



2016 级

《物联网数据存储与管理》课程

实 验 报 告

姓 名 潘 越

学 号 U201614897

班 号 物联网 1601 班

日 期 2018.05.27

目 录

一、实验目的	1
二、实验背景	1
三、实验环境	2
3.1 实验环境一览	2
3.2 实验环境结构	2
四、实验内容	3
4.1 对象存储技术实践	3
4.2 对象存储性能分析	3
五、实验过程	3
5.1 实验环境配置	4
5.2 服务端——Openstack Swift 配置	4
5.3 评测工具——COSBench 部署	7
5.4 存储性能实验与分析	8
六、实验总结	12
参考文献	12

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

当涉及到在过去的十年里的信息技术时，如果说存在一个普遍性的共识的话，那就是：数据的增长是不可避免，不可阻挡的！无论是由个人或专业动机的驱动，我们每个个体所创造出的数字资产比以往任何时候都要多得多。无论是任何行业，当前企业组织的成功运营均取决于其利用数字化资产的能力。无论是充分利用更高分辨率的视频数据的媒体和娱乐业界或是对数字化的影响进行更现实的开发，捕获详细的 3D 或 4D 地震数据的能源勘探企业；又或是安全系统公司捕捉高分辨率的安全性素材；还是在线内容分发、创作行业。对于几乎每一个行业，有效利用数字资产对于保持其市场竞争力都是至关重要的。

例如，在我们身边，QQ 的月活跃用户量大于 8 亿，微信的月活跃用户量大于 6 亿，QQ 空间相册的图片大于 4000 亿张，而微信朋友圈的图片日上传量也达到了 10 亿张，腾讯的社交、游戏，访问密度高达 100 万次/秒/100GB 量级的数据读写，等等……但是，在如此海量数据的背景下，在线业务还应保证良好的用户体验，不论数据访问密集程度如何，均要求延迟在 100 毫秒以内。那么，在应对这种情况的时候，一种先进的技能存储海量数据又具有高性能高可用性的存储技术是我们迫切需要的，这就加速了对象存储技术的发展。

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。其对应存储和保护大量非结构化的数据所带来的挑战提供了一个直接的响应。

对象存储综合了 NAS 和 SAN 的优点，同时具有 SAN 的高速直接访问和 NAS 的分布式数据共享等优势，提供了具有高性能、高可靠性、跨平台以及安全的数据共享的存储体系结构，非常适用于现在的海量数据的场景。

而在物联网方面，数据则存在着两大特性。一是数据的规模庞大，数据规模持续扩张；二是数据结构复杂，内容丰富，各种物品均携带自己的信息，这些信息上传到网络后，错综复杂的数据也带给管理人员很多管理上的难题。

本实验基于以上背景，探究在物联网数据存储上可以使用的对象存储技术，自己搭建环境被配置对象存储服务端与客户端，并对自己的系统进行评测，以此来学习和实践课堂中学习的相关知识，加深理解。

三、实验环境

3.1 实验环境一览

实验平台	环境与版本
本地环境操作系统	Arch Linux Kernel version 5.1.4
本地硬件配置	Intel(R) Core(TM) i5-8250 CPU @ 1.60GHz / 8GB
云服务器操作系统	Ubuntu 16.04 Linux Kernel version 4.4.0 / 2GB
对象存储服务端	Openstack Swift (Docker version 18.09.6-ce)
对象存储客户端	python-swiftclient version 3.7.0-1
对象存储评测工具	COSBench version 0.4.2.c3

3.2 实验环境结构

本实验的实验环境全部配置在云服务器上，将多个服务启动在不同的端口，通过本地浏览器来观察实验评测结果。

对象存储服务端采用 Openstack Swift 实现，环境配置在一个 Docker 中，启动端口为 8080，并在 Docker 实例启动时将 8080 端口映射到 12345 端口，测试客户端选用 Arch 包管理中自带的 python-swiftclient，可以通过本地命令执行对象存储测试。

大规模数据评测工具选用 COSBench，其环境配置在云服务器，同时将打包好的 Openstack Swift 的 Docker 移植到云服务器上，方便配合测试。其中 COSBench 的主页开放端口为 19088，测试数据细节页面端口为 18088，在实验时在浏览器中输入服务器 IP 地址:端口号即可配置评测内容，查看评测工具的运行结果。

实验环境总体的结构图如下：

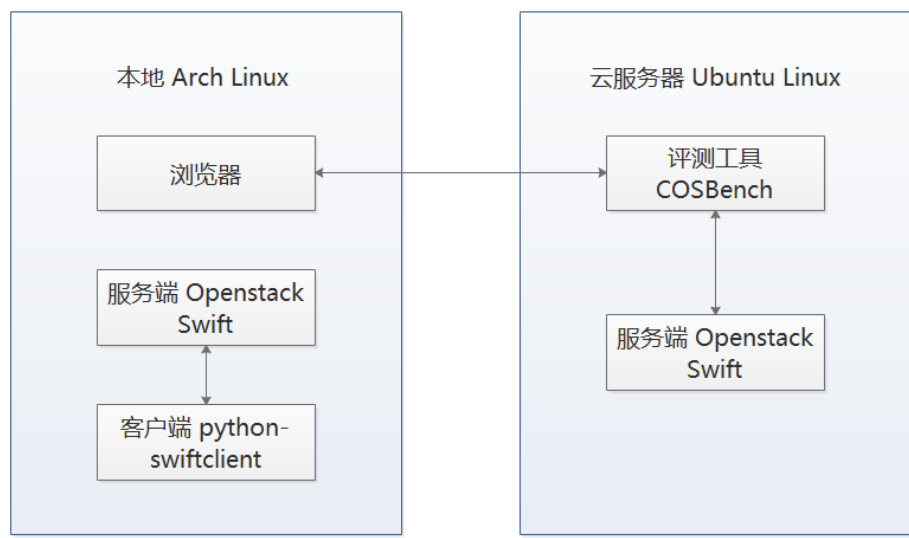


图 1 实验环境结构图

四、实验内容

本实验的主要内容有，熟悉实验环境并在本地配置环境，通过在本地配置对象存储服务端与客户端，来熟悉和尝试使用对象存储的技术，同时使用标准的评测工具来对实验环境进行评测，分析对象存储的性能，探索优化速率的方法。

4.1 对象存储技术实践

（1）在本地环境上安装 Docker，并在 Docker 上配置并部署一个单节点的 Openstack Swift 服务端。

（2）在本地环境上安装 python-swiftclient 客户端，并通过该客户端来对服务端进行功能上的测试。

4.2 对象存储性能分析

（1）将章节 4.1 中完成的 Docker 镜像移植到云服务器，做好端口映射便于测试。

（2）在云服务器上安装并部署 COSBench，在本地浏览器进行测试，确保在本地可以对云服务器上的环境进行配置和观测。

（3）编写测试文件，对实验环境进行测试，观察运行结果并分析对象存储性能的情况。

五、实验过程

5.1 实验环境配置

本次实验要求熟悉 Git 和 Linux 系统，以及配置环境。由于我使用 Arch Linux 系统、github 已经两年，对环境已经非常熟悉，因此基本不需要环境的配置。

分析后决定选用 Openstack Swift 作为服务端，同时发现 Arch 的包管理 pacman 中自带 Swift 的客户端，可以快速安装进行测试，如下图所示：

```
panyue@zxcryp ~$ pacman -Ss docker
community/container-diff 0.15.0-1
Diff your Docker containers
community/docker 1:18.09.6-1 [已安装]
Pack, ship and run any application as a lightweight container
community/docker-compose 1.24.0-1
Fast, isolated development environments using Docker
community/docker-machine 0.16.1-2
Machine management for a container-centric world
community/podman-docker 1.3.1-1
Emulate Docker CLI using podman
```

图 2 Docker 安装（已完成）

```
panyue@zxcryp ~$ pacman -Ss swift
community/python-swiftclient 3.7.0-1 [已安装]
An SDK for building applications to work with OpenStack
community/python2-swiftclient 3.7.0-1
An SDK for building applications to work with OpenStack
```

图 3 Python Swift Client 安装（已完成）

5.2 服务端——Openstack Swift 配置

实验二要求我们配置完整的对象存储服务端-客户端-评测工具的环境，分为以下三个步骤。

（1）服务端——Openstack Swift

首先克隆实验文档提供的 Github 仓库，执行如下命令：

```
git clone https://github.com/cs-course/openstack-swift-docker.git
```

接着启动 Docker 服务：

```
systemctl start docker
```

执行之后，使用命令 systemctl 查看已启动的服务，如下图：

```
accounts-daemon.service loaded active running Accounts Service
bluetooth.service loaded active running Bluetooth Service
colord.service loaded active running Manage, Install and Generate Color Profiles
dbus.service loaded active running D-Bus System Message Bus
docker.service loaded active running Docker Application Container Engine
gdm.service loaded active running GNOME Display Manager
geoclue.service loaded active running Location Lookup Service
haveged.service loaded active running Entropy Harvesting Daemon
lines 33-52
```

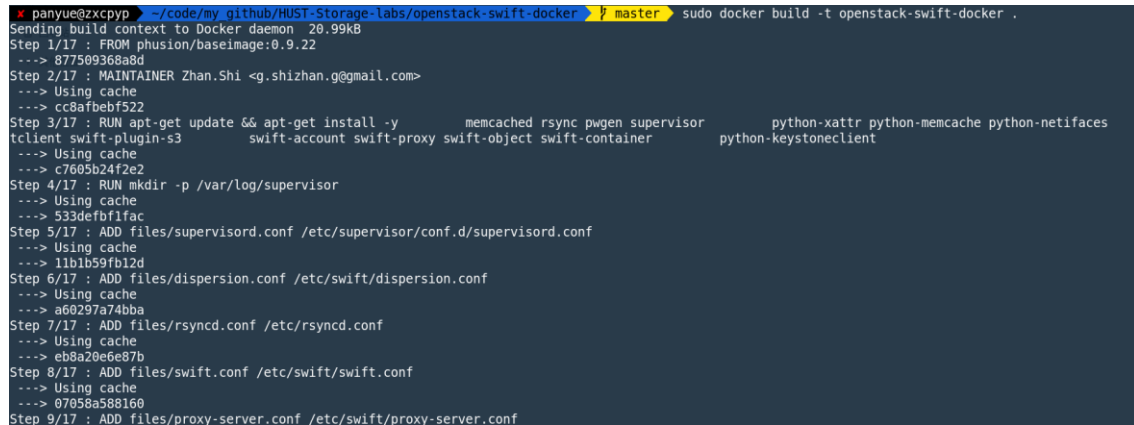
图 4 Docker 服务已启动

如图可以看到 docker.service 已经正常启动了，接下来可以开始构建 Openstack Swift 的容器。

接着执行以下命令构建 Docker 镜像：

`Sudo docker build -t openstack-swift-docker .`

结果如下图所示，系统根据 Dockerfile 中的步骤执行自动的环境配置与构建。



```

* panyue@zxcypyp ~/code/my_github/HUST-Storage-labs/openstack-swift-docker  master sudo docker build -t openstack-swift-docker .
Sending build context to Docker daemon 20.99kB
Step 1/17 : FROM phusion/baseimage:0.9.22
--> 877500368a8d
Step 2/17 : MAINTAINER Zhan.Shi <g.shizhan.g@gmail.com>
--> Using cache
--> cc8afbebf522
Step 3/17 : RUN apt-get update && apt-get install -y memcached rsync pwgen supervisor python-xattr python-memcache python-netifaces
            tcldclient swift-plugin-s3 swift-account swift-proxy swift-object swift-container python-keystoneclient
--> Using cache
--> c7605b24f2e2
Step 4/17 : RUN mkdir -p /var/log/supervisor
--> Using cache
--> 533defbf1fac
Step 5/17 : ADD files/supervisord.conf /etc/supervisor/conf.d/supervisord.conf
--> Using cache
--> 11b1b59fb12d
Step 6/17 : ADD files/dispersion.conf /etc/swift/dispersion.conf
--> Using cache
--> a60297a74bba
Step 7/17 : ADD files/rsyncd.conf /etc/rsyncd.conf
--> Using cache
--> eb8a20e6e87b
Step 8/17 : ADD files/swift.conf /etc/swift/swift.conf
--> Using cache
--> 07058a588160
Step 9/17 : ADD files/proxy-server.conf /etc/swift/proxy-server.conf
--> Using cache
--> 07058a588160

```

图 5 Docker 镜像定制

然后准备数据分卷：

`sudo docker run -v /srv --name SWIFT_DATA busybox`

此时本地环境已经准备完毕，接着配合客户端在本地测试服务端的情况。

首先启动一个容器实例，执行如下命令：

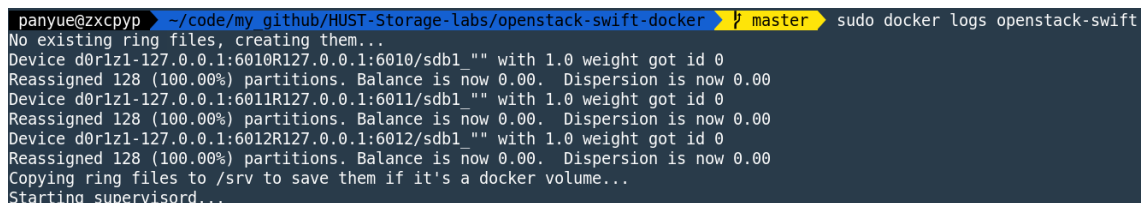
`sudo docker run -d --name openstack-swift -p 12345:8080 --volumes-from SWIFT_DATA -t openstack-swift-docker`

之后测试容器实例是否启动，执行如下两条命令，分别查看结果：

`sudo docker logs openstack-swift`

`sudo docker ps`

结果如下面两图所示：



```

* panyue@zxcypyp ~/code/my_github/HUST-Storage-labs/openstack-swift-docker  master sudo docker logs openstack-swift
No existing ring files, creating them...
Device d0r1z1-127.0.0.1:6010R127.0.0.1:6010/sdb1 "" with 1.0 weight got id 0
Reassigned 128 (100.00%) partitions. Balance is now 0.00. Dispersion is now 0.00
Device d0r1z1-127.0.0.1:6011R127.0.0.1:6011/sdb1 "" with 1.0 weight got id 0
Reassigned 128 (100.00%) partitions. Balance is now 0.00. Dispersion is now 0.00
Device d0r1z1-127.0.0.1:6012R127.0.0.1:6012/sdb1 "" with 1.0 weight got id 0
Reassigned 128 (100.00%) partitions. Balance is now 0.00. Dispersion is now 0.00
Copying ring files to /srv to save them if it's a docker volume...
Starting supervisord...

```

图 6 Docker log

panyue@zxcpp	~/code/my_github/HUST-Storage-labs/openstack-swift-docker	master	sudo docker ps			
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
3b83e418995e	openstack-swift-docker	"/bin/sh -c /usr/loc..."	40 seconds ago	Up 39 seconds	0.0.0.0:12345->8080/tcp	openstack-swift
panyue@zxcpp	~/code/my_github/HUST-Storage-labs/openstack-swift-docker	master				

图 7 Docker 运行的实例

从图 中可以看到我们 39 秒前启动的容器实例，为了验证服务端功能的正确性，我们使用 `python-swiftclient` 来对服务端进行测试。

执行以下命令：

```
sudo swift -A http://127.0.0.1:12345/auth/v1.0 -U test:tester -K testing stat
```

结果却遇到如下的错误：

```
/opt/az/lib/python3.6/site-packages/requests/__init__.py:91:
```

RequestsDependencyWarning: urllib3 (1.25.2) or chardet (3.0.4) doesn't match a supported version!

RequestsDependencyWarning)

经过我在网上查阅资料后，在 `github issue` 上查到了这个问题的解决方案，链接：<https://github.com/Azure/azure-cli/issues/9257>

通过修改 `python` 原生库的代码，在文件 `Python37\Lib\site-packages\requests` 中的第 63 行改为：

```
assert minor <= 25
```

即把原先的 24 改为 25 之后即可解决这个问题，之后再执行命令可以看到如下图所示的结果：

```
panyue@zxcpp ~/code/my_github/HUST-Storage-labs/openstack-swift-docker master sudo 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: 1559033428.50328
X-Put-Timestamp: 1559033428.50328
X-Trans-Id: txa2d1ed421b384ee8b9d6c-0055cccf654
```

图 8 Swift 客户端测试

如图 所示，可以看到我们设置的用户名 `test`（在配置文件中），以及其他的属性信息，接着我们测试一下单个文件的存储和下载。

首先执行如下命令，尝试把实验文档 `PPT` 上传进入服务端中，并且命名为 `test.pptx`：

```
sudo swift -A http://127.0.0.1:12345/auth/v1.0 -U test:tester -K testing upload
--object-name test.pptx SWIFT_DATA ./iot-storage-experiment.pptx
```

接着尝试从对象存储服务中下载这个文件，执行命令：


```
sudo swift -A http://127.0.0.1:12345/auth/v1.0 -U test:tester -K testing download  
SWIFT_DATA test.pptx
```

如下面两图：我们成功执行了上传和下载的流程，验证了程序的正确性。

```
panyue@zxcpp ~$ sudo swift -A http://127.0.0.1:12345/auth/v1.0 -U test:tester -K testing upload --object-  
name test.pptx SWIFT_DATA ./iot-storage-experiment.pptx  
test.pptx  
panyue@zxcpp ~$ sudo swift -A http://127.0.0.1:12345/auth/v1.0 -U test:tester -K testing download SWIFT_D  
ATA test.pptx  
test.pptx [auth 0.008s, headers 0.019s, total 0.050s, 202.825 MB/s]  
panyue@zxcpp ~$ ls  
iot-storage-experiment.pptx  report-template.doc  test.pptx
```

图 9 Swift 客户端上传和下载文件

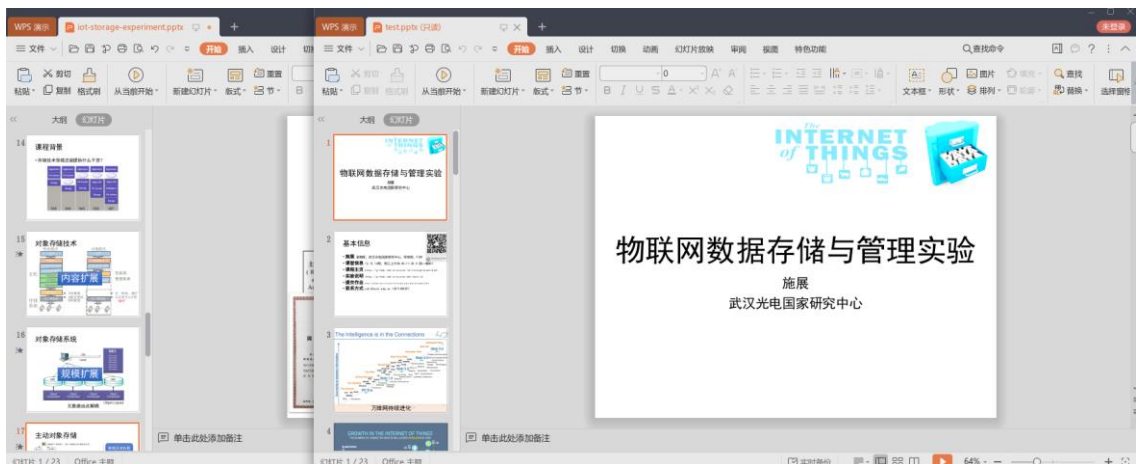


图 10 验证文件是同一个

5.3 评测工具——COSBench 部署

在 COSBench 的官方 Github 上的 release 里可以下载到 0.4.2.c4 版本，下载后解压，直接运行根目录下的 start-all.sh 即可启动服务。

```
panyue@SaltedfishAli ~$ cd /HUST-Storage-Lab/cos  
$ ls  
3rd-party-licenses.pdf  lib-src  start-controller.sh  
archive                 LICENSE  start-driver.bat  
BUILD.md                licenses start-driver.sh  
BUILD.no                log      stop  
CHANGELOG               main     stop-all.sh  
cli.sh                  NOTICE  stop-controller.sh  
conf                    osgi     stop-driver.sh  
COSBenchAdaptorDevGuide.pdf pkg.lst  TODO.md  
cosbench-start.sh       README.md VERSION  
cosbench-stop.sh        scripts web.bat  
COSBenchUserGuide.pdf  start   work  
ext                     start-all.bat workloads  
ip-port.list            start-all.sh workspace  
javadoc                 start-controller.bat
```

图 10 服务器端配置 COSBench

在这里要注意查看云服务器的后台是否开放 19088 端口和 18088 端口，这两个端口一个是 COSBench 主配置页面，一个是错误细节页面，如果没有开放端口则看不到相应的数据。

在 COSBench 中，可以设置评测环境，如下图，输可以配置自务器的认证信息，对象存储服务地址端口，然后可以在特定的 Workload 中输入指定的对象大小、

container 数量、Worker 的数量等。

COSBENCH - CONTROLLER WEB CONSOLEtime: Tue M

Add Init Stage

☒ Prepare Stage:

Worker Count	Container Selector	Object Selector	Size Selector
1	1 - 32	1 - 50	64 - 64 KB

☐ Delay:

Add Prepare Stage

☒ Main Stage:

Worker Count	Rampup Time (in second)	Runtime (in second)
8	10	30

Operation	Ratio	Container Selector	Object Selector	Size Selector	File selector
Read	80	1 - 32	1 - 50		
Write	20	1 - 32	51 - 10	64 - 64 KB	
File-Write	0	1 - 32			/tmp/testfiles/
Delete	0	1 - 10	1 - 10		

☐ Delay:

Add Main Stage

图 10 COSBench 配置评测界面

5.4 存储性能实验与分析

打开测试运行的分析，可以看到如下图所示，记录了该次测试的提交、准备和完成的时间。

Submitted At: 2019-5-22 11:36:42 Started At: 2019-5-22 11:36:42 Stopped At: 2019-5-22 11:59:11

State History

State	Date
queuing	2019-5-22 11:36:42
processing	2019-5-22 11:36:42
finished	2019-5-22 11:59:11

Time Remaining

图 11 测试记录

实验运行的结果如下图：

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
op1: init -write	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
op1: init -write	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
op1: init -write	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
op1: prepare -write	20 ops	80 KB	170.2 ms	170.15 ms	58.86 op/s	235.46 KB/S	100%
op1: prepare -write	20 ops	2.56 MB	8259.6 ms	5707.2 ms	1.23 op/s	157.33 KB/S	100%
op1: prepare -write	20 ops	10.24 MB	33482.9 ms	7074.65 ms	0.3 op/s	155.78 KB/S	100%
op1: read	6.07 kops	24.28 MB	35.56 ms	15.29 ms	20.25 op/s	80.99 KB/S	76.28%
op2: write	1.49 kops	5.96 MB	38.25 ms	38.23 ms	4.97 op/s	19.86 KB/S	100%
op3: delete	465 ops	0 B	17.94 ms	17.94 ms	1.55 op/s	0 B/S	100%
op1: read	224 ops	28.67 MB	1078.17 ms	14.96 ms	0.75 op/s	96.28 KB/S	82.35%
op2: write	48 ops	6.14 MB	1156.19 ms	1156.04 ms	0.16 op/s	20.63 KB/S	100%
op3: delete	18 ops	0 B	18.72 ms	18.72 ms	0.06 op/s	0 B/S	100%
op1: read	61 ops	31.23 MB	4026.44 ms	15.59 ms	0.21 op/s	105.69 KB/S	70.11%
op2: write	12 ops	6.14 MB	4120.42 ms	1000.42 ms	0.04 op/s	20.79 KB/S	100%
op3: delete	7 ops	0 B	19.86 ms	19.86 ms	0.02 op/s	0 B/S	100%
op1: cleanup -delete	20 ops	0 B	19.9 ms	19.9 ms	50.25 op/s	0 B/S	100%
op1: cleanup -delete	20 ops	0 B	18.4 ms	18.4 ms	54.2 op/s	0 B/S	100%
op1: cleanup -delete	20 ops	0 B	17.25 ms	17.25 ms	57.64 op/s	0 B/S	100%
op1: dispose -delete	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
op1: dispose -delete	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A
op1: dispose -delete	0 ops	0 B	N/A	N/A	0 op/s	0 B/S	N/A

图 12 运行结果统计

Stages

Current Stage	Stages completed	Stages remaining	Start Time	End Time	Time Remaining	
ID	Name	Works	Workers	Op-Info	State	Link
w6-s1-w(4)KB_c2_init_1	w(4)KB_c2_init_1	1 wks	1 wkrs	init	completed	view details
w6-s2-w(128)KB_c2_init_1	w(128)KB_c2_init_1	1 wks	1 wkrs	init	completed	view details
w6-s3-w(512)KB_c2_init_1	w(512)KB_c2_init_1	1 wks	1 wkrs	init	completed	view details
w6-s4-w(4)KB_c2_o10_prepare_10	w(4)KB_c2_o10_prepare_10	1 wks	10 wkrs	prepare	completed	view details
w6-s5-w(128)KB_c2_o10_prepare_10	w(128)KB_c2_o10_prepare_10	1 wks	10 wkrs	prepare	completed	view details
w6-s6-w(512)KB_c2_o10_prepare_10	w(512)KB_c2_o10_prepare_10	1 wks	10 wkrs	prepare	completed	view details
w6-s7-w(4)KB_c2_o10_r80w15d5_1	w(4)KB_c2_o10_r80w15d5_1	1 wks	1 wkrs	read, write, delete	completed	view details
w6-s8-w(128)KB_c2_o10_r80w15d5_1	w(128)KB_c2_o10_r80w15d5_1	1 wks	1 wkrs	read, write, delete	completed	view details
w6-s9-w(512)KB_c2_o10_r80w15d5_1	w(512)KB_c2_o10_r80w15d5_1	1 wks	1 wkrs	read, write, delete	completed	view details
w6-s10-w(4)KB_c2_o10_cleanup_1	w(4)KB_c2_o10_cleanup_1	1 wks	1 wkrs	cleanup	completed	view details
w6-s11-w(128)KB_c2_o10_cleanup_1	w(128)KB_c2_o10_cleanup_1	1 wks	1 wkrs	cleanup	completed	view details
w6-s12-w(512)KB_c2_o10_cleanup_1	w(512)KB_c2_o10_cleanup_1	1 wks	1 wkrs	cleanup	completed	view details
w6-s13-w(4)KB_c2_dispose_1	w(4)KB_c2_dispose_1	1 wks	1 wkrs	dispose	completed	view details
w6-s14-w(128)KB_c2_dispose_1	w(128)KB_c2_dispose_1	1 wks	1 wkrs	dispose	completed	view details
w6-s15-w(512)KB_c2_dispose_1	w(512)KB_c2_dispose_1	1 wks	1 wkrs	dispose	completed	view details

图 13 实验运行步骤

从图 13 中可以看到整个评测执行的流程，三组初始化，三组准备，三组读写测试，三组清除测试，三组部署。

分析图 12 的运行结果，我们可以看到，当 Worker 增加时，读写操作产生的字节总数、处理的平均时间、吞吐率和带宽等都有明显增加；当 Container 增加时，数据的变化不是特别明显；当对象大小逐渐增大时，产生的字节数和带宽有明显的增大。而在读、写和删除三种操作中，读操作的成功率并没有达到 100%，但是

和其他同学比较了之后，发现有的同学成功率可以达到 100%，推测可能是读操作的数据量超过了服务器负荷，配置文件在写的时候没有考虑好，并且本身该云服务器配置不是特别好，而其他有同学配置在本地，可以达到 100% 的成功率。

其中几个操作的分析图如下：

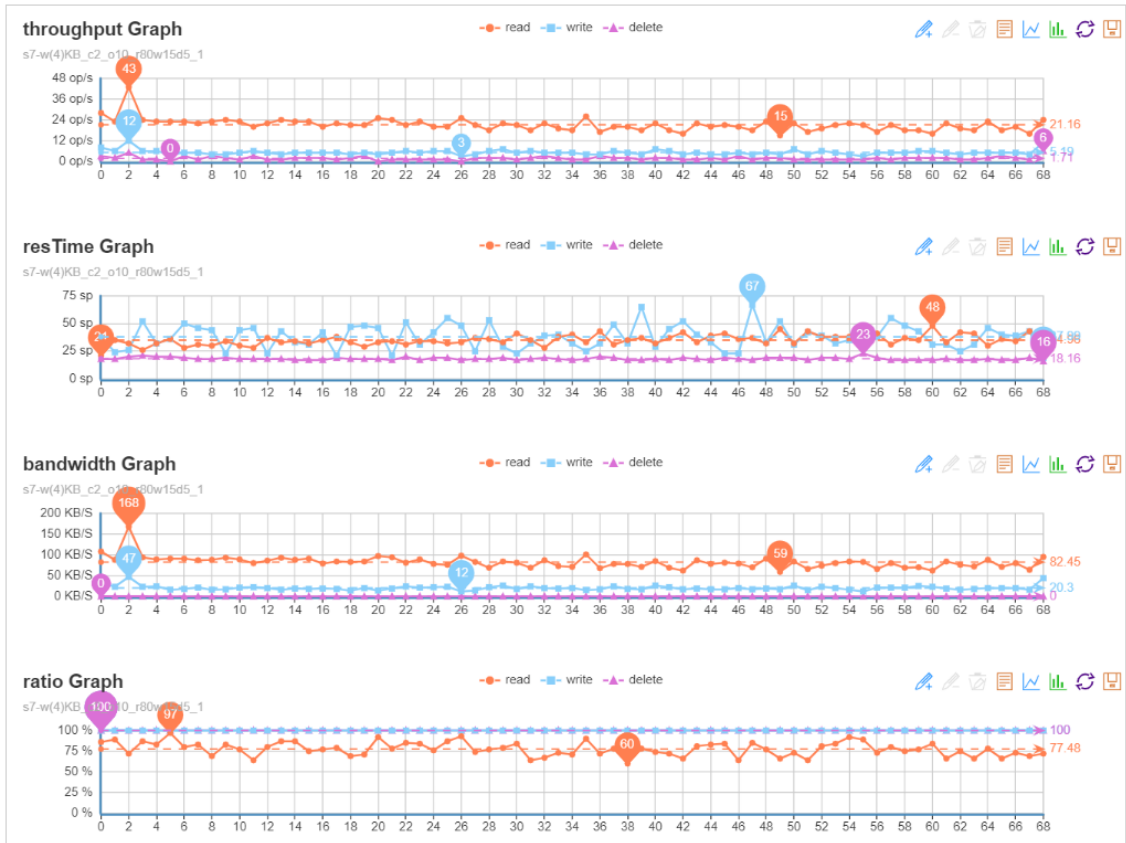


图 14 读操作的分析图

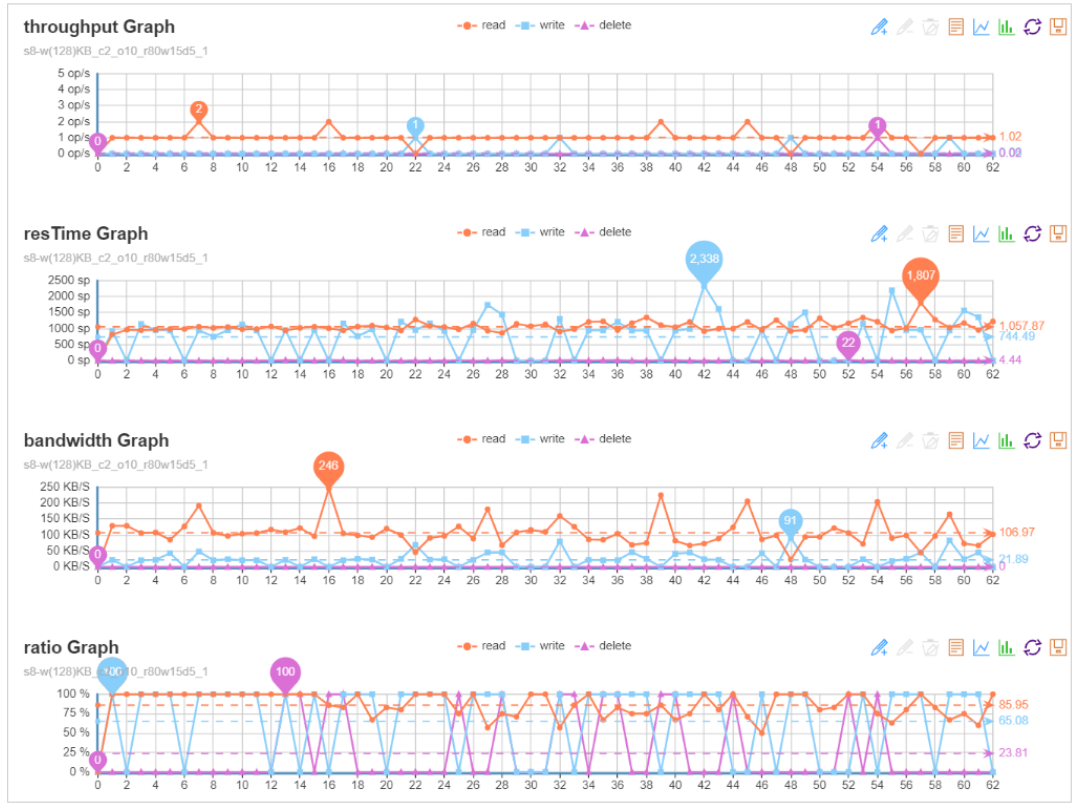


图 15 写操作的分析图

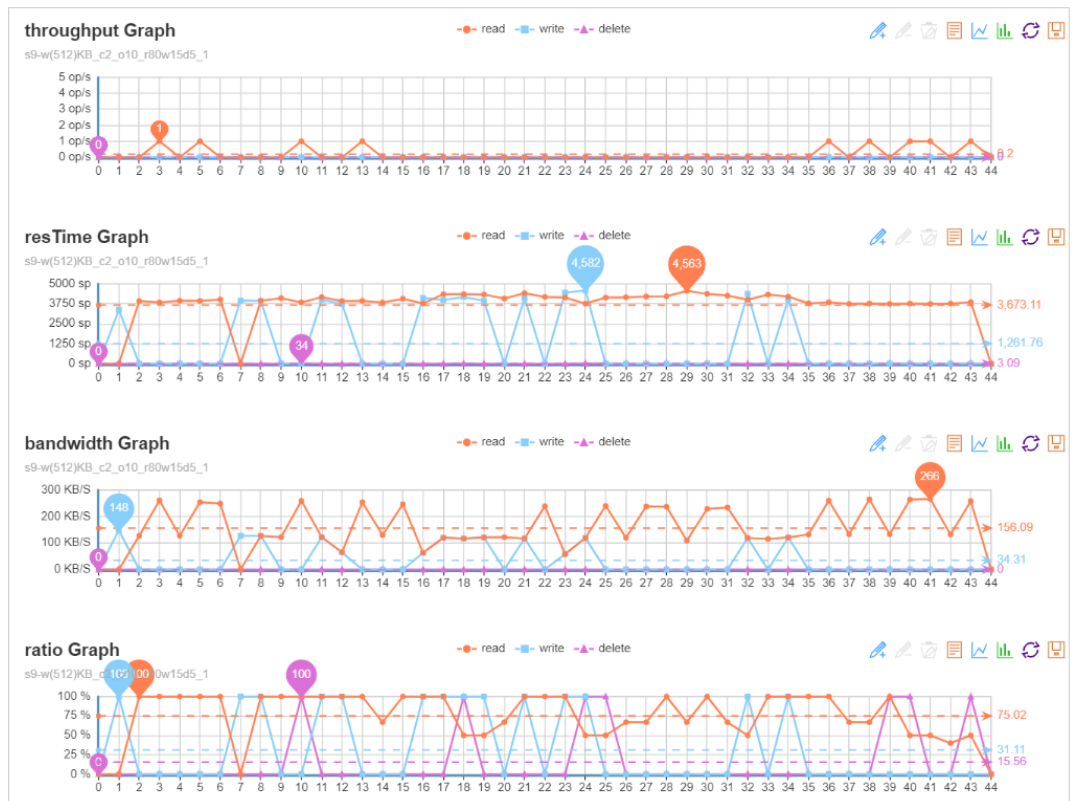


图 15 删除操作的分析图

六、实验总结

通过本次实验，我学会了整个对象存储环境的配置。首先是玩会了 Docker，在做实验的过程中又进行了其他很多操作，不仅可以用来完成对象存储的环境，也可以打包很多其他的环境，易于在不同的平台之间的移植，也巩固了物联网中间件这门课中学到的知识。然后就是配好了对象存储客户端之后，通过自己实际的操作，也明白了它的优势之处，在课后，我还可以在配好环境的云服务器上做一点扩展，比如增加 Web 客户端什么的，让自己可以随时打开浏览器就可以使用对象存储客户端，做一个简单的网盘，这也是在之后要学习的地方。还有就是通过对性能的测试，让我了解了对象存储的几个参数对运行结果的影响，巩固理解了上课学到的知识。

在实验中主要遇到的问题就是本地配环境的问题，本来想把整个环境全部配置在本机，结果在实际操作中遇到了很多奇怪的问题，结果浪费了大量时间，导致课外花了点时间并且最后还搞的很紧张。后来查到很多资料都是基于 Ubuntu 系统的，而我本地使用的是 Arch 系统，稍微有些区别，结果折腾到最后没能解决，就使用了云服务器来配置。但是云服务器配置也有好处，可以随时随地控制程序运行并查看结果，只需要有浏览器即可。

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.