结果如图 5. 20，发现和 minio 的基本持平，说明在网络环境比较良好(workers 比较少时)的情况下，两者的表现是差不多的。

op1: read	18.2 kops	18.2 MB	10.55 ms	10.55 ms	606.76 op/s	606.76 KB/S	100%
op2: write	4.45 kops	35.6 MB	10.52 ms	10.52 ms	148.37 op/s	1.19 MB/S	100%
op1: read	18.07 kops	18.07 MB	10.53 ms	10.53 ms	602.57 op/s	602.57 KB/S	100%
op2: write	4.61 kops	73.79 MB	10.5 ms	10.5 ms	153.76 op/s	2.46 MB/S	100%
op1: read	9.07 kops	9.07 MB	10.5 ms	10.49 ms	302.48 op/s	302.48 KB/S	100%
op2: write	2.31 kops	74.02 MB	10.45 ms	10.45 ms	77.11 op/s	2.47 MB/S	100%
op1: read	9.12 kops	9.12 MB	10.49 ms	10.49 ms	304.02 op/s	304.02 KB/S	100%
op2: write	2.27 kops	145.09 MB	10.48 ms	10.47 ms	75.59 op/s	4.84 MB/S	100%
op1: read	2.26 kops	2.26 MB	10.45 ms	10.45 ms	75.25 op/s	75.25 KB/S	100%
op2: write	597 ops	76.42 MB	10.49 ms	10.47 ms	19.9 op/s	2.55 MB/S	100%
op1: read	2.26 kops	2.26 MB	10.49 ms	10.49 ms	75.43 op/s	75.43 KB/S	100%
op2: write	579 ops	148.22 MB	10.51 ms	10.48 ms	19.31 op/s	4.94 MB/S	100%
op1: read	2.32 kops	2.32 MB	10.49 ms	10.49 ms	77.18 op/s	77.18 KB/S	100%
op2: write	527 ops	269.82 MB	10.51 ms	10.46 ms	17.57 op/s	9 MB/S	100%
op1: read	2.3 kops	2.3 MB	10.44 ms	10.44 ms	76.57 op/s	76.57 KB/S	100%
op2: write	555 ops	555 MB	10.55 ms	10.45 ms	18.5 op/s	18.5 MB/S	100%

图 5. 20 上传 workers 较小的负载

进行压力测试，负载如图 5. 21 所示

w14-s3-8kb	8kb	1 wks	32 wkrs
w14-s4-16kb	16kb	1 wks	64 wkrs
w14-s5-32kb	32kb	1 wks	128 wkrs
w14-s6-64kb	64kb	1 wks	256 wkrs
w14-s7-128kb	128kb	1 wks	512 wkrs
w14-s8-256kb	256kb	1 wks	1,024 wkrs
w14-s9-512kb	512kb	1 wks	2,048 wkrs
w14-s10-1mb	1mb	1 wks	4,096 wkrs

图 5. 21 压力测试所用负载

结果如图 5. 22

op1: read	73.36 kops	73.36 MB	10.43 ms	3	10.43 ms	2445.66 op/s	2.45 MB/S	100%
op2: write	18.29 kops	146.35 MB	10.43 ms	4	10.43 ms	609.9 op/s	4.88 MB/S	100%
op1: read	142.65 kops	142.65 MB	10.73 ms	5	10.73 ms	4755.75 op/s	4.76 MB/S	100%
op2: write	35.72 kops	571.47 MB	10.73 ms	6	10.73 ms	1190.78 op/s	19.05 MB/S	100%
op1: read	192.16 kops	192.16 MB	15.99 ms	7	15.99 ms	6407.29 op/s	6.41 MB/S	100%
op2: write	47.79 kops	1.53 GB	15.9 ms	8	15.89 ms	1593.33 op/s	50.99 MB/S	100%
op1: read	200.95 kops	200.95 MB	30.6 ms	9	30.6 ms	6702.79 op/s	6.7 MB/S	100%
op2: write	50.32 kops	3.22 GB	30.18 ms	10	30.18 ms	1678.64 op/s	107.43 MB/S	100%
op1: read	203.61 kops	203.61 MB	60.52 ms	11	60.52 ms	6797.62 op/s	6.8 MB/S	100%
op2: write	50.81 kops	6.5 GB	59.17 ms	12	59.16 ms	1696.4 op/s	217.14 MB/S	100%
op1: read	199.99 kops	199.99 MB	123.03 ms	13	123.03 ms	6700.61 op/s	6.7 MB/S	100%
op2: write	50.08 kops	12.82 GB	118.81 ms	14	118.79 ms	1677.92 op/s	429.55 MB/S	100%
op1: read	172.33 kops	172.33 MB	285.88 ms	15	285.88 ms	5793.92 op/s	5.79 MB/S	100%
op2: write	42.78 kops	21.9 GB	272.2 ms	16	272.16 ms	1438.32 op/s	736.42 MB/S	100%
op1: read	161.68 kops	161.68 MB	603.18 ms	17	603.18 ms	5493.91 op/s	5.49 MB/S	100%
op2: write	40.4 kops	40.4 GB	569.67 ms	18	569.6 ms	1372.82 op/s	1.37 GB/S	100%

图 5. 22 压力测试下 s3proxy 结果

相同条件下，使用 minio 服务进行测试（workload 相同），结果如图 5. 23

op1: read	73.43 kops	73.43 MB	10.4 ms	3	10.4 ms	2448.66 op/s	2.45 MB/S	100%
op2: write	18.44 kops	147.52 MB	10.4 ms	4	10.4 ms	614.9 op/s	4.92 MB/S	100%
op1: read	145.63 kops	145.63 MB	10.53 ms	5	10.53 ms	4857.58 op/s	4.86 MB/S	100%
op2: write	35.96 kops	575.42 MB	10.54 ms	6	10.54 ms	1199.63 op/s	19.19 MB/S	100%
op1: read	238.51 kops	238.51 MB	12.84 ms	7	12.84 ms	7957.5 op/s	7.96 MB/S	100%
op2: write	59.46 kops	1.9 GB	12.88 ms	8	12.88 ms	1983.85 op/s	63.48 MB/S	100%
op1: read	215.67 kops	215.67 MB	28.4 ms	9	28.4 ms	7195.81 op/s	7.2 MB/S	100%
op2: write	54.06 kops	3.46 GB	28.5 ms	10	28.5 ms	1803.84 op/s	115.45 MB/S	100%
op1: read	192.37 kops	192.37 MB	63.96 ms	11	63.96 ms	6425.41 op/s	6.43 MB/S	100%
op2: write	47.72 kops	6.11 GB	63.23 ms	12	63.22 ms	1593.81 op/s	204.01 MB/S	100%
op1: read	176.52 kops	176.52 MB	139.58 ms	13	139.58 ms	5910.77 op/s	5.91 MB/S	100%
op2: write	44.15 kops	11.3 GB	134.44 ms	14	134.42 ms	1478.47 op/s	378.49 MB/S	100%
op1: read	163.38 kops	163.38 MB	298.85 ms	15	298.85 ms	5522.67 op/s	5.52 MB/S	100%
op2: write	41.14 kops	21.07 GB	285.52 ms	16	285.48 ms	1390.68 op/s	712.03 MB/S	100%
op1: read	163.47 kops	163.47 MB	591.03 ms	17	591.03 ms	5574.97 op/s	5.57 MB/S	100%
op2: write	41.36 kops	41.36 GB	567.05 ms	18	566.97 ms	1410.53 op/s	1.41 GB/S	100%

图 5. 23 压力测试下 minio 结果

分析 s3、s4，workers 为 32，64，我们知道，这个时候是不会出现拥挤堵塞的，所以两个服务端表现的都十分良好。

分析 s5，workers 为 128，minio（12ms）的平均延迟比 s3proxy（15ms）低了 3ms，读写速率都明显优于 s3proxy。

分析 s6，workers 为 256，minio（28ms）的平均延迟比 s3proxy（30ms）低

了 2ms，读写速率优于 s3proxy。

分析 s7，workers 为 512，minio (63ms) 的平均延迟比 s3proxy (60ms) 高了 3ms，读写速率略低于 s3proxy。

分析 s8，workers 为 1024，minio (137ms) 的平均延迟比 s3proxy (120ms) 高了 17ms，读写速率明显低于 s3proxy。

分析 s9，workers 为 2048，minio (293ms) 的平均延迟比 s3proxy (278ms) 高了 15ms，读写速率明显低于 s3proxy。

分析 s10，workers 为 4096，minio (578ms) 的平均延迟比 s3proxy (586ms) 低了 8ms，读写速率略高于 s3proxy。

经过以上分析可以得到结论：

当 workers 较少时，两者的表现持平。

当 workers 增加到出现拥塞时（平均延迟开始增加），minio 的表现比 s3proxy 好。

但当 workers 继续增加（>256），minio 的表现比 s3proxy 差。

当 workers 增加到使网络非常拥挤时（>1024），两者都会因为因为拥塞排队出现大量延时，基本上没有优劣之分。

总体而言，s3proxy 略优于 minio。

## 六、实验总结

第一天实验课本来是用 Windows 跑的，结果 COSBench 一直失败。试了一个晚上还是不行，现在想，应该是因为以前在 Windows 上为了装很多不同的软件，装了多个版本的 JDK，Java 的库、环境配置可能都被打乱了，所以 COSBench 没有正确的找到 Java 库，所以我转移到了 Linux 平台，就跑成功了，相当于白白浪费了一个晚上，不过这也给我一个提醒，以后要做好软件管理，已经存在的软件尽量不要再装其他版本。

在 Linux 成功跑了 minio 和 mc，开始使用 S3proxy 和 S3cmd。因为另一课程的原因，重装了系统，无法启动 COSBench，改用 Docker，使用 scalability/cosbench，然后就发现一个关于网络连接的问题。Docker 容器有四种网络模式：host、container、none、bridge，默认的模式是 bridge，在这种模式下，Docker 有自己的 IP 地址，主机就充当路由器。在这种情况下，Docker 容器内打开了 COSBench，但主机是监听不到 Docker 的端口的，所以我们需要将网络模式改为 host，在 host 模式下，容器和主机共用一个 Network Namespace，不再有自己的虚拟网卡，IP 地址，而是使用主机的 IP 地址和端口，使用 `--net=host` 来指定。

但是在第二天重新尝试 COSBench0.4.2.c4，发现成功了，所以就不再使用 Docker 容器了。

配置 s3cmd 的时候，一直提示找不到配置文件。我跑到目录去看，用 `ls -all` 命令发现文件是有的，我试着把权限提高到 777，继续 run，还是提示找不到配置文件。我就很纳闷，于是问老师，老师说是 key 的问题，加了 key 之后问题果然解决了。之前没加 key 是因为文档里面说 key 是 amoson 的 s3 账户，我们没有阿，而且 s3cmd 配置的时候提示 Access key and Secret key are your identifiers for Amazon S3. Leave them empty for using the env variables. 我就把它给留空了。没想到居然是这里的问题。不过我搞不懂，如果是 key 的问题，那为什么配置 key 为空的时候它会提示如图 6.1，它这样提示之后我以为我的配置都是正确的，已经正确连接到服务端了。所以就从其他方面考虑这个问题了。



```
Test access with supplied credentials? [Y/n] y
Please wait, attempting to list all buckets...
Success. Your access key and secret key worked fine :-)
```

图 6.1 配置 key 为空的结果

此外，我还认真拜读了《COSBench: A Benchmark Tool for Cloud Object Storage Services》这篇论文，这是我读的与对象存储相关的第一篇论文。里面的一些内容我已经在 COSBenchUserGuide 了解过了，同时也发现了一些新的知识点。比如 Containers are just like directories except there is no sub-container

.Objects are regular files though with limitations。这让我更好的理解了 Containers 和 Objects 以及两者的关系。另外，我还了解到了 controller 和 driver 的区别。COSBench 有两种模式：一种是 independent，只需要用到 driver；另一种是 managed，这时 controller 和 driver 都需要，controller 起到监管的作用，使多个 driver 在分布式环境中能够协调工作。

至此，本次的对象存储实验就算是结束了。在本次实验中，我学到了对象存储服务器 minio、s3proxy，客户端 mc、s3cmd，对于测评工具 COSBench 有了一定的体会，发现了 Docker 这种非常有用的工具，可以算是收货颇丰了。

最后，感谢同学，老师在本次试验中给我的大量帮助，让我能顺利完成此次试验。

## 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.