

2016 级

《物联网数据存储与管理》课程
实 验 报 告

姓 名 潘翔

学 号 U201614898

班 号 物联网 1601 班

日 期 2019.06.05

目 录

1 实验目的.....	1
2 实验背景.....	1
2.1 ACCOUNT SERVER.....	1
2.2 CONTAINER.....	1
2.3 OBJECT SERVER.....	2
3 实验环境.....	3
3.1 系统环境.....	3
3.2 MINIO-SERVER.....	3
3.3 MINIO-PYTHON API.....	5
3.4 AWS-CLI.....	7
3.5 S3BENCH.....	8
3.6 COSBENCH 测试.....	9
3.7 OPENSTACK SWIFT CLI.....	9
3.8 OPENSTACK SWIFT PYTHON-API.....	11
4 实验内容.....	12
4.1 对象存储技术实践.....	12
4.2 对象存储性能分析.....	12
5 实验过程.....	12
5.1 环境配置.....	12
5.2 基于 COSBENCH 进行单变量测试.....	13
5.3 进行 HDD/SSD 性能比较.....	17
6 实验总结.....	20
参考文献.....	20

1 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。

2 实验背景

OpenStack Object Storage 是一个高度可用，分布式，最终一致的对象 blob 存储。可以使用 Object Storage API 创建，修改和获取对象和元数据，该 API 是作为一组 Representational State Transfer (REST) Web 服务实现的。

2.1 Account Server

表示层次结构的顶级

服务提供商会创建帐户，并拥有该帐户中的所有资源。该帐户定义容器的命名空间。

容器在两个不同的帐户中可能具有相同的名称。

在 OpenStack 环境中，帐户与项目或租户同义。

2.2 Container

定义对象的命名空间。两个不同容器中具有相同名称的对象表示两个不同的对象。

可以在帐户中创建任意数量的容器。

除了包含对象之外，还可以使用容器通过使用访问控制列表 (ACL) 来控制对对象的访问。不能使用单个对象存储 ACL。

在容器级别配置和控制许多其他功能，例如对象版本控制。

可以在一个请求中批量删除多达 10,000 个容器。

可以在具有云提供商的预定义名称和定义的容器上设置存储策略。

2.3 Object Server

存储数据内容，例如文档，图像等。还可以使用对象存储自定义元数据。

使用 Object Storage API,

存储无限数量的对象。每个对象可以大到 5 GB，这是默认值。可以配置最大对象大小。

- 1) 使用大对象创建上载和存储任何大小的对象。
- 2) 使用跨源资源共享来管理对象安全性。
- 3) 使用内容编码元数据压缩文件。
- 4) 使用内容处置元数据覆盖对象的浏览器行为。
- 5) 安排删除对象。
- 6) 在单个请求中批量删除最多 10,000 个对象。
- 7) 自动提取存档文件。
- 8) 生成一个 URL，该 URL 提供对对象的时间限制 GET 访问。
- 9) 使用表单 POST 中间件从浏览器直接将对象上传到 Object Storage 系统。
- 10) 创建指向其他对象的符号链接。

3 实验环境

3.1 系统环境.

OS:Manjaro 18.0.4 Illyria

Kernel: x86_64 Linux 5.0.18-1-MANJARO

Shell: zsh 5.7.1

Docker:18.09.6-ce

Swift: Docker Version(Swift All In One)

Cosbench:4.1/4.2

Storage:HDD/SSD

3.2 Minio-Server

```
hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs master minio server ./data
Endpoint: http://10.11.57.137:9000 http://127.0.0.1:9000
AccessKey: K6V0APDE34MX8Z53MMTK
SecretKey: WfYTKb+EMNw8nQT3Zi8nah8g2hslkg4+wyotlCHV

Browser Access:
http://10.11.57.137:9000 http://127.0.0.1:9000

Command-line Access: https://docs.minio.io/docs/minio-client-quickstart-guide
$ mc config host add myminio http://10.11.57.137:9000 K6V0APDE34MX8Z53MMTK WfYTKb+EMNw8nQT3Zi8nah8g2hslkg4+wyotlCHV

Object API (Amazon S3 compatible):
Go: https://docs.minio.io/docs/golang-client-quickstart-guide
Java: https://docs.minio.io/docs/java-client-quickstart-guide
Python: https://docs.minio.io/docs/python-client-quickstart-guide
JavaScript: https://docs.minio.io/docs/javascript-client-quickstart-guide
.NET: https://docs.minio.io/docs/dotnet-client-quickstart-guide
```

图 3.2 Minio-Server 启动验证

```
hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs master mc config host add minioserver http://10.11.57.137:9000 K6V0APDE34MX8Z53MMTK WfYTKb+EMNw8nQT3Zi8nah8g2hslkg4+wyotlCHV
```

图 3.3 Minio Host 添加

此时在浏览器打开 127.0.0.1:9000(endpoint)，在打开的 minio browser，输入自己的 AccessKey 和 SerectKey 登陆。

此时可以选择右下角红色标记随意添加 bucket 和上传文件，方便地实现类似网盘的功能。

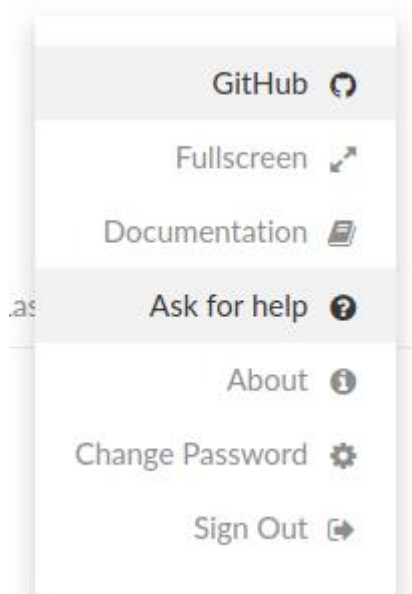


图 3.4 账户管理相关选项

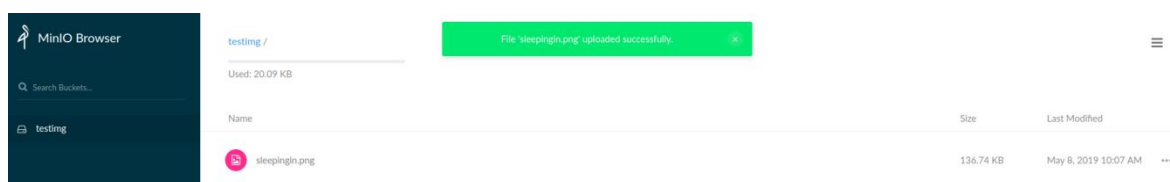


图 3.5 Bucket/file 测试图

注意，bucket 有相应的命名限制，只能采用小写字母，而不允许下划线和相应大写字母的出现。

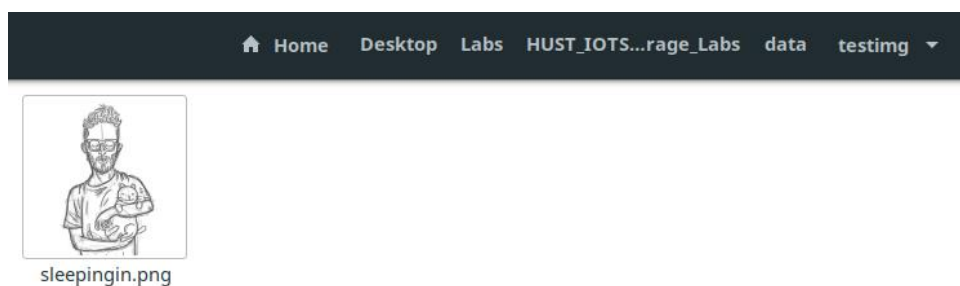


图 3.6 本地文件验证

华中科技大学课程实验报告

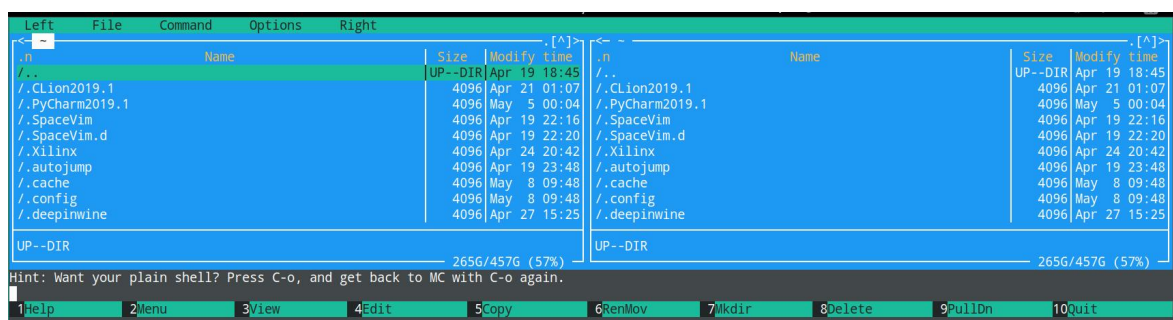


图 3.7 MC 界面

3.3 Minio-python API

使用 python-tutorial 进行 API 编程测试，使用官方测试仓库，代码如下

代码 3.1 Minio-python API

```
from minio import Minio
from minio.error import ResponseError
from minio.error import (ResponseError, BucketAlreadyOwnedByYou,
                          BucketAlreadyExists)

minioClient = Minio('play.min.io:9000',
                    access_key='Q3AM3UQ867SPQQA43P2F',
                    secret_key='zuf+tfteSlsWRu7BJ86wekitnifILbZam1KYY3TG',
                    secure=True)

# Make a bucket with the make_bucket API call.
try:
    minioClient.make_bucket("maylogs", location="us-east-1")
except BucketAlreadyOwnedByYou as err:
    pass
except BucketAlreadyExists as err:
    pass
```

华中科技大学课程实验报告

```
except ResponseError as err:

    raise

else:

    # Put an object 'pumaserver_debug.log' with contents from 'pumaserver_debug.log'.

    try:

        minioClient.fput_object('maylogs', 'pumaserver_debug.log',

'/tmp/pumaserver_debug.log')

    except ResponseError as err:

        print(err)
```

运行结果如下

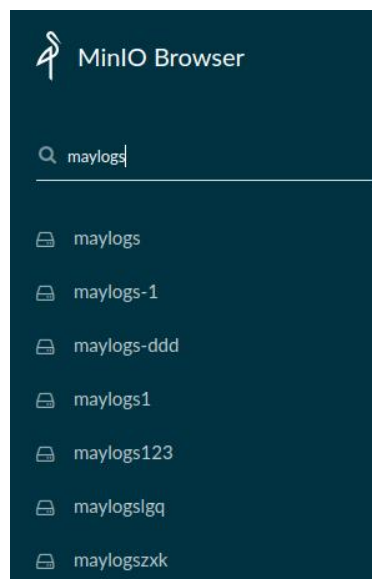


图 3.8 Minio-python 测试结果

华中科技大学课程实验报告

3.4 AWS-CLI

```
x hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs master aws configure
AWS Access Key ID [None]: Q3AM3UQ867SPQQA43P2F
AWS Secret Access Key [None]: zuf+tfteSlswRu7BJ86wekitnifILbZam1KYY3TG
Default region name [None]: us-east-1
Default output format [None]:
hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs master aws configure set default.s3.signature_version s3v4
hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs master aws --endpoint-url https://play.min.io:9000 s3 ls
```

图 3.9 使用官方测试 AWS-CLI 测试

```
hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs master aws --endpoint-url https://play.min.io:9000 s3 ls
2019-05-07 13:29:07 00test
2019-05-07 15:25:26 00zq
2019-05-07 13:30:32 0123
2019-05-08 10:31:02 0chreuropetrip
2019-05-08 00:35:11 0q2w1gcrcuo1c790jk3j1313lo7qije8
2019-05-08 09:50:22 0viay
```

图 3.10 此存储下的 bucket

```
x hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs master aws --endpoint-url https://play.min.io:9000 s3 mb s3://hvoerbucket
make_bucket: hvoerbucket
```

图 3.11 创建 bucket

```
hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs master aws --endpoint-url https://play.min.io:9000 s3 cp data/testing/sleepingin.png s3://hvoerbucket
upload: data/testing/sleepingin.png to s3://hvoerbucket/sleepingin.png
```

图 3.12 上载 IMG 测试

```
hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs master aws --endpoint-url https://play.min.io:9000 s3 rm s3://hvoerbucket/sleepingin.png
delete: s3://hvoerbucket/sleepingin.png
```

图 3.13 删除 IMG 测试

华中科技大学课程实验报告

3.5 s3bench

使用如下命令进行测试，其中参数可自行调节

```
go run ./s3bench.go -accessKey=KEY -accessSecret=SECRET -bucket=loadgen
-endpoint=http://endpoint1:80,http://endpoint2:80 -numClients=2 -numSamples=10
-objectNamePrefix=loadgen -objectSize=1024
```

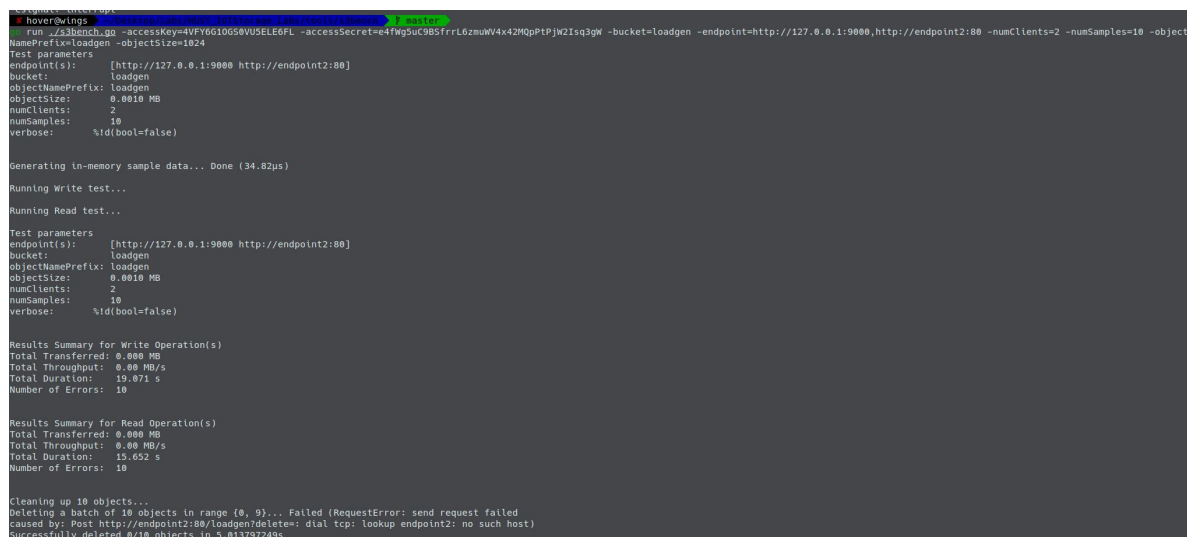
```
#!/bin/bash

for i in {1..10};do

    ~/go/bin/s3-benchmark -a 4VFY6G10GS0VU5ELE6FL -s e4fWg5uC9BSfrrL6zmuW
V4x42MQpPtPjW2Isq3gW -u http://127.0.0.1:9000 -b wasabi-benchmark \

    -d 3 -t i -z 1K | tail -1 -al > result.csv

done
```



```
hovers@hovers:~/go$ go run ./s3bench.go -accessKey=KEY -accessSecret=SECRET -bucket=loadgen -endpoint=http://127.0.0.1:9000,http://endpoint2:80 -numClients=2 -numSamples=10 -objectNamePrefix=loadgen -objectSize=1024
Test parameters
endpoint(s): [http://127.0.0.1:9000 http://endpoint2:80]
bucket: loadgen
objectNamePrefix: loadgen
objectSize: 0.0010 MB
numClients: 2
numSamples: 10
verbose: %d(bool=false)

Generating in-memory sample data... Done (34.82us)
Running Write test...
Running Read test...
Test parameters
endpoint(s): [http://127.0.0.1:9000 http://endpoint2:80]
bucket: loadgen
objectNamePrefix: loadgen
objectSize: 0.0010 MB
numClients: 2
numSamples: 10
verbose: %d(bool=false)

Results Summary for Write Operation(s)
Total Transferred: 0.000 MB
Total Throughput: 0.00 MB/s
Total Duration: 19.471 s
Number of Errors: 10

Results Summary for Read Operation(s)
Total Transferred: 0.000 MB
Total Throughput: 0.00 MB/s
Total Duration: 19.652 s
Number of Errors: 10

Cleaning up 10 objects...
Deleting a batch of 10 objects in range (0, 9)... Failed (RequestError: send request failed
caused by: Post http://endpoint2:80/loadgen/delete: dial tcp: lookup endpoint2: no such host)
Successfully deleted 0/10 objects in 5.013797249s
```

图 3.14 s3bench 测试

华中科技大学课程实验报告

3.6 CosBench 测试

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Rati
init-write	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
prepare-write	8 ops	64 KB	49.38 ms	49.25 ms	165.61 op/s	1.32 MB/S	100%
prepare-write	8 ops	128 KB	35.25 ms	35.25 ms	223.89 op/s	3.58 MB/S	100%
prepare-write	8 ops	256 KB	48.5 ms	48.38 ms	173.19 op/s	5.54 MB/S	100%
prepare-write	8 ops	512 KB	36.38 ms	35.75 ms	228.2 op/s	14.6 MB/S	100%
prepare-write	8 ops	1.02 MB	34.5 ms	32.12 ms	230.43 op/s	29.5 MB/S	100%
prepare-write	8 ops	2.05 MB	21.25 ms	19.5 ms	371.49 op/s	95.1 MB/S	100%
prepare-write	8 ops	4.1 MB	13.12 ms	10.38 ms	615.81 op/s	315.3 MB/S	100%
prepare-write	8 ops	8 MB	61.12 ms	52.12 ms	131.76 op/s	131.76 MB/S	100%
read	59.18 kops	473.45 MB	1.25 ms	1.25 ms	1973.17 op/s	15.79 MB/S	100%
write	14.91 kops	119.31 MB	7.97 ms	7.97 ms	497.25 op/s	3.98 MB/S	100%
read	55.1 kops	881.55 MB	1.37 ms	1.37 ms	1836.71 op/s	29.39 MB/S	100%
write	13.78 kops	220.43 MB	8.8 ms	8.8 ms	459.27 op/s	7.35 MB/S	100%
read	34.84 kops	1.11 GB	1.29 ms	1.29 ms	1161.29 op/s	37.16 MB/S	100%
write	8.72 kops	279.1 MB	5.42 ms	5.42 ms	290.75 op/s	9.3 MB/S	100%
read	33.87 kops	2.17 GB	1.42 ms	1.42 ms	1129.09 op/s	72.26 MB/S	100%
write	8.54 kops	546.56 MB	5.49 ms	5.49 ms	284.69 op/s	18.22 MB/S	100%
read	7.88 kops	1.01 GB	2.02 ms	2.02 ms	262.53 op/s	33.6 MB/S	100%
write	2 kops	255.36 MB	5.08 ms	5.03 ms	66.51 op/s	8.51 MB/S	100%
read	6.73 kops	1.72 GB	2.21 ms	1.99 ms	224.21 op/s	57.4 MB/S	100%
write	1.73 kops	443.9 MB	5.53 ms	4.5 ms	57.8 op/s	14.8 MB/S	100%
read	5.24 kops	2.68 GB	3.25 ms	2.03 ms	174.58 op/s	89.38 MB/S	100%
write	1.32 kops	675.33 MB	6.88 ms	4.79 ms	43.97 op/s	22.51 MB/S	100%
read	3.74 kops	3.74 GB	5.17 ms	2.04 ms	124.75 op/s	124.75 MB/S	100%
write	879 ops	879 MB	9.38 ms	5.14 ms	29.3 op/s	29.3 MB/S	100%
cleanup-delete	128 ops	0 B	1.18 ms	1.18 ms	535.56 op/s	0 B/S	100%
dispose-delete	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A

图 3.15 cosbench 测试

3.7 OpenStack Swift CLI

使用 OpenStack Swift 进行相关测试

Searching for curl from log system... (Note: curl is not found in your environment, you are starting the container in a bash shell)						
hover@wings	~/Desktop/Labs/HUST_IOTStorage_Labs/tools/openstack-swift-docker	master	docker ps			
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
5fc16e8b99bf	openstack-swift-docker	"/bin/sh -c /usr/loc..."	5 minutes ago	Up 5 minutes	0.0.0.0:12345->8080/tcp	openstack-swift

图 3.16 Swift Docker 启动

hover@wings	swift -A http://127.0.0.1:12345/auth/v1.0 -U test:tester -K testing stat
Account: AUTH_test	
Containers: 0	
Objects: 0	
Bytes: 0	
Content-Type: text/plain; charset=utf-8	
X-Timestamp: 1557886895.94559	
X-Put-Timestamp: 1557886895.94559	
X-Trans-Id: txa0e0c68a8404424e812fe-005cdb77af	

图 3.17 Swift Docker 启动检测

华中科技大学课程实验报告

设置系统环境变量，避免每次命令行需要输入，由于采用 V1.0 验证，具有和 V2.0 不同的环境变量配置，需要注意。

```
# swift -A http://127.0.0.1:12345/auth/v1.0 -U test:tester
# http://127.0.0.1:12345/auth/v1.0 -U test:tester -K testing
# ST_AUTH, ST_USER, and ST_KEY
export ST_AUTH="http://127.0.0.1:12345/auth/v1.0"
export ST_USER="test:tester"
export ST_KEY="testing"
```

图 3.18 SwiftAUTH 环境变量设置

```
hover@wings:~$ swift -A http://127.0.0.1:12345/auth/v1.0 -U chris:chris1234 -K testing upload --object-name mypdf.pdf user_uploads /home/hover/Desktop/Labs/Back/HUST_RFID_Labs/RFID.pdf
/usr/lib/python3.7/site-packages/requests/__init__.py:91: RequestsDependencyWarning: urllib3 (1.25.2) or chardet (3.0.4) doesn't match a supported version!
RequestsDependencyWarning)
mypdf.pdf
```

图 3.19 Swift 上传测试

```
hover@wings:~$ swift -A http://127.0.0.1:12345/auth/v1.0 -U chris:chris1234 -K testing download user_uploads mypdf.pdf
/usr/lib/python3.7/site-packages/requests/__init__.py:91: RequestsDependencyWarning: urllib3 (1.25.2) or chardet (3.0.4) doesn't match a supported version!
RequestsDependencyWarning)
mypdf.pdf [auth 0.006s, headers 0.021s, total 0.024s, 39.851 MB/s]
```

图 3.20 Swift 下载测试

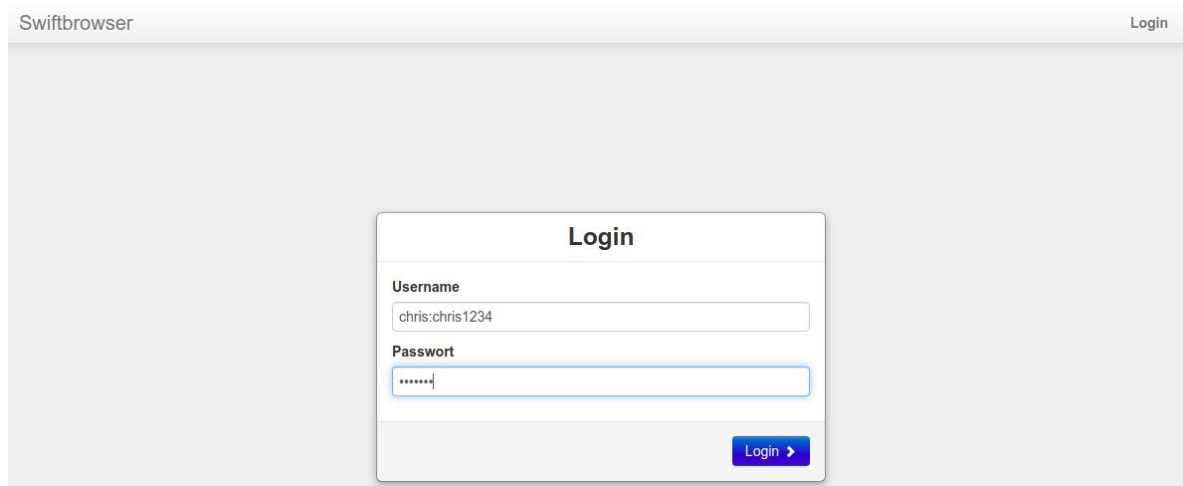
The image shows a web browser window titled "Swiftbrowser" with a "Login" button in the top right corner. In the center of the page is a "Login" form. The form has two input fields: "Username" with the text "chris:chris1234" and "Password" with masked characters "*****". Below the password field is a blue "Login" button with a right-pointing arrow.

图 3.21 Swift GUI-Client 测试

华中科技大学课程实验报告

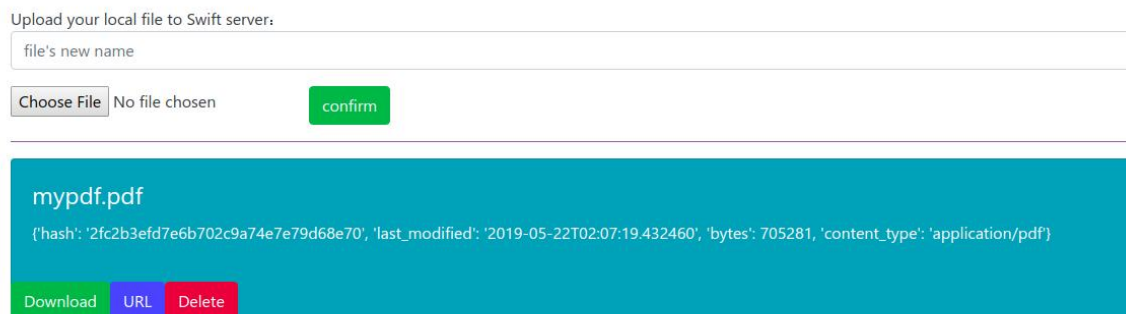


图 3.22 Swift GUI 测试

3.8 OpenStack Swift Python-API

```
hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs/data master ➤ swift post C1
hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs/data master ➤ swift stat
Account: AUTH_test
Containers: 1
Objects: 0
Bytes: 0
Containers in policy "policy-0": 1
Objects in policy "policy-0": 0
Bytes in policy "policy-0": 0
X-Timestamp: 1557888550.36011
Content-Type: text/plain; charset=utf-8
Accept-Ranges: bytes
X-Trans-Id: txe7620aaf2797455b9bb08-005cdb7e3a
```

图 3.23 Swift 创建 Container

```
hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs/data master ➤ swift upload C1 sleepingin.png
sleepingin.png
hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs/data master ➤ cd ..
hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs master ➤ swift upload C1 data
data/.minio.sys/tmp/e137626d-0a6b-4eb2-89bf-61622af27c4a
data/.minio.sys/tmp/e53272ab-42f3-4b46-a28c-c8888265a51e
data/.minio.sys/multipart
data/.minio.sys/tmp/fa8fef16-3bab-4e16-a4e5-2aa4bffa3f30
data/.minio.sys/config/config.json
data/.minio.sys/format.json
data/sleepingin.png
data/培养计划.pdf
```

图 3.24 Swift 上传文件和文件夹测试

4 实验内容

4.1 对象存储技术实践

- 1) 采用 Docker 进行环境相关配置
- 2) 相关 API 测试以及测试程序编写
- 3) 测试数据下载分析

4.2 对象存储性能分析

- 1) 对于不同情况下的测试结果进行绘图
- 2) 测试结果分析

5 实验过程

5.1 环境配置

- 1) 安装软件包

由于基于 Arch 系统优良的包管理和 AUR，所需要的包和依赖可以一键安装。

```
2 community/docker 1:18.09.6-1 (34.8 MiB 171.0 MiB) (Installed)
   Pack, ship and run any application as a lightweight container
1 community/container-diff 0.15.0-1 (2.8 MiB 10.4 MiB)
   Diff your Docker containers
==> Packages to install (eg: 1 2 3, 1-3 or ^4)
==> █
```

图 5.1 yay 安装相关包

- 2) 启动服务程序

```
hover@wings ~/Desktop/Labs/HUST_IOTStorage_Labs master ● systemctl start docker
```

图 5.2 启动 Docker 服务程序

由于 Docker 服务程序的性能占用，故采用 start，若希望开机启动，可以使用 enable

- 3) 运行相应的 Docker 环境

- a) Swift

华中科技大学课程实验报告

采用 swift all in one 进行启动, 由于 arch 系的支持原因, Open Stack 系列并未支持 swift, 导致无官方包和相应 AUR。

```
hover@wings: ~/Desktop/Labs/HUST_IOTStorage_Labs [master] docker run -d --name openstack-swift -p 12345:8080 --volumes-from SWIFT_DATA -t openstack-swift-docker  
aa3d244e8bb629c12648e35deb7f92b6c0b14005394105c452544455d932e8be
```

图 5.3 swift docker run

b) Cosbench

由于 Cosbench 存在较严重的问题, 不同的 docker 进行了尝试, 也尝试对于 docker 内的问题进行修复。

其中:

i. Cosbench 启动失败

在不同的环境上均遇到了此问题, 推测 JAVA Version 相关, docker 内出现符号链接错误。

ii. 特定的 workload 启动失败

在 8 线程的时候出现失败, 本地为四核八线程 CPU, 但 docker 中详细的设定可能会有差异, 推测与 docker 有关。

5.2 基于 Cosbench 进行单变量测试

5.2.1 Worker 数量影响

ID	Name	Works	Workers	Op-Info
w11-s1-w(1)KB_c1_init_1	w(1)KB_c1_init_1	1 wks	1 wkrs	init
w11-s2-w(1)KB_c1_o100_prepare_100	w(1)KB_c1_o100_prepare_100	1 wks	100 wkrs	prepare
w11-s3-w(1)KB_c1_o100_r80w15d5_1	w(1)KB_c1_o100_r80w15d5_1	1 wks	1 wkrs	read, write, delete
w11-s4-w(1)KB_c1_o100_r80w15d5_2	w(1)KB_c1_o100_r80w15d5_2	1 wks	2 wkrs	read, write, delete
w11-s5-w(1)KB_c1_o100_r80w15d5_4	w(1)KB_c1_o100_r80w15d5_4	1 wks	4 wkrs	read, write, delete
w11-s6-w(1)KB_c1_o100_r80w15d5_16	w(1)KB_c1_o100_r80w15d5_16	1 wks	16 wkrs	read, write, delete
w11-s7-w(1)KB_c1_o100_r80w15d5_32	w(1)KB_c1_o100_r80w15d5_32	1 wks	32 wkrs	read, write, delete
w11-s8-w(1)KB_c1_o100_r80w15d5_64	w(1)KB_c1_o100_r80w15d5_64	1 wks	64 wkrs	read, write, delete
w11-s9-w(1)KB_c1_o100_cleanup_1	w(1)KB_c1_o100_cleanup_1	1 wks	1 wkrs	cleanup
w11-s10-w(1)KB_c1_dispose_1	w(1)KB_c1_dispose_1	1 wks	1 wkrs	dispose

华中科技大学课程实验报告

图 5.4 workload 配置-Workers 变量

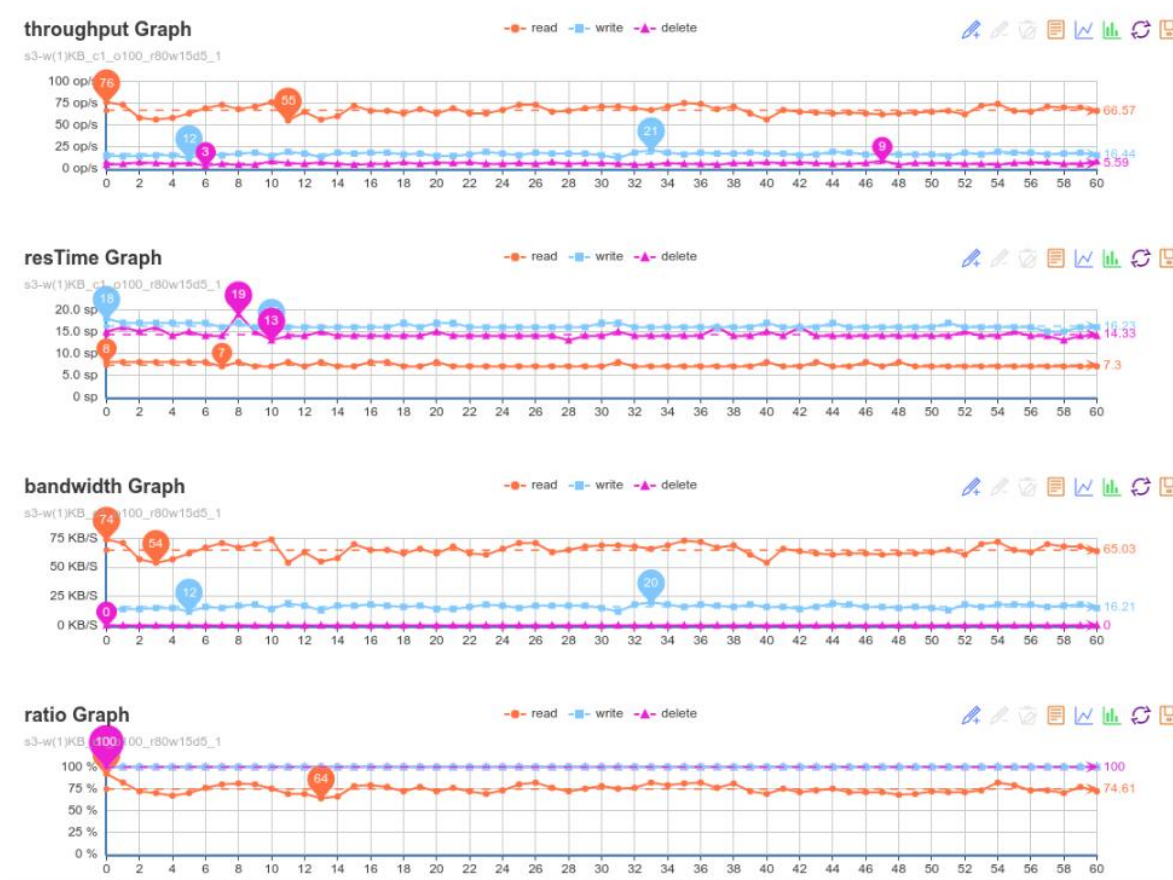


图 5.5 workers 变量改变结果

可以看到，在 workers 数量提升之后，整个曲线更加的平滑，同时各项指标有了相应的提升，读写吞吐率和带宽等都有明显增加，但是限于硬件的支持，多 worker 的上限与 CPU 线程数有关，并发度不可能无限提高，同时对于不同大小的文件，需要考虑创建/销毁 worker，以及相应的调度开销。

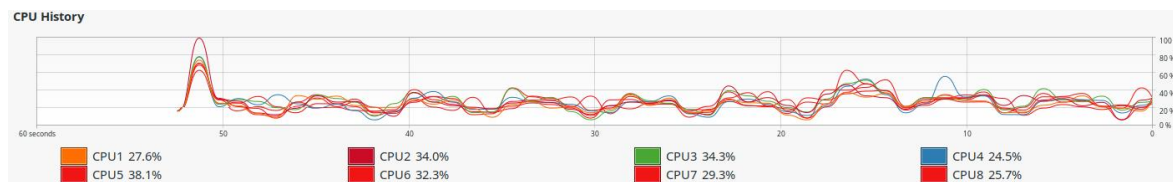


图 5.6 CPU 利用率曲线

此时可以看到，相对于通常的一核有难，七核围观的情况，核心之间的负载均衡做的较好。

华中科技大学课程实验报告

5.2.2 OBJ 大小测试

ID	Name	Works	Workers	Op-Info
w16-s1-w(1)KB_c1_init_1	w(1)KB_c1_init_1	1 wks	1 wkrs	init
w16-s2-w(2)KB_c1_init_1	w(2)KB_c1_init_1	1 wks	1 wkrs	init
w16-s3-w(4)KB_c1_init_1	w(4)KB_c1_init_1	1 wks	1 wkrs	init
w16-s4-w(8)KB_c1_init_1	w(8)KB_c1_init_1	1 wks	1 wkrs	init
w16-s5-w(16)KB_c1_init_1	w(16)KB_c1_init_1	1 wks	1 wkrs	init
w16-s6-w(32)KB_c1_init_1	w(32)KB_c1_init_1	1 wks	1 wkrs	init
w16-s7-w(64)KB_c1_init_1	w(64)KB_c1_init_1	1 wks	1 wkrs	init
w16-s8-w(128)KB_c1_init_1	w(128)KB_c1_init_1	1 wks	1 wkrs	init
w16-s9-w(256)KB_c1_init_1	w(256)KB_c1_init_1	1 wks	1 wkrs	init
w16-s10-w(1)KB_c1_o1_prepare_1	w(1)KB_c1_o1_prepare_1	1 wks	1 wkrs	prepare
w16-s11-w(2)KB_c1_o1_prepare_1	w(2)KB_c1_o1_prepare_1	1 wks	1 wkrs	prepare
w16-s12-w(4)KB_c1_o1_prepare_1	w(4)KB_c1_o1_prepare_1	1 wks	1 wkrs	prepare
w16-s13-w(8)KB_c1_o1_prepare_1	w(8)KB_c1_o1_prepare_1	1 wks	1 wkrs	prepare
w16-s14-w(16)KB_c1_o1_prepare_1	w(16)KB_c1_o1_prepare_1	1 wks	1 wkrs	prepare
w16-s15-w(32)KB_c1_o1_prepare_1	w(32)KB_c1_o1_prepare_1	1 wks	1 wkrs	prepare
w16-s16-w(64)KB_c1_o1_prepare_1	w(64)KB_c1_o1_prepare_1	1 wks	1 wkrs	prepare
w16-s17-w(128)KB_c1_o1_prepare_1	w(128)KB_c1_o1_prepare_1	1 wks	1 wkrs	prepare
w16-s18-w(256)KB_c1_o1_prepare_1	w(256)KB_c1_o1_prepare_1	1 wks	1 wkrs	prepare
w16-s19-w(1)KB_c1_o1_r80w15d5_8	w(1)KB_c1_o1_r80w15d5_8	1 wks	8 wkrs	read, write, delete
w16-s20-w(2)KB_c1_o1_r80w15d5_8	w(2)KB_c1_o1_r80w15d5_8	1 wks	8 wkrs	read, write, delete
w16-s21-w(4)KB_c1_o1_r80w15d5_8	w(4)KB_c1_o1_r80w15d5_8	1 wks	8 wkrs	read, write, delete
w16-s22-w(8)KB_c1_o1_r80w15d5_8	w(8)KB_c1_o1_r80w15d5_8	1 wks	8 wkrs	read, write, delete
w16-s23-w(16)KB_c1_o1_r80w15d5_8	w(16)KB_c1_o1_r80w15d5_8	1 wks	8 wkrs	read, write, delete
w16-s24-w(32)KB_c1_o1_r80w15d5_8	w(32)KB_c1_o1_r80w15d5_8	1 wks	8 wkrs	read, write, delete
w16-s25-w(64)KB_c1_o1_r80w15d5_8	w(64)KB_c1_o1_r80w15d5_8	1 wks	8 wkrs	read, write, delete
w16-s26-w(128)KB_c1_o1_r80w15d5_8	w(128)KB_c1_o1_r80w15d5_8	1 wks	8 wkrs	read, write, delete
w16-s27-w(256)KB_c1_o1_r80w15d5_8	w(256)KB_c1_o1_r80w15d5_8	1 wks	8 wkrs	read, write, delete
w16-s28-w(1)KB_c1_o1_cleanup_1	w(1)KB_c1_o1_cleanup_1	1 wks	1 wkrs	cleanup
w16-s29-w(2)KB_c1_o1_cleanup_1	w(2)KB_c1_o1_cleanup_1	1 wks	1 wkrs	cleanup
w16-s30-w(4)KB_c1_o1_cleanup_1	w(4)KB_c1_o1_cleanup_1	1 wks	1 wkrs	cleanup
w16-s31-w(8)KB_c1_o1_cleanup_1	w(8)KB_c1_o1_cleanup_1	1 wks	1 wkrs	cleanup
w16-s32-w(16)KB_c1_o1_cleanup_1	w(16)KB_c1_o1_cleanup_1	1 wks	1 wkrs	cleanup
w16-s33-w(32)KB_c1_o1_cleanup_1	w(32)KB_c1_o1_cleanup_1	1 wks	1 wkrs	cleanup
w16-s34-w(64)KB_c1_o1_cleanup_1	w(64)KB_c1_o1_cleanup_1	1 wks	1 wkrs	cleanup
w16-s35-w(128)KB_c1_o1_cleanup_1	w(128)KB_c1_o1_cleanup_1	1 wks	1 wkrs	cleanup
w16-s36-w(256)KB_c1_o1_cleanup_1	w(256)KB_c1_o1_cleanup_1	1 wks	1 wkrs	cleanup
w16-s37-w(1)KB_c1_dispose_1	w(1)KB_c1_dispose_1	1 wks	1 wkrs	dispose
w16-s38-w(2)KB_c1_dispose_1	w(2)KB_c1_dispose_1	1 wks	1 wkrs	dispose
w16-s39-w(4)KB_c1_dispose_1	w(4)KB_c1_dispose_1	1 wks	1 wkrs	dispose
w16-s40-w(8)KB_c1_dispose_1	w(8)KB_c1_dispose_1	1 wks	1 wkrs	dispose
w16-s41-w(16)KB_c1_dispose_1	w(16)KB_c1_dispose_1	1 wks	1 wkrs	dispose
w16-s42-w(32)KB_c1_dispose_1	w(32)KB_c1_dispose_1	1 wks	1 wkrs	dispose
w16-s43-w(64)KB_c1_dispose_1	w(64)KB_c1_dispose_1	1 wks	1 wkrs	dispose
w16-s44-w(128)KB_c1_dispose_1	w(128)KB_c1_dispose_1	1 wks	1 wkrs	dispose
w16-s45-w(256)KB_c1_dispose_1	w(256)KB_c1_dispose_1	1 wks	1 wkrs	dispose

图 5.7 OBJ Size 测试

华中科技大学课程实验报告

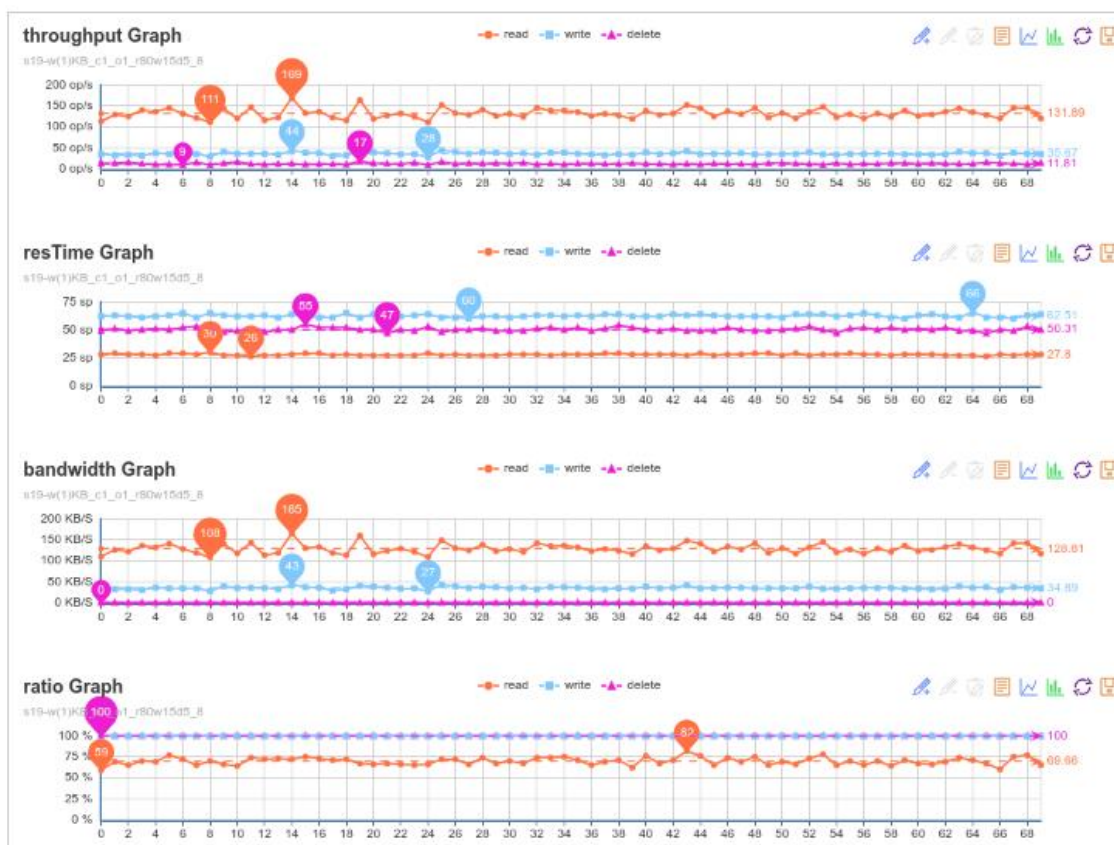


图 5.8 OBJ Size 测试图

当对象大小逐渐增大时，各项性能变优，判断 OBS 对于大文件的支持较好，基于此特性，在进行对象传输的过程中，存在相应的封装，对于 minio 等较为轻型的 OBS，未提供或者封装较为简单，很多情况下传输的为裸文件。而 Swift 存在相应封装，对于小 obj 传输，其寻址，读取开销，封装开销严重影响吞吐率和带宽。同时，此时并未考虑相应的加解密开销，如果指定相应的加解密算法，差距将进一步拉大。

华中科技大学课程实验报告

5.2.3 Container 数量测试

ID	Name	Works	Workers	Op-Info
w17-s1-w(1)KB_c1_init_1	w(1)KB_c1_init_1	1 wks	1 wkrs	init
w17-s2-w(1)KB_c2_init_1	w(1)KB_c2_init_1	1 wks	1 wkrs	init
w17-s3-w(1)KB_c4_init_1	w(1)KB_c4_init_1	1 wks	1 wkrs	init
w17-s4-w(1)KB_c8_init_1	w(1)KB_c8_init_1	1 wks	1 wkrs	init
w17-s5-w(1)KB_c1_o1_prepare_1	w(1)KB_c1_o1_prepare_1	1 wks	1 wkrs	prepare
w17-s6-w(1)KB_c2_o1_prepare_1	w(1)KB_c2_o1_prepare_1	1 wks	1 wkrs	prepare
w17-s7-w(1)KB_c4_o1_prepare_1	w(1)KB_c4_o1_prepare_1	1 wks	1 wkrs	prepare
w17-s8-w(1)KB_c8_o1_prepare_1	w(1)KB_c8_o1_prepare_1	1 wks	1 wkrs	prepare
w17-s9-w(1)KB_c1_o1_r80w15d5_8	w(1)KB_c1_o1_r80w15d5_8	1 wks	8 wkrs	read, write, delete
w17-s10-w(1)KB_c2_o1_r80w15d5_8	w(1)KB_c2_o1_r80w15d5_8	1 wks	8 wkrs	read, write, delete
w17-s11-w(1)KB_c4_o1_r80w15d5_8	w(1)KB_c4_o1_r80w15d5_8	1 wks	8 wkrs	read, write, delete
w17-s12-w(1)KB_c8_o1_r80w15d5_8	w(1)KB_c8_o1_r80w15d5_8	1 wks	8 wkrs	read, write, delete
w17-s13-w(1)KB_c1_o1_cleanup_1	w(1)KB_c1_o1_cleanup_1	1 wks	1 wkrs	cleanup
w17-s14-w(1)KB_c2_o1_cleanup_1	w(1)KB_c2_o1_cleanup_1	1 wks	1 wkrs	cleanup
w17-s15-w(1)KB_c4_o1_cleanup_1	w(1)KB_c4_o1_cleanup_1	1 wks	1 wkrs	cleanup
w17-s16-w(1)KB_c8_o1_cleanup_1	w(1)KB_c8_o1_cleanup_1	1 wks	1 wkrs	cleanup
w17-s17-w(1)KB_c1_dispose_1	w(1)KB_c1_dispose_1	1 wks	1 wkrs	dispose
w17-s18-w(1)KB_c2_dispose_1	w(1)KB_c2_dispose_1	1 wks	1 wkrs	dispose
w17-s19-w(1)KB_c4_dispose_1	w(1)KB_c4_dispose_1	1 wks	1 wkrs	dispose
w17-s20-w(1)KB_c8_dispose_1	w(1)KB_c8_dispose_1	1 wks	1 wkrs	dispose

图 5.9 Container Number 测试

Container Number 增加时，各项数据变化不明显，由于 Container 之间可以做冗余或者多节点存储，同步开销将成为主要的部分，此处查阅文档得知，Swift 支持定期同步，且可以设置同步授权，此时，Container 数量将会影响到多节点同步。

5.3 进行 HDD/SSD 性能比较

由于本机采用 SSD，而 NAS 之上采用 HDD,在实际过程中，对于非时间敏感型数据的读取通常采用 HDD 存储，Swift docker 形式封装不易指定相应的存储位置，采用 Minio 在不同分的地方启动服务，探究 HDD/SSD 的性能差异。

华中科技大学课程实验报告

5.3.1 SSD 测试

Current Stage	Stages completed	Stages remaining	Start Time	End Time	Time Remaining	
ID	Name	Works	Workers	Op-Info	State	Link
w1-s1-init	init	1 wks	1 wkrs	init	completed	view details
w1-s2-prepare	prepare	8 wks	64 wkrs	prepare	completed	view details
w1-s3-8kb	8kb	1 wks	8 wkrs	read, write	completed	view details
w1-s4-16kb	16kb	1 wks	8 wkrs	read, write	completed	view details
w1-s5-32kb	32kb	1 wks	4 wkrs	read, write	completed	view details
w1-s6-64kb	64kb	1 wks	4 wkrs	read, write	completed	view details
w1-s7-128kb	128kb	1 wks	1 wkrs	read, write	completed	view details
w1-s8-256kb	256kb	1 wks	1 wkrs	read, write	completed	view details
w1-s9-512kb	512kb	1 wks	1 wkrs	read, write	completed	view details
w1-s10-1mb	1mb	1 wks	1 wkrs	read, write	completed	view details
w1-s11-cleanup	cleanup	1 wks	1 wkrs	cleanup	completed	view details
w1-s12-dispose	dispose	1 wks	1 wkrs	dispose	completed	view details

图 5.10 测试配置

General Report

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
init-write	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
prepare-write	8 ops	64 KB	136.38 ms	135.62 ms	56.03 op/s	448.21 KB/S	100%
prepare-write	8 ops	128 KB	140.25 ms	139.38 ms	59.78 op/s	956.55 KB/S	100%
prepare-write	8 ops	256 KB	149.75 ms	139.62 ms	54.12 op/s	1.73 MB/S	100%
prepare-write	8 ops	512 KB	146 ms	136.38 ms	56.38 op/s	3.61 MB/S	100%
prepare-write	8 ops	1.02 MB	151.38 ms	131.25 ms	55.08 op/s	7.05 MB/S	100%
prepare-write	8 ops	2.05 MB	179 ms	130.88 ms	43.35 op/s	11.1 MB/S	100%
prepare-write	8 ops	4.1 MB	188.5 ms	100 ms	42.77 op/s	21.9 MB/S	100%
prepare-write	8 ops	8 MB	209.12 ms	116.75 ms	38.2 op/s	38.2 MB/S	100%
read	28.93 kops	231.41 MB	1.87 ms	1.86 ms	964.52 op/s	7.72 MB/S	100%
write	7.28 kops	58.25 MB	22.92 ms	22.92 ms	242.78 op/s	1.94 MB/S	100%
read	29.62 kops	473.87 MB	1.78 ms	1.78 ms	987.38 op/s	15.8 MB/S	100%
write	7.46 kops	119.34 MB	22.23 ms	22.23 ms	248.67 op/s	3.98 MB/S	100%
read	18.37 kops	587.9 MB	2.02 ms	2.02 ms	612.66 op/s	19.61 MB/S	100%
write	4.56 kops	145.76 MB	15.26 ms	15.25 ms	151.9 op/s	4.86 MB/S	100%
read	17.8 kops	1.14 GB	2.29 ms	2.28 ms	593.36 op/s	37.97 MB/S	100%
write	4.4 kops	281.79 MB	15.31 ms	15.29 ms	146.8 op/s	9.4 MB/S	100%
read	5.23 kops	668.93 MB	2.59 ms	2.23 ms	174.21 op/s	22.3 MB/S	100%
write	1.29 kops	164.61 MB	10.12 ms	9.54 ms	42.87 op/s	5.49 MB/S	100%
read	4.52 kops	1.16 GB	3.23 ms	2.21 ms	150.72 op/s	38.58 MB/S	100%
write	1.14 kops	290.82 MB	10.81 ms	9.27 ms	37.87 op/s	9.69 MB/S	100%
read	3.85 kops	1.97 GB	4.12 ms	2.16 ms	128.25 op/s	65.66 MB/S	100%
write	929 ops	475.65 MB	12.42 ms	9.74 ms	30.97 op/s	15.86 MB/S	100%
read	3.11 kops	3.11 GB	5.45 ms	2.1 ms	103.55 op/s	103.55 MB/S	100%
write	788 ops	788 MB	13.87 ms	9.51 ms	26.27 op/s	26.27 MB/S	100%
cleanup-delete	128 ops	0 B	1.77 ms	1.77 ms	463.77 op/s	0 B/S	100%
dispose-delete	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A

图 5.11 SSD 测试结果

华中科技大学课程实验报告

5.3.2 HDD 测试

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
init-write	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
prepare-write	8 ops	64 KB	2185.88 ms	2185.88 ms	3.67 op/s	29.34 KB/S	100%
prepare-write	8 ops	128 KB	2113.88 ms	2113.88 ms	3.8 op/s	60.83 KB/S	100%
prepare-write	8 ops	256 KB	961.38 ms	961.25 ms	8.33 op/s	266.51 KB/S	100%
prepare-write	8 ops	512 KB	1997 ms	1996.62 ms	4.11 op/s	263.02 KB/S	100%
prepare-write	8 ops	1.02 MB	2150.38 ms	2150.38 ms	3.73 op/s	477.36 KB/S	100%
prepare-write	8 ops	2.05 MB	2258.38 ms	2257.5 ms	3.55 op/s	908.99 KB/S	100%
prepare-write	8 ops	4.1 MB	2299.12 ms	2292 ms	3.48 op/s	1.78 MB/S	100%
prepare-write	8 ops	8 MB	2347.38 ms	2338.12 ms	3.41 op/s	3.41 MB/S	100%
read	6.83 kops	54.67 MB	20.1 ms	20.09 ms	227.89 op/s	1.82 MB/S	100%
write	1.73 kops	13.82 MB	56.83 ms	56.83 ms	57.62 op/s	460.98 KB/S	100%
read	5.94 kops	94.96 MB	23.96 ms	23.96 ms	197.87 op/s	3.17 MB/S	100%
write	1.39 kops	22.18 MB	67.79 ms	67.79 ms	46.21 op/s	739.35 KB/S	100%
read	6.57 kops	210.11 MB	9.86 ms	9.86 ms	218.96 op/s	7.01 MB/S	100%
write	1.6 kops	51.04 MB	31.83 ms	31.83 ms	53.19 op/s	1.7 MB/S	100%
read	6.71 kops	429.31 MB	9.35 ms	9.35 ms	224.78 op/s	14.39 MB/S	100%
write	1.65 kops	105.34 MB	31.63 ms	31.63 ms	55.16 op/s	3.53 MB/S	100%
read	3.82 kops	489.22 MB	2.48 ms	2.47 ms	127.42 op/s	16.31 MB/S	100%
write	1 kops	128.64 MB	17.69 ms	17.68 ms	33.51 op/s	4.29 MB/S	100%
read	2.9 kops	742.14 MB	3.2 ms	2.88 ms	96.65 op/s	24.74 MB/S	100%
write	738 ops	188.93 MB	25.76 ms	24.77 ms	24.6 op/s	6.3 MB/S	100%
read	2.3 kops	1.18 GB	4.17 ms	2.91 ms	76.82 op/s	39.33 MB/S	100%
write	581 ops	297.47 MB	32.76 ms	30.73 ms	19.38 op/s	9.92 MB/S	100%
read	1.48 kops	1.48 GB	6.13 ms	2.98 ms	49.59 op/s	49.59 MB/S	100%
write	376 ops	376 MB	53.17 ms	48.83 ms	12.56 op/s	12.56 MB/S	100%
cleanup-delete	128 ops	0 B	1.79 ms	1.79 ms	449.12 op/s	0 B/S	100%
dispose-delete	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A

由上表可以看出，在读取方面，约有五倍的性能提升，在写方面约 5 倍的性能提升，同时在准备的时间大大缩短。而在较大文件的读取时，退化到约 2 倍，可以看到在顺序读取时 SSD 的性能优势不如随机读取时明显，同时 worker 数量对于 SSD 的性能提升远远大于 HDD,限于 HDD 的寻道操作，并行提升并不明显，更适用于顺序读取。

6 实验总结

在实验过程中,环境配置可以说相对来说比较简单,但是某些玄学 BUG 可能无明显报错信息,转而转向 Docker,由于所有的包括服务端和测试端均在 Docker 中启动,存在部分 Automated Docker 入口写死,需要自己手动更改 Docker 文件进行相应的编译,在此过程中提高了 Docker 的熟练程度。

实验过程中也体会到了实际工业环境与 Toy 之间的区别,对于 swift,通常采用 Obj Server,Container Server,Account Server 三个服务器进行分别管理,处理调度分配,实际工作执行。而 Docker 封装 all in one 则仅仅使用端口号进行区分,实际上存在 Container Server,Account Server 等多个 docker 以进行配合,且单机模式无法判断带宽影响,同 hadoop 等分布式平台类似,大多数实验环境基于本地虚拟机,在云服务平台上,大多采用的是硬件虚拟化之后再使用多个 docker 模拟集群,与实际的物理存储环境可能存在差异。

测试方面,由于常规使用本地 NAS 做存储,使用花生壳做内网穿透,故相应的测试也在本地进行,并进行了 HDD/SSD 进行相应的比较。

最后,感谢实验过程中提供帮助的老师 and 同学以及所参阅资料的提供者。

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,

华 中 科 技 大 学 课 程 实 验 报 告

USA: USENIX Association, 2006: 307–320.

[4] Swift Document. URL: <https://docs.openstack.org/swift>