2015 级

《物联网数据存储与管理》课程

实 验 报 告

姓　名 龚慧媛

学　号 U201514892

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
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.....9

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。

Minio 是 Apache License v2.0 下发布的一款高性能的分布式对象存储服务器，专为大型私有云基础架构而设计。它与 Amazon S3 云存储服务兼容，最适合存储非结构化数据，如照片，视频，日志文件，备份和容器/虚拟机映像。

S3Proxy 是使用 Go 编写的 AWS S3 专用存储 bucket 的 http 代理。通过 AWS API 凭证，S3Proxy 允许通过未经身份验证的 HTTP 请求私有 S3 存储 bucket 下载对象，使得 curl 等工具可以轻松访问专用存储 bucket。

三、实验环境

表 1 实验环境

操作系统版本	Ubuntu 16.04.2
内核版本	Linux ubuntu 4.8.0-36-generic
Java 版本	java version "1.8.0_171"
Python	Python 2.7.12
Windows OS 版本	16299.371
服务器端	Minio、s3proxy
客户端	mc、s3cmd

四、实验内容

首先搭建 python 和 Java 的基础环境，然后再进行服务器端和客户端的搭建，最后启动 cosbench 提交测试文件。

4.1 对象存储技术实践

1. 在 Windows 环境下下载配置 minio 服务器端；
2. 下载 mc 客户端，创建 bucket，上传文件；
3. 安装使用 cosbench 提交 xml 文件测试；
4. 在 Linux 环境下安装 s3proxy 作为服务器；
5. 安装配置 s3cmd 作为客户端，创建 bucket，上传文件；
6. 安装使用 cosbench 提交 xml 文件测试。

4.2 对象存储性能分析

1. 测试读与写的性能优劣；
2. 修改 worker 的值，分析其对于吞吐率、带宽等的影响。
3. 比较不同的服务器端和客户端的性能差别。

五、实验过程

1. Minio server-mc

下载 minio.exe 后，将其拖入命令行界面打开，输入 server　C:\minio 打开 minio 服务器，其中 C:\minio 是我本机准备作为服务器的地址。成功后在 dos 界面显示 endpoint、accessKey 和 secretKey，如图 1 所示。

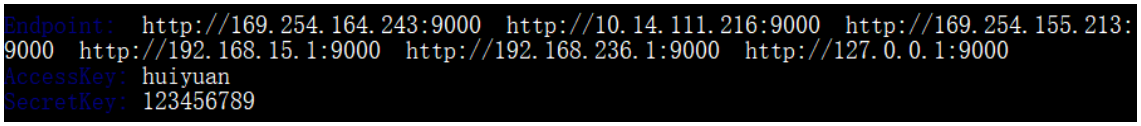


图 1 打开 minio 服务器

这两个 Key 可以在 C:\user\huiyuan\.minio\config.json 下更改。我使用的端点是 http://10.14.111.216:9000，在浏览器打开后显示图 2 所示，上传至 endpoint 的文件在这里都可以看到。

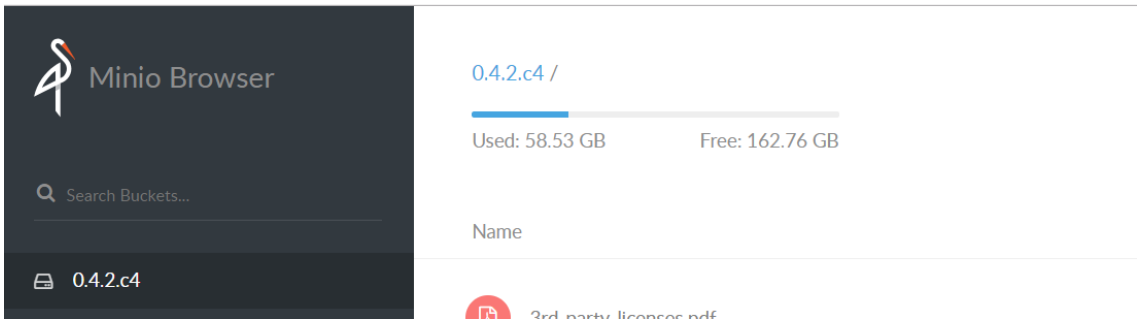


图 2 访问 endpoint

然后下载 mc 客户端，将其拖如 dos 界面，通过如图 3 所示界面添加主机。其中 minio 是想上传至的主机名字，后三个参数分别是 endpoint、accessKey 和 secretKey，最后则是选择的 API 签名 默认设置为“S3v4”。

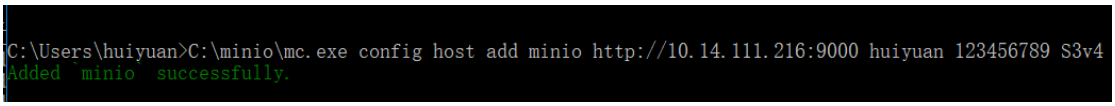


图 3 添加主机

服务器中所有的文件是放在相应的 bucket 中，客户端可以新建自己相应的 Bucket，如图 4 所示。mb 即为 makebucket 的简写，服务器主机名为 minio,bucket 名称为 mynewbucket。创建成功后即可在 Minio Browser 中看到新的 bucket，如图 5 所示。

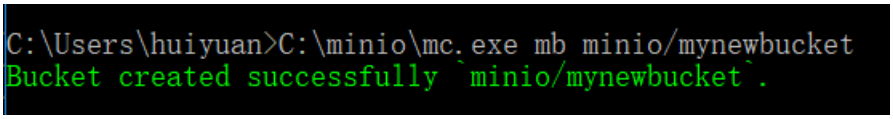


图 4 创建新的 bucket

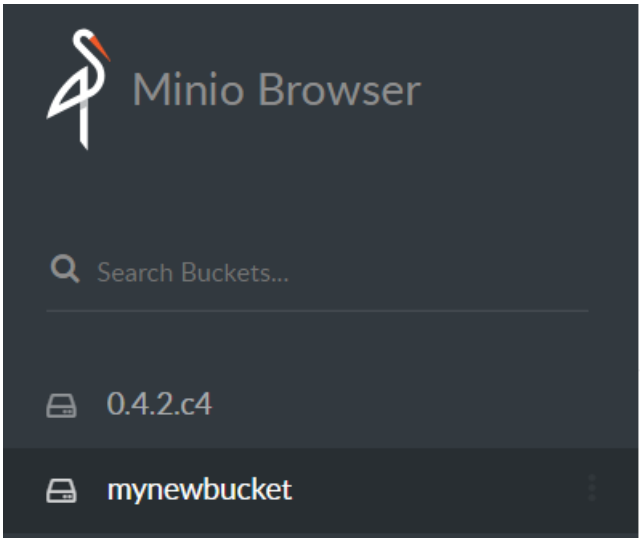


图 5 minio browser 显示新的 bucket

然后是上传文件测试。在 dos 界面输入 mc cp 要上传的文件名 目的服务器主机名/目的 bucket ，mc 通过将 mc.exe 拖入 dos 界面实现，如图 6 所示。此时刷新 endpoint 的网页就可以在相应 bucket 下看到刚传进去的文件，如图 7。

```
C:\Users\huiyuan>C:\minio\mc.exe cp C:\workload-example.xml minio/mynewbucket
...example.xml:  4.20 KB / 4.20 KB [================================] 100.00% 220.57 KB/s 0s
C:\Users\huiyuan>
```

图 6 上传文件

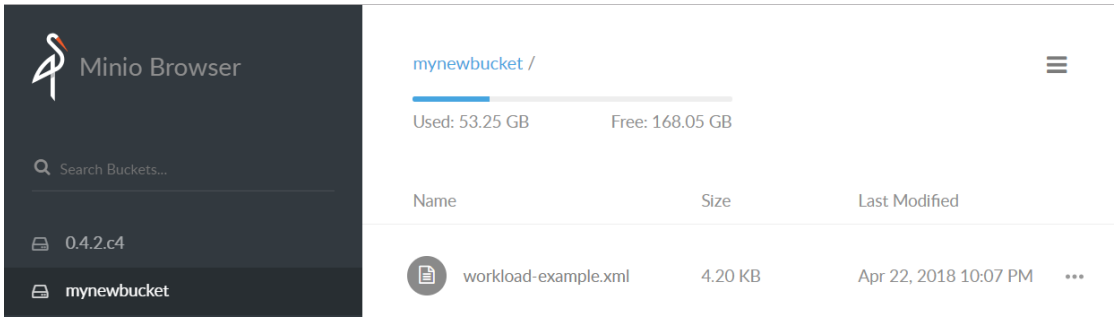


图 7 在相应的 bucket 里面看到新文件

下载 cosbench 压缩包，解压后进入该目录下，点击 start-all 批处理文件启动 driver 和 controller，再点击 web 批处理文件打开 cosbench 上传负载测试网页，然后上传 workload-example.xml，结果如图 8 所示。在此之前一定要将 workload-example.xml 文件里的 endpoint、accessKey 和 secretKey 修改正确。

ID	Name	Works	Workers	Op-Info	State	Link
w8-s1-init	init	1 wks	1 wkrs	init	completed	view details
w8-s2-prepare	prepare	8 wks	64 wkrs	prepare	completed	view details
w8-s3-8kb	8kb	1 wks	8 wkrs	read, write	completed	view details
w8-s4-16kb	16kb	1 wks	8 wkrs	read, write	completed	view details
w8-s5-32kb	32kb	1 wks	4 wkrs	read, write	completed	view details
w8-s6-64kb	64kb	1 wks	4 wkrs	read, write	completed	view details
w8-s7-128kb	128kb	1 wks	1 wkrs	read, write	completed	view details
w8-s8-256kb	256kb	1 wks	1 wkrs	read, write	completed	view details
w8-s9-512kb	512kb	1 wks	1 wkrs	read, write	completed	view details
w8-s10-1mb	1mb	1 wks	1 wkrs	read, write	completed	view details
w8-s11-cleanup	cleanup	1 wks	1 wkrs	cleanup	completed	view details
w8-s12-dispose	dispose	1 wks	1 wkrs	dispose	completed	view details

图 8 测试结果

2. S3proxy-s3cmd

在 Linux 下使用 Docker 运行 S3Proxy。在终端下输入 docker pull andrewgaul/s3proxy 获取镜像，然后使用 docker run --publish 80:80 --env S3PROXY_AUTHORIZATION=none andrewgaul/s3proxy 命令运行服务器，如下图 9 所示。

使用 pip 下载 s3cmd 后，输入 s3cmd –configure 进行配置，配置结果如图所示。然后 gedit /home/用户名/.s3cfg 修改部分信息如图 10。

```
New settings:
  Access Key: Q3AM3UQ867SPQQA43P2F
  Secret Key: zuf+tfteSlswRu7BJ86wekitnifILbZam1KYY3TG
  Default Region: US
  Encryption password: 123456
  Path to GPG program: /usr/bin/gpg
  Use HTTPS protocol: False
  HTTP Proxy server name:
  HTTP Proxy server port: 0
```

图 9　s3cmd 配置

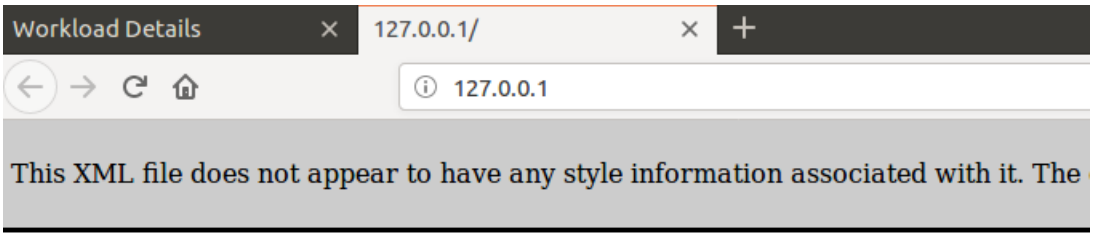
```
host_base = 127.0.0.1:80
host_bucket = 80
```

图 10　修改 s3cfg

接着是上传文件测试。输入 s3cmd mb s3://mybucket 命令创建新的 bucket，输入 s3cmd put /home/huiyuan/test.py s3://mybucket 命令将 test.py 文件上传至 mybucket，利用 ls 即可查看到该文件，如图 11 所示。另外打开 127.0.01 也能看到新添加的 bucket，如图 12 所示，说明客户端与服务器端连接成功。

```
huiyuan@ubuntu:~$ s3cmd put /home/huiyuan/test.py  s3://mybucket
upload: '/home/huiyuan/test.py' -> 's3://mybucket/test.py'  [1 of 1]
 2509 of 2509   100% in    0s    13.34 kB/s  done
huiyuan@ubuntu:~$ s3cmd ls s3://mybucket
2018-04-23 08:59      2509   s3://mybucket/test.py
huiyuan@ubuntu:~$
```

图 11　上传文件




```
−<ListAllMyBucketsResult>
  −<Owner>
    −<ID>
       75aa57f09aa0c8caeab4f8c24e99d10f8e7faeebf76c078efc7c6caea54ba06a
     </ID>
     <DisplayName>CustomersName@amazon.com</DisplayName>
   </Owner>
  −<Buckets>
    −<Bucket>
       <Name>mybucket</Name>
       <CreationDate>2018-04-23T16:30:23.000Z</CreationDate>
     </Bucket>
   </Buckets>
 </ListAllMyBucketsResult>
```

图 12　在 endpoint 查看 bucket

下载 cosbench 完成后，进入 cosbench，使用. start-all.sh 命令启动 driver 和 controller。然后用浏览器打开 http://127.0.0.1:19088/controller/index.html，然后上传 workload-example.xml，结果如图所示。

Current Stage	Stages completed	Stages remaining	Start Time	End Time	Time Remaining

ID	Name	Works	Workers	Op-Info	State	Link
w2-s1-init	init	1 wks	1 wkrs	init	completed	view details
w2-s2-prepare	prepare	8 wks	64 wkrs	prepare	completed	view details
w2-s3-8kb	8kb	1 wks	8 wkrs	read, write	completed	view details
w2-s4-16kb	16kb	1 wks	8 wkrs	read, write	completed	view details
w2-s5-32kb	32kb	1 wks	4 wkrs	read, write	completed	view details
w2-s6-64kb	64kb	1 wks	4 wkrs	read, write	completed	view details
w2-s7-128kb	128kb	1 wks	1 wkrs	read, write	completed	view details
w2-s8-256kb	256kb	1 wks	1 wkrs	read, write	completed	view details
w2-s9-512kb	512kb	1 wks	1 wkrs	read, write	completed	view details
w2-s10-1mb	1mb	1 wks	1 wkrs	read, write	completed	view details
w2-s11-cleanup	cleanup	1 wks	1 wkrs	cleanup	completed	view details
w2-s12-dispose	dispose	1 wks	1 wkrs	dispose	completed	view details

There are 12 stages in this workload.

show error statistics details

图 13 cosbench 测试文件结果

3. 性能分析

图 14 是 s3proxy-s3cmd 系统的测试结果详细，可以看到对于任何一个 Workstage，read 的带宽和吞吐率都明显优于 write。由此可得到结论：

就如同用户上传数据和下载数据时，上行带宽会小于下行带宽，对于 workstage，read 的性能会优于 write。

op1: read	2.67 kops	21.38 MB	68.47 ms	57.32 ms	89.11 op/s	712.89 KB/S	98.49%
op2: write	664 ops	5.31 MB	75.2 ms	64.91 ms	22.14 op/s	177.16 KB/S	100%
op1: read	4.08 kops	65.26 MB	43.29 ms	33.55 ms	136.06 op/s	2.18 MB/S	95.93%
op2: write	1.07 kops	17.14 MB	49.96 ms	39.73 ms	35.69 op/s	571.59 KB/S	100%
op1: read	3.4 kops	108.86 MB	23.62 ms	21.74 ms	113.46 op/s	3.63 MB/S	93.46%
op2: write	896 ops	28.67 MB	35.56 ms	21.63 ms	29.88 op/s	956.25 KB/S	100%
op1: read	2.89 kops	185.15 MB	23.43 ms	21.24 ms	96.46 op/s	6.17 MB/S	84.57%
op2: write	846 ops	54.14 MB	42.5 ms	20.34 ms	28.21 op/s	1.81 MB/S	100%
op1: read	2.04 kops	260.48 MB	9.24 ms	7.7 ms	67.85 op/s	8.68 MB/S	100%
op2: write	512 ops	65.54 MB	21.34 ms	6.7 ms	17.07 op/s	2.18 MB/S	100%
op1: read	1.02 kops	262.4 MB	15.28 ms	10.5 ms	34.18 op/s	8.75 MB/S	100%
op2: write	292 ops	74.75 MB	48.33 ms	10.95 ms	9.74 op/s	2.49 MB/S	100%
op1: read	840 ops	430.08 MB	17.64 ms	9.53 ms	28.03 op/s	14.35 MB/S	100%
op2: write	219 ops	112.13 MB	67.64 ms	8.75 ms	7.31 op/s	3.74 MB/S	100%
op1: read	493 ops	493 MB	32.59 ms	15.58 ms	16.51 op/s	16.51 MB/S	100%
op2: write	122 ops	122 MB	111.77 ms	12.4 ms	4.08 op/s	4.08 MB/S	100%

图 14　s3proxy-s3cmd 系统的测试细节

将 s3 和 s2（原 worker 为 8）都改为 16，重新提交测试文件，其结果如图 15 所示。观察吞吐率和带宽可以看到，s2 和 s3 的读写性能都得到了大幅度提升，其他的 workstage 也有小幅度提升，但是 read 的成功率都有小幅度下降。于是得到结论：

随着 worker 的增加，并行线程的数量增加，读写性能得到提升，但是可能

由于冲突碰撞等原因，成功率有小幅度下降。

op1: read	6.14 kops	49.08 MB	59.47 ms	46.79 ms	204.62 op/s	1.64 MB/S	98.02%
op2: write	1.6 kops	12.78 MB	66.13 ms	60.34 ms	53.23 op/s	426.36 KB/S	100%
op1: read	6.77 kops	108.32 MB	47.99 ms	41.2 ms	225.85 op/s	3.61 MB/S	92.49%
op2: write	1.86 kops	29.76 MB	63.25 ms	48.72 ms	62.02 op/s	992.8 KB/S	100%
op1: read	3.31 kops	105.98 MB	21.43 ms	20.48 ms	110.47 op/s	3.54 MB/S	90%
op2: write	893 ops	28.58 MB	45.75 ms	34.18 ms	29.79 op/s	953.12 KB/S	100%
op1: read	3.67 kops	235.07 MB	17.65 ms	16.02 ms	122.48 op/s	7.84 MB/S	84.03%
op2: write	1.09 kops	69.63 MB	36.67 ms	16.56 ms	36.28 op/s	2.32 MB/S	100%
op1: read	1.96 kops	251.52 MB	8.72 ms	7.04 ms	65.56 op/s	8.39 MB/S	100%
op2: write	503 ops	64.38 MB	25.11 ms	7.95 ms	16.78 op/s	2.15 MB/S	100%
op1: read	1.86 kops	476.93 MB	8.85 ms	6 ms	62.15 op/s	15.91 MB/S	100%
op2: write	438 ops	112.13 MB	30.53 ms	5.7 ms	14.61 op/s	3.74 MB/S	100%
op1: read	1.09 kops	557.57 MB	13.64 ms	7.5 ms	36.3 op/s	18.59 MB/S	100%
op2: write	286 ops	146.43 MB	52.73 ms	7.17 ms	9.53 op/s	4.88 MB/S	100%
op1: read	694 ops	694 MB	24.92 ms	12.1 ms	23.27 op/s	23.27 MB/S	100%
op2: write	152 ops	152 MB	82.09 ms	8.03 ms	5.1 op/s	5.1 MB/S	100%

图 15　增加 s2 和 s3 的 worker 后测试结果

另外 minio 系统与 s3proxy 的 cosbench 测试结果分别如图 16 和 17 所示，此次测试结果中 minio 的吞吐率高于 s3proxy，s3proxy 的成功率高于 minio。但是由于两个系统搭建的操作系统不同，测试时的网络环境也不同，所以没有办法单纯比较两个系统的性能优劣。

w8-s3-1	read	3.96 ms	< 10 ms	422.44 op/s	81.55%
w8-s3-2	write	46.55 ms	< 100 ms	127.97 op/s	100%
w8-s4-1	read	3.58 ms	< 10 ms	474.84 op/s	81.88%
w8-s4-2	write	42.59 ms	< 90 ms	139.47 op/s	100%
w8-s5-1	read	3.22 ms	< 10 ms	281.31 op/s	79.84%
w8-s5-2	write	33.44 ms	< 70 ms	84.98 op/s	100%
w8-s6-1	read	3.45 ms	< 10 ms	248.05 op/s	76%
w8-s6-2	write	34.37 ms	< 60 ms	82.96 op/s	100%
w8-s7-1	read	5.74 ms	< 10 ms	62.2 op/s	100%
w8-s7-2	write	40.4 ms	< 60 ms	15.83 op/s	100%
w8-s8-1	read	7.43 ms	< 20 ms	58.4 op/s	100%
w8-s8-2	write	40.91 ms	< 70 ms	13.77 op/s	100%
w8-s9-1	read	10.49 ms	< 20 ms	43.71 op/s	100%
w8-s9-2	write	46.44 ms	< 70 ms	11.6 op/s	100%
w8-s10-1	read	12.07 ms	< 30 ms	47.56 op/s	100%
w8-s10-2	write	37.91 ms	< 70 ms	11.19 op/s	100%

图 16　minio 性能测试

ID	Op-Type	Avg-ResTime	95%-ResTime	Throughput	Succ-Ratio	Link
w3-s3-8kb-1	read	68.47 ms	< 150 ms	89.11 op/s	98.49%	view details
w3-s3-8kb-2	write	75.2 ms	< 160 ms	22.14 op/s	100%	view details
w3-s4-16kb-1	read	43.29 ms	< 100 ms	136.06 op/s	95.93%	view details
w3-s4-16kb-2	write	49.96 ms	< 110 ms	35.69 op/s	100%	view details
w3-s5-32kb-1	read	23.62 ms	< 50 ms	113.46 op/s	93.46%	view details
w3-s5-32kb-2	write	35.56 ms	< 70 ms	29.88 op/s	100%	view details
w3-s6-64kb-1	read	23.43 ms	< 50 ms	96.46 op/s	84.57%	view details
w3-s6-64kb-2	write	42.5 ms	< 70 ms	28.21 op/s	100%	view details
w3-s7-128kb-1	read	9.24 ms	< 20 ms	67.85 op/s	100%	view details
w3-s7-128kb-2	write	21.34 ms	< 50 ms	17.07 op/s	100%	view details
w3-s8-256kb-1	read	15.28 ms	< 40 ms	34.18 op/s	100%	view details
w3-s8-256kb-2	write	48.33 ms	< 90 ms	9.74 op/s	100%	view details
w3-s9-512kb-1	read	17.64 ms	< 50 ms	28.03 op/s	100%	view details
w3-s9-512kb-2	write	67.64 ms	< 140 ms	7.31 op/s	100%	view details
w3-s10-1mb-1	read	32.59 ms	< 80 ms	16.51 op/s	100%	view details
w3-s10-1mb-2	write	111.77 ms	< 240 ms	4.08 op/s	100%	view details

图 17 s3proxy-s3cmd 性能测试

六、实验总结

通过此次实验，我基本了熟悉对象存储技术的代表性系统及其特性，并且部署实验环境，成功地进行了初步测试。

起初在 windows 下搭建 minio 时，每一步都挺顺利的，直到跑 cosbench 的时候，前后两个 workstage 都能 completed，但是中间 8 个读写的就一直 failed，自己苦苦思索好久也没找到问题所在。后来翻阅老师给的 know-issue 文档，其中说过可以找 workload.log 寻找 error，于是打开文件，发现如下图错误，猜想是不是 HTTP 地址给错了，赶紧翻出 workload-examp.xml，发现果然如此。所以说很多错误都是由于自己不够细心造成，由此造成大量时间浪费真的可惜。

```
Unable to execute HTTP request: Remote host closed connection during handshake
Unable to execute HTTP request: Remote host closed connection during handshake
Unable to execute HTTP request: Remote host closed connection during handshake
Unable to execute HTTP request: Remote host closed connection during handshake
Unable to execute HTTP request: Remote host closed connection during handshake
Unable to execute HTTP request: Remote host closed connection during handshake
Unable to execute HTTP request: Remote host closed connection during handshake
Unable to execute HTTP request: Remote host closed connection during handshake
```

图 18 workload.log 错误信息

但是改好之后又出现了新的问题: s6 一直 failed。有了前面的经验，我马上打开 workstage 文档，发现错误信息:“com.amazonaws.AmazonClientException: Unable to verify integrity of data download. Client calculated content hash didn't match hash calculated by Amazon S3. The data may be corrupt.”上网查询后发现是因为 S3 客户端根据其源代码的 eTag 检查内容 MD5。如果由于某种原因，此验证失败，则下载的数据可能是虚假的。编辑 cosbench-start.sh 文件,找到 java 启动命令行参数, 增加"-Dcom.amazonaws.services.s3.disableGetObjectMD5Validation=true" 关闭 S3 的 MD5 校验功能，再次 submit 时，全部通过。

然后时在 linux 环境下配置 s3proxy-s3cmd 系统，使用 s3cmd -configure 配置 s3cmd 时，我的设置似乎少了某些步骤如设置端点和端口，于是我用 gedit 打开了.s3cfg 文件，手动更改配置，后来进行创建 bucket 测试就成功了。最后是配置 cosbench 时，driver 一直打不开，我反复确认配置步骤与文件也没有找到问题所在，后来经同学提醒才知道原来时 Java 版本不对，心情难以言喻。

此次实验至此终于结束了，我从中学习到了课本之外的知识，也提高了自己寻找 bug 以及解决问题的能力。感谢老师以及同学的帮助。

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.