2016 级

《物联网数据存储与管理》课程

实 验 报 告

姓　名　王依凡

学　号　U201614883

班　号　物联网1601班

日　期　2019.05.27

目　　录

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

本次实验为对象存储入门实验，其中主要的部分有：基础环境搭建；对象存储服务器端准备；对象存储客户端准备；对象存储测评工具的使用。

基础环境在下一节详细介绍。

对象存储服务端使用了 Swift，由 Rackspace 开发的用来为云计算提供可扩展存储的项目。

对象存储客户端使用了基于 python 的 python-swiftclient 终端客户端，同时利用网页的形式显示 swift 服务器中对象存储的内容。

对象存储测评工具是 COSBench 和 swift-bench, COSBench（cloud object storage benchmark）是 Intel 开发的，对云存储的测试工具。

三、实验环境

实验中使用的服务端操作系统操作系统使用的是 google cloud 轻量级服务器，其系统配置如图1.1所示：

```
root@instancessh:~# uname -a
Linux instancessh 4.15.0-1030-gcp #32~16.04.1-Ubuntu SMP Wed Apr 10 13:45:19 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
```

图1.1 服务端操作系统

服务端java 版本如图 1.2所示：

```
root@instancessh:~# java -version
java version "1.8.0_211"
Java(TM) SE Runtime Environment (build 1.8.0_211-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.211-b12, mixed mode)
```

图1.2 服务端java 版本

因为cosbench 是在服务端运行的，同时根据 cosbench github中的 issus 可以了解到，java7 或 8 支持 cosbench 的运行，所以这里使用 java8 版本。服务端存储空间如图1.3 所示。

```
root@instancessh:~# df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            834M     0  834M   0% /dev
tmpfs           169M   18M  152M  11% /run
/dev/sda1       9.7G  7.6G  2.1G  80% /
tmpfs           845M     0  845M   0% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           845M     0  845M   0% /sys/fs/cgroup
tmpfs           169M   24K  169M   1% /run/user/0
overlay         9.7G  7.6G  2.1G  80% /var/lib/docker/overlay2/a1341a6ed1b786e24ec9f7fff3bdc37169db415177de39de4df1832c14852564/merged
shm              64M  4.0K   64M   1% /var/lib/docker/containers/7a7388c4d5e1700125fe7ea2b8e6017d469b06c3477de8573c09dce2a12ef625/mounts/shm
```

图1.3　服务端存储空间

如图所示，因为在服务端，用 docker 跑的 swift，所以如图所示，docker 使用了大部分的磁盘容量。服务端使用的Python版本如图 1.4 所示：

```
root@instancessh:~# python --version
Python 3.6.5 :: Anaconda, Inc.
```

图1.4　服务端存储空间

如图所示，因为要使用服务端的 Jupyter 进行 python 的编写，所以直接使用了 anaconda 的 python3 版本。同时本机 python 版本如图 1.5 所示。

```
→ ~/Desktop python --version
Python 3.7.3
```

图 1.5 本机 python 版本

因为需要在本机使用基于 python 的 python-swiftclient 终端客户端，所以在这里展示了本机 python 的版本信息。

使用的服务器端：Swift 服务端

使用的客户端：Swift-UI 网页客户端和 python-swiftclient 终端客户端

四、实验内容

4.1 对象存储技术实践

（1）在租用的服务器上搭建一个 Swift 服务器端，其中总的存储空间即为副武器的总空间，不过后期可以使用外部磁盘挂载的方式增加服务器存储空间，利用 swift 相关 api 进行文件的上传下载删除容器和增加容器等操作。

（2）在本机开始 Swift 客户端程序，即利用终端输入相应的服务器地址，用户名和密码访问 swift 对象存储服务端。同时利用 swift-UI ，绑定相应的端口，在浏览器中输入相应的服务器 IP 地址和相应的端口号即可利用 GUI 访问到 swift 服务端，进而进行相关操作。

4.2 对象存储性能分析

（1） 在服务器上部署 COSBench 以及 Swift-Bench，从而在外部终端能发起评测请求，同时避免在带宽上被局限。

（2） 通过 COSBench 和 swift-bench 对 Swift 进行并发下的多组速度测试。

五、实验过程

（1） 实验环境的搭建。

本次实验需要的Python之前已经搭建完成，这里就不一一赘述。主要描述服务端安装 java 版本的过程。因为之前在服务器上安装的是 java11 的版本，但是运行 cosbench 的时候，程序输出一直是 error，看到 github 中的 issue中才了解到 cosbench 需要 java7 或者 8 版本，在Oracle官网利用wget 命令获取下载包，setup by setup 即可完成安装，这里需要注意的是需要替换系统默认的 java 版本，最终输出 java8 的系统默认版本。

（2） 搭建Swift服务端和客户端

首先在服务器上安装 docker 环境，利用 apt-get install docker.io 进行安装，之后根据iot-tutorial 中的openstack-swift-docker的镜像安装方法进行安装，其过程会提示安装python-swiftclient客户端，一步一步按照提示进行安装即可，最终在终端输入 docker ps 显示所有镜像信息，其结果如图 1.6 所示。

```
root@instancessh:~# docker ps
CONTAINER ID        IMAGE                    COMMAND                  CREATED             STATUS              PORTS                     NAMES
7a7388c4d5e1        openstack-swift-docker   "/bin/sh -c /usr/loc…"   12 days ago         Up 12 days          0.0.0.0:12345->8080/tcp   openstack-swift
```

图 1.6 服务器端 swift 配置

同时在终端中输入 swift 检测在服务器上的客户端配置是否完成，其结果如图 1.7 所示。

```
root@instancessh:~# swift
usage: swift [--version] [--help] [--os-help] [--snet] [--verbose]
             [--debug] [--info] [--quiet] [--auth <auth_url>]
             [--auth-version <auth_version> |
                 --os-identity-api-version <auth_version> ]
             [--user <username>]
             [--key <api_key>] [--retries <num_retries>]
             [--os-username <auth-user-name>]
             [--os-password <auth-password>]
             [--os-user-id <auth-user-id>]
             [--os-user-domain-id <auth-user-domain-id>]
             [--os-user-domain-name <auth-user-domain-name>]
             [--os-tenant-id <auth-tenant-id>]
             [--os-tenant-name <auth-tenant-name>]
             [--os-project-id <auth-project-id>]
             [--os-project-name <auth-project-name>]
             [--os-project-domain-id <auth-project-domain-id>]
             [--os-project-domain-name <auth-project-domain-name>]
             [--os-auth-url <auth-url>]
             [--os-auth-token <auth-token>]
             [--os-storage-url <storage-url>]
             [--os-region-name <region-name>]
             [--os-service-type <service-type>]
             [--os-endpoint-type <endpoint-type>]
             [--os-cacert <ca-certificate>]
             [--insecure]
             [--os-cert <client-certificate-file>]
             [--os-key <client-certificate-key-file>]
             [--no-ssl-compression]
             [--force-auth-retry]
             <subcommand> [--help] [<subcommand options>]

Command-line interface to the OpenStack Swift API.

Positional arguments:
  <subcommand>
    delete              Delete a container or objects within a container.
    download            Download objects from containers.
    list                Lists the containers for the account or the objects
                        for a container.
    post                Updates meta information for the account, container,
                        or object; creates containers if not present.
    copy                Copies object, optionally adds meta
    stat                Displays information for the account, container,
                        or object.
    upload              Uploads files or directories to the given container.
    capabilities        List cluster capabilities.
    tempurl             Create a temporary URL.
```

图 1.7 swift 客户端配置

如图所示，输入相应的命令，其可完成测试，这里同样在本机利用 pip3 install python-swiftclient 命令进行本机 swift 客户机的安装，然后输入对应服务端的 ip 地址进行测试，其结果如图 1.8所示。

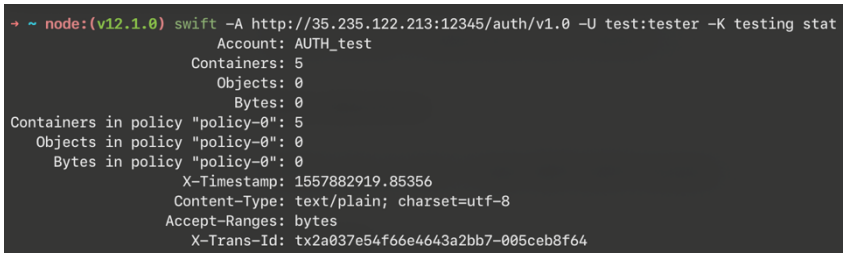


图 1.8 swift 客户端测试

如图所示，输入相应的参数，stat 表示输出 swift服务端状态信息，然后利用 list 参数输出服务端容器信息，其结果如图 1.9 所示。

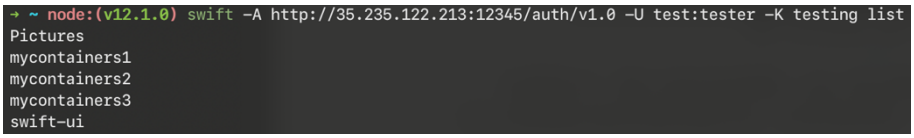


图 1.9 swift 客户端测试

如图所示，list 参数输出的是服务端的容器信息，如图所示，共有 5 个容器。为更好的进行图形化操作 swift 服务端，在网上借鉴了 https://github.com/fanatic/swift-ui 的 repo，其 instruction 如图 1.10 所示

图 1.10 swift-ui instruction

如图所示，修改对应的配置文件，将文件上传到服务端，之后访问对应的 ip 地址和对应的端口号，其结果如图 1.11 所示。

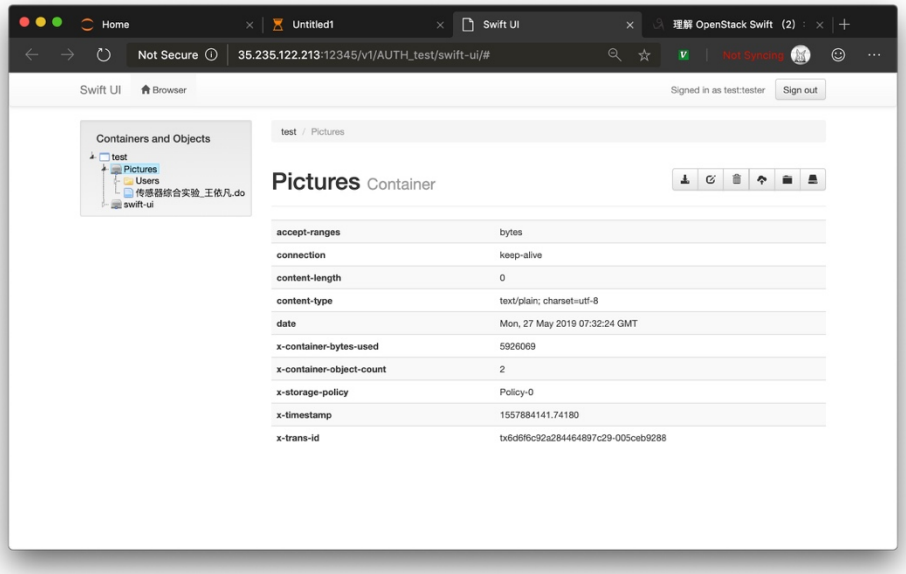


图 1.11 swift-ui

如图所示，该界面显示了容器的详细信息，支持服务端的增删改查操作。

至此，完成了服务端和客户端的所有配置信息，在服务器上配置了 swift 的服务端和客户端，在本机配置了 swift 的客户端。

（3） 搭建Swift服务端评测工具

a. CosBench 评测工具

打开 COSBench 的 github 官网 https://github.com/intel-cloud/cosbench，根据 issue 提示，下载版本号为 0.4.2.c4 的 COSBench 评测工具，之后解压相关安装包，在终端输入 unset http_proxy 和 sh start-all.sh 运行 cosbench 评测工具，之后在浏览器中打开 ip 地址:19088 即可进入相关页面，其结果如图 1.12 所示.

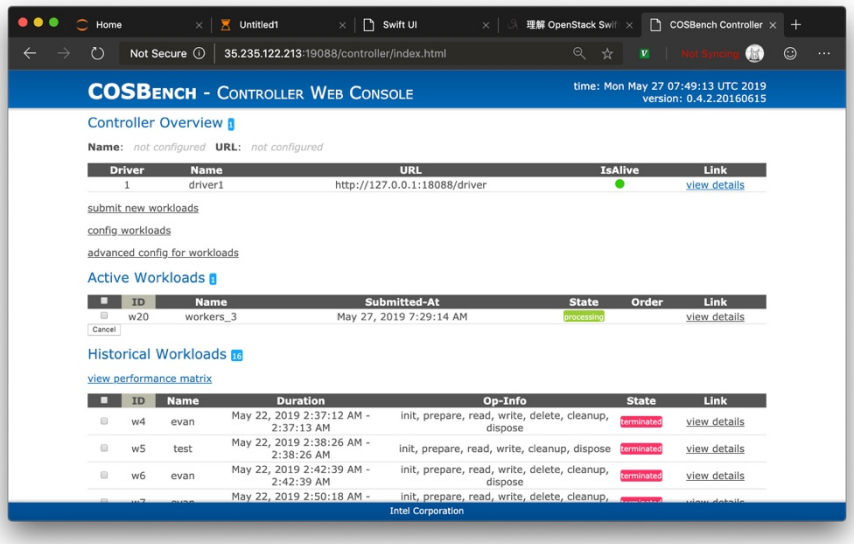


图 1.12 COSBench 评测工具

如图所示，点击 advanced config for workloads，即可进入相关配置界面，输入相关参数信息，然后提交 workload 即可完成任务，其提交界面如图 1.13 所示。

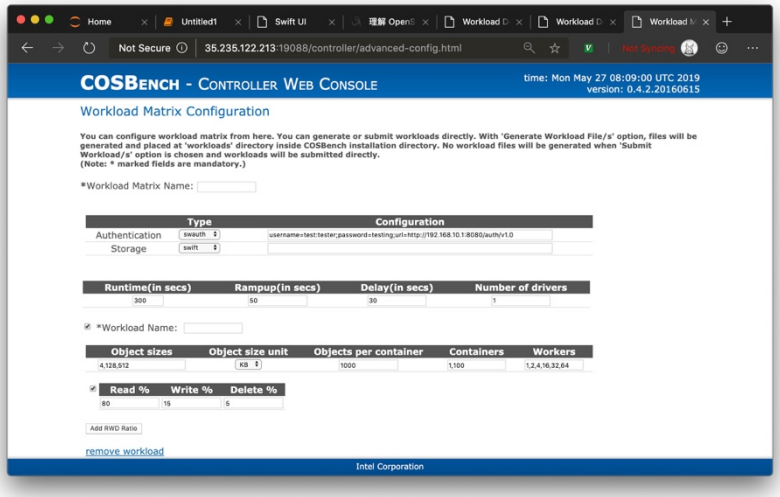


图 1.13 workloads submit 界面

之后运行结束后，点击相关信息，即可显示最终的输出结果，其结果如图 1. 14 所示。

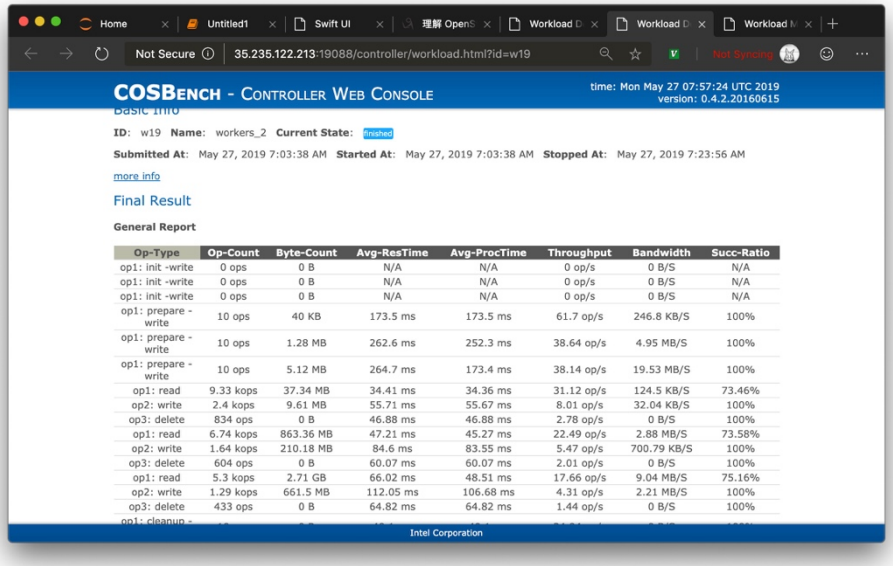


图 1.13 workloads complete 界面

如图所示，测评完毕。

b. swift-bench 评测工具

通过 pip2 install swift-bench 进行安装，这里注意，虽然可以用 pip3 install swift-bench 进行安装，但 swift-bench 的版本是 python2.x，很多语法规则

在 python3.x 中无法运行，所以一定要检查 swift-bench 的安装路径，安装成功之后，在终端运行 swift-bench -h，其程序输出如图 1.14 所示。

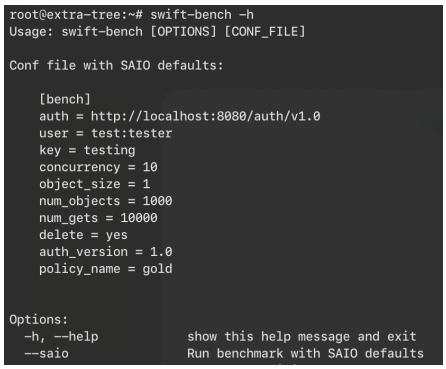


图 1.14 swift-bench 示例

如图所示，测评程序安装成功。

六、实验结果分析

（1） 利用 COSBench 评测工具进行性能测试

从实验内容中的测试结果可以看出，workload 中每一步骤都成功执行了，所以在实验结果分析中，我挑选了其中的几种进行了比较测试，针对 16kb、128kb 和 1mb 选择 1、4、8workers 进行读写测试，测试结果如下：

表1 16kb对照表

16kb 1workers

16kb 4workers

16kb 8workers

Final Result

General Report

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: prepare - write	10 ops	160 KB	299.1 ms	299 ms	33.04 op/s	528.64 KB/S	100%
op1: read	8.68 kops	138.85 MB	18.03 ms	18 ms	28.94 op/s	462.97 KB/S	75.32%
op2: write	2.12 kops	34 MB	34.87 ms	34.78 ms	7.09 op/s	113.37 KB/S	100%
op3: delete	708 ops	0 B	30.03 ms	30.03 ms	2.36 op/s	0 B/S	100%
op1: read	6.6 kops	105.57 MB	96.32 ms	96.25 ms	22.02 op/s	352.27 KB/S	74.81%
op2: write	1.66 kops	26.62 MB	162.52 ms	162.45 ms	5.55 op/s	88.84 KB/S	99.88%
op3: delete	556 ops	0 B	139.13 ms	139.13 ms	1.86 op/s	0 B/S	100%
op1: read	6.82 kops	109.1 MB	182.8 ms	182.7 ms	22.79 op/s	364.67 KB/S	76.2%
op2: write	1.68 kops	26.88 MB	361.23 ms	361.14 ms	5.62 op/s	89.85 KB/S	99.94%
op3: delete	533 ops	0 B	310.55 ms	310.55 ms	1.78 op/s	0 B/S	100%
op1: cleanup - delete	10 ops	0 B	23.6 ms	23.6 ms	42.19 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

表2 128kb对照表

128kb 1workers

128kb 4workers

128kb 8workers

Final Result

General Report

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: prepare - write	10 ops	1.28 MB	243 ms	236.5 ms	41.74 op/s	5.34 MB/S	100%
op1: read	8.56 kops	1.1 GB	17.65 ms	17.38 ms	28.53 op/s	3.65 MB/S	73.73%
op2: write	2.21 kops	283.01 MB	36.93 ms	36.15 ms	7.37 op/s	943.49 KB/S	100%
op3: delete	769 ops	0 B	27.5 ms	27.5 ms	2.56 op/s	0 B/S	100%
op1: read	6.38 kops	816.77 MB	98.55 ms	93.41 ms	21.28 op/s	2.72 MB/S	73.05%
op2: write	1.61 kops	206.08 MB	182.09 ms	181.13 ms	5.37 op/s	687.37 KB/S	100%
op3: delete	555 ops	0 B	119.19 ms	119.19 ms	1.85 op/s	0 B/S	100%
op1: read	6.07 kops	776.45 MB	206.46 ms	193.67 ms	20.26 op/s	2.59 MB/S	73.68%
op2: write	1.59 kops	203.65 MB	402.63 ms	401.76 ms	5.31 op/s	680.18 KB/S	100%
op3: delete	538 ops	0 B	283.07 ms	283.07 ms	1.8 op/s	0 B/S	100%
op1: cleanup - delete	10 ops	0 B	24.7 ms	24.7 ms	40.49 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

表3 1mb对照表

1mb 1workers

1mb 4workers

1mb 8workers

Final Result

General Report

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: prepare - write	10 ops	10 MB	383.9 ms	225.2 ms	30.64 op/s	30.64 MB/S	100%
op1: read	4.32 kops	4.32 GB	39.99 ms	24.78 ms	14.4 op/s	14.4 MB/S	73.04%
op2: write	1.12 kops	1.12 GB	74.38 ms	62.15 ms	3.74 op/s	3.74 MB/S	100%
op3: delete	376 ops	0 B	35.8 ms	35.8 ms	1.25 op/s	0 B/S	100%
op1: read	4.26 kops	4.26 GB	171.17 ms	98.14 ms	14.2 op/s	14.2 MB/S	73.82%
op2: write	1.11 kops	1.11 GB	272.78 ms	265.13 ms	3.7 op/s	3.7 MB/S	100%
op3: delete	370 ops	0 B	113.05 ms	113.05 ms	1.23 op/s	0 B/S	100%
op1: read	4.02 kops	4.02 GB	347.67 ms	186.97 ms	13.43 op/s	13.43 MB/S	71.58%
op2: write	1.08 kops	1.08 GB	569.22 ms	563.71 ms	3.6 op/s	3.6 MB/S	99.91%
op3: delete	364 ops	0 B	280.76 ms	280.76 ms	1.22 op/s	0 B/S	100%
op1: cleanup - delete	10 ops	0 B	27.3 ms	27.3 ms	36.63 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

可以得到随着 Worker 数量的增加，在读写删三种操作的整体平均时间、处理的平均时间都有显著增加，但吞吐率、带宽随着 worker 数量的增加没有显著增

加的迹象。但在同一个 worker 数量的情况下随着 size 的变大，整体的平均时间和处理的平均时间以及带宽也都有显著增加。

同时成功率不收以上参数的影响，读取的成功率在 75%左右，其他操作的成功率都是 100%。

（2） 利用 swift-bench 评测工具进行性能测试

在使用 swift-bench 进行测试的过程可以进行自定义的参数配置，可以对对象的大小，写入次数、读取次数、并发量进行配置，然后在指定的 Swift 上进行运行。

```
 % swift-bench -A http://ali.grayxu.cn:12345/auth/v1.0 -U test:tester -K testing -s 1024 -c 100
swift-bench 2019-05-15 15:28:05,866 INFO Auth version: 1.0
swift-bench 2019-05-15 15:28:06,473 INFO Auth version: 1.0
swift-bench 2019-05-15 15:28:08,496 INFO 59 PUTS [0 failures], 29.5/s
swift-bench 2019-05-15 15:28:21,666 INFO 1000 PUTS **FINAL** [0 failures], 65.9/s
swift-bench 2019-05-15 15:28:21,666 INFO Auth version: 1.0
swift-bench 2019-05-15 15:28:23,716 INFO 153 GETS [0 failures], 76.0/s
swift-bench 2019-05-15 15:28:38,720 INFO 2100 GETS [0 failures], 123.4/s
swift-bench 2019-05-15 15:28:53,726 INFO 3884 GETS [0 failures], 121.3/s
swift-bench 2019-05-15 15:29:08,732 INFO 5627 GETS [0 failures], 119.7/s
swift-bench 2019-05-15 15:29:23,735 INFO 7426 GETS [0 failures], 119.7/s
swift-bench 2019-05-15 15:29:38,736 INFO 9370 GETS [0 failures], 121.6/s
swift-bench 2019-05-15 15:29:43,179 INFO 10000 GETS **FINAL** [0 failures], 122.7/s
swift-bench 2019-05-15 15:29:43,179 INFO Auth version: 1.0
swift-bench 2019-05-15 15:29:45,224 INFO 119 DEL [0 failures], 59.3/s
swift-bench 2019-05-15 15:29:54,645 INFO 1000 DEL **FINAL** [0 failures], 87.5/s
swift-bench 2019-05-15 15:29:54,645 INFO Auth version: 1.0
```

图 1.15 swift-bench 示例

如图所示，为一个运行 swift-bench 例子命令，设置为 50 并发量，对象大小为 1KB(1024B)，写入次数默认为 100，读取次数默认为 500，其命令结果如图 1.16 所示。

```
root@extra-tree:~# swift-bench -A http://127.0.0.1:12345/auth/v1.0 -U test:tester -K testing -s 1024 -c 100 -n 100 -g 500
swift-bench 2019-05-29 03:15:38,731 INFO Auth version: 1.0
swift-bench 2019-05-29 03:15:39,624 INFO Auth version: 1.0
swift-bench 2019-05-29 03:15:42,930 INFO 100 PUTS **FINAL** [0 failures], 30.5/s
swift-bench 2019-05-29 03:15:42,931 INFO Auth version: 1.0
swift-bench 2019-05-29 03:15:44,981 INFO 101 GETS [0 failures], 50.1/s
swift-bench 2019-05-29 03:15:49,113 INFO 500 GETS **FINAL** [0 failures], 81.3/s
swift-bench 2019-05-29 03:15:49,113 INFO Auth version: 1.0
swift-bench 2019-05-29 03:15:51,126 INFO 100 DEL **FINAL** [0 failures], 50.5/s
swift-bench 2019-05-29 03:15:51,127 INFO Auth version: 1.0
```

图 1.16 swift-bench 示例

之后通过大量数据总结，利用 Excel 生成表格，结果表如下。

表 4

对象大小	并发数	写入操作速率	读取操作速率	删除操作速率	写入速度（Byte/s）	读取速度（Byte/s）	删除速度（Byte/s）
1B	50	34.8	73.5	52.2	34.8	73.5	52.2
1KB	50	33.1	80.8	46.2	33894.4	82739.2	47308.8
512KB	50	24.4	42.9	54.0	12792627.2	22491955.2	28311552
1MB	50	18.4	29.6	46.4	39321600	52114227	86297804.8
1MB	1	20.7	31.9	35.6	21705523.2	33449574.4	37329305.6

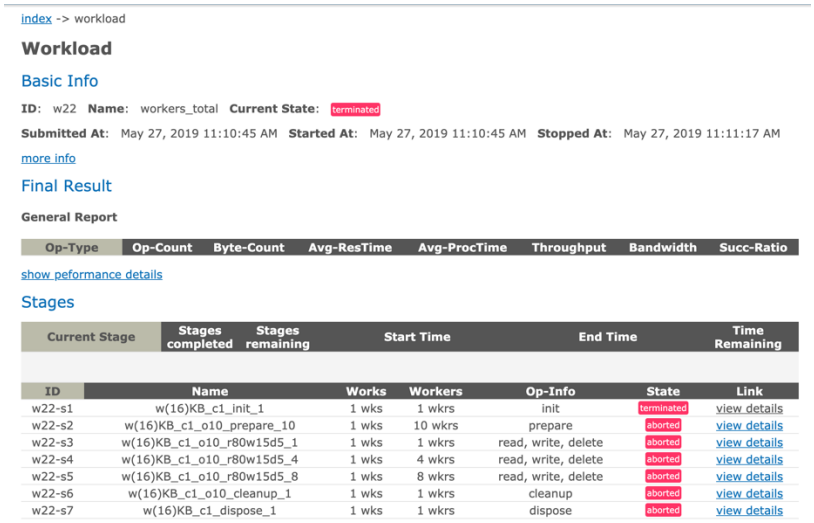
1MB	10	37.5	49.7	82.3	24641536	34707865.6	66479718.4
1MB	20	35.6	46.8	82.2	26528972.8	35756441.6	57042534.4
1MB	30	36.3	44.3	78.4	25585254.4	32191283.2	58405683.2
1MB	40	35.1	46.1	76.2	24326963.2	34917580.8	57147392
1MB	50	35.5	45.8	77.2	22439526.4	32925286.4	57147392

在表中容易观察到，随着对象的大小增大，写入和读取的速度都有提升（单位是 Byte/s）。其中

同时，固定设置参数对象大小为 1MB，上传 100 次，读取 500 次，取速度的平均值来进行统计。由表中可以观察到，并发量从 1 提升到 10 的时候，写入读取删除的速度都有一个很明显的提升，但是从并发量 10 到 50 过程中，写入，读取和删除速度都基本维持在一个不变的水平，甚至出现了下降的趋势，通过询问同学，我了解到，当服务器的可用资源支持并发总量为 10 时，1 到 10 的过程可以增加对系统的可用度，但超过 10 之后，系统并没有那么多的可用资源供多并发使用，于是每个并发过程的资源收到限制，无法完成了读写速度的大幅度提升，同时，当设置并发量为 50，文件大小为 50Mb 的时候，系统出现了无响应的状态，这也证明了系统此时出现了高负载的情况。

七、实验总结

（1） 因为之前比较熟悉 unix 和 linux 的终端命令，同时比较熟悉在服务器上进行相关的操作，所以环境配置方面没有太大的问题，主要是在配置 cosbench 中出现了一些问题，因为需要 java 8 的版本，所以在安装 java 环境过程花费了写功夫。同时在测试过程中，很多时候会因为测试的数据超过了服务器的负载而导致服务器长时间没有响应，如图所示为评测程序终止的情况，需减少相应的参数，才能完成实验（后期测试过程中部署了内存较大的轻量级服务器完成了实验）



index -> workload

Workload

Basic Info

ID: w22 **Name:** workers_total **Current State:** terminated

Submitted At: May 27, 2019 11:10:45 AM **Started At:** May 27, 2019 11:10:45 AM **Stopped At:** May 27, 2019 11:11:17 AM

more info

Final Result

General Report

Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio

show peformance details

Stages

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

ID	Name	Works	Workers	Op-Info	State	Link
w22-s1	w(16)KB_c1_init_1	1 wks	1 wkrs	init	terminated	view details
w22-s2	w(16)KB_c1_o10_prepare_10	1 wks	10 wkrs	prepare	aborted	view details
w22-s3	w(16)KB_c1_o10_r80w15d5_1	1 wks	1 wkrs	read, write, delete	aborted	view details
w22-s4	w(16)KB_c1_o10_r80w15d5_4	1 wks	4 wkrs	read, write, delete	aborted	view details
w22-s5	w(16)KB_c1_o10_r80w15d5_8	1 wks	8 wkrs	read, write, delete	aborted	view details
w22-s6	w(16)KB_c1_o10_cleanup_1	1 wks	1 wkrs	cleanup	aborted	view details
w22-s7	w(16)KB_c1_dispose_1	1 wks	1 wkrs	dispose	aborted	view details

（2） 在评测方面通过COSBench、Swift-Bench来对Swift的传输性能在不同的参数设置上进行一个全方面的检测统计，在评测过程中，因为操作失误 ，会有很多容器没有及时清理，这时需要 python 的帮助进行脚本的书写，进而进行相关操作的实现。

（3） 通过本次实验，我了解到，文件传输共有 2 个步骤，一个是连接的建立，一个是文件的传输过程，当文件较小时，主要时间开销是建立连接所花费的时间，当文件较大时，总时间只是会因为文件的大小二产生轻微的改变，所以在最后文件传输的总速率趋于平稳。

（4） 本次实验体会：

本次实验是对象存储入门实验，主要是在服务器搭建一个对象存储服务端，然后进行性能的评测，因为之前没有写过 Python 的脚本，所以这次实验增加了我对 python 的熟悉度，同时更加了解了 对象存储的特性，通过调整客户端的数量可能影响到数据读、写成功率。但是除此之外，根据同学的实验过程，还是有很多没有测评的地方，比如利用 python 以及相关 api 进行自己代码书写，然后进行评测，希望以后多多利用 python 解决问题，完成更多的实验内容。

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.

[4] GitHub

[5] CSDN

[6] StackOverflow