2016 级

《物联网数据存储与管理》课程

实 验 报 告

姓　名　付内东

学　号　U201614891

班　号　物联网 1601 班

日　期　2019.05.29

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

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。

就像文件一样，对象包含数据，但是和文件不同的是，对象在一个层结构中不会再有层级结构。每个对象都在一个被称作存储池的扁平地址空间的同一级别里，一个对象不会属于另一个对象的下一级。

对象存储系统（Object-Based Storage System）是综合了 NAS 和 SAN 的优点，同时具有 SAN 的高速直接访问和 NAS 的数据共享等优势，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。

Minio 是 Apache License v2.0 下发布的对象存储服务器。它与 Amazon S3 云存储服务兼容。它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/ VM 映像。对象的大小可以从几 KB 到最大 5TB

COSBench 是衡量云对象存储服务性能的基准测试工具。它包含两个关键元素：Driver 和 Controller，前者产生负载，向目标对象存储系统发布操作，收集统计数据，后者协调 driver，处理实时状态，接收提交的负载。

三、实验环境

实验中的软硬件环境如下表所示

表 1 实验环境

实验名称	实验 1（easy）	实验 2（medium）
处理器	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
操作系统	Manjaro	Manjaro
Python 版本	Python 3.7.3	Python 3.7.3
Java 版本	1.8.0_212	1.8.0_212
Go 版本	go1.12.4	go1.12.4
Server	Minio	mock-s3
Client	Minio Client	
Test	S3bench	COSBench

四、实验内容

本次实验在 Linux 环境下进行，主要进行的实验内容如下：

1. 熟悉 git 和 github 操作以及安装各种环境
2. 利用 Minio/mc 和 mock-s3 进行对象存储系统实践，进行简单的添加删除操作
3. 利用 COSBench 和 S3bench 进行负载测试
4. 对测试结果进行分析

4.1 对象存储技术实践

1. 根据官网的步骤下载 Minio 服务端，通过浏览器登陆 127.0.0.1 查看效果，进行简单的创建 bucket，上传文件，删除文件和 bucket 操作

2. 在官网下载 mc 客户端，将 mc 客户端与 minio 服务端连接

3. 上载老师给的负载样例 s3bench，修改参数后看运行的各种状态指标

4. 下载 mock-s3，运行后利用 cosbench 修改参数进行可视化的性能比较

4.2 对象存储性能分析

1. 读写性能对比。

2. 测试块大小对运行结果的影响。

   将负载样例中的 NumClient 和 NumSample 数目保持不变，块大小从逐渐翻倍增大，观察运行结果。

3. 测试 workers 值对运行结果的影响

   在 mock-S3 测试时将负载样例中的 workers 数改为 1,2,4,8,16,32，查看 workers 数翻倍对测试结果的影响

五、实验过程

5.1 Minio 服务端与 S3bench 测试

1.配置 Minio 服务端，根据官网上的命令配置 minio server

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
chmod +x minio
./minio server /mnt/data
```

因为这里自己虽然 chmod 了还是无法运行所以加了 sudo

运行结果如下图所示：

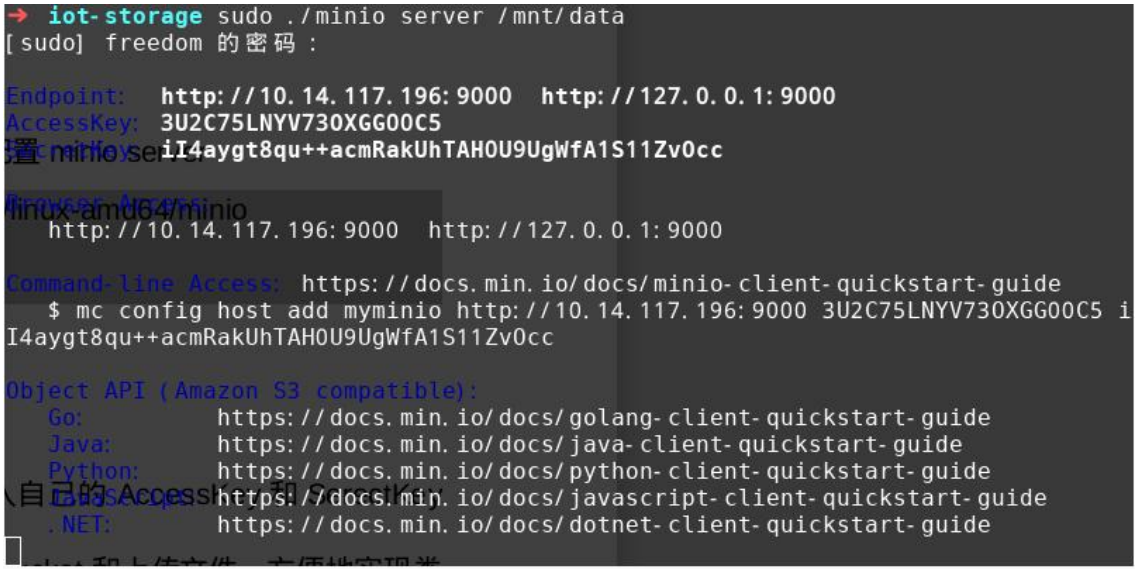

图 5.1 运行 minio 客户端

输入地址，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，如图所示。

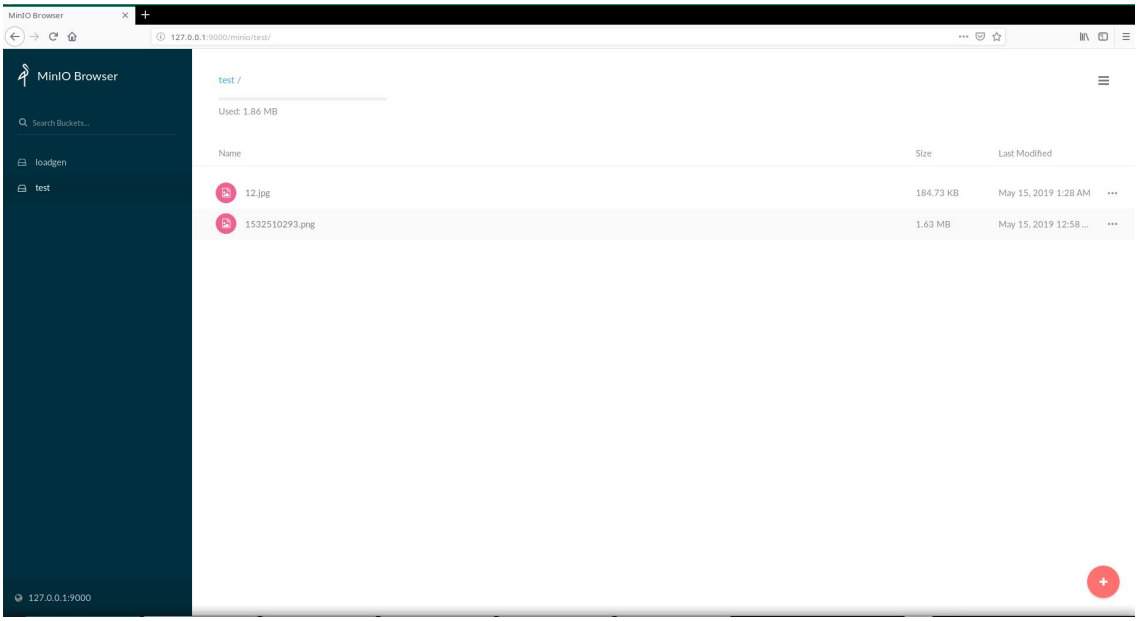

图 5.2 网页打开 minio 服务端

2. 配置 mc client

此时另开一个终端，输入如下命令：

```
wget https://dl.minio.io/client/mc/release/linux-amd64/mc
chmod +x mc
./mc -help
```

下载 mc client 以及查看是否运行正常，运行结果如图所示：

```
→ iot-storage ./mc -help
NAME:
  mc - MinIO Client for cloud storage and filesystems.

USAGE:
  mc [FLAGS] COMMAND [COMMAND FLAGS | -h] [ARGUMENTS...]

COMMANDS:
  ls       list buckets and objects
  mb       make a bucket
  rb       remove a bucket
  cat      display object contents
  head     display first 'n' lines of an object
  pipe     stream STDIN to an object
  share    generate URL for temporary access to an object
  cp       copy objects
  mirror   synchronize object(s) to a remote site
  find     search for objects
  sql      run sql queries on objects
  stat     show object metadata
  diff     list differences in object name, size, and date between two buckets
  rm       remove objects
  event    configure object notifications
  watch    listen for object notification events
  policy   manage anonymous access to buckets and objects
  admin    manage MinIO servers
  session  resume interrupted operations
  config   configure MinIO client
  update   update mc to latest release
  version  show version info
```

图 5.3 运行 mc 客户端

将客户端与之前开着的服务端连接，可以看到已经成功与服务端连接上，测试的功能正常。

```
→ iot-storage ./mc config host add myminio http://10.14.117.196:9000 3U2C75LNYV730XGGO0C5 iI
4aygt8qu++acmRakUhTAHOU9UgWfA1S11ZvOcc
Added `myminio` successfully.
→ iot-storage ./mc ls myminio
[2019-05-15 02:45:33 CST]      0B loadgen/
[2019-05-15 01:28:12 CST]      0B test/
→ iot-storage ./mc mb test_mc
Bucket created successfully `test_mc`.
```

图 5.4 连接服务端

3. 配置 S3bench 并提交负载样例

首先安装好 go 语言环境

图 5.5 go 语言环境配置

然后按 git 上给的命令安装 S3bench

```
go get -u github.com/igneous-systems/s3bench
```

安装完成后写一个脚本文件依次改变 Objectsize 的大小和客户端数量并对输出结果重定向到文件 test.txt 中。脚本内容已随报告上传到自己的目录下。

（1）将 ObjectSize 大小改为 1kb，2kb，4kb，10kb，20kb，40kb，100kb，200kb，400kb，1000kb,10000kb 进行测试，客户端数量为 10，sample 数量为 100,测试结果如下表

ObjectSize	Total Transferred	write	read
1kb	0.098 MB	1.21 MB/s	5.16 MB/s
2kb	0.195 MB	2.24 MB/s	8.26 MB/s
4kb	0.391 MB	5.54 MB/s	21.88 MB/s
10kb	0.977 MB	12.38 MB/s	24.71 MB/s
20kb	1.953 MB	18.46 MB/s	56.17 MB/s
40kb	3.906 MB	37.77 MB/s	107.79 MB/s
100kb	9.766 MB	74.82 MB/s	247.95 MB/s
200kb	19.531 MB	166.88 MB/s	608.46 MB/s
400kb	39.062 MB	314.84 MB/s	2145.91 MB/s
1000kb	97.656 MB	679.85 MB/s	3252.51 MB/s
10000kb	976.562 MB	1050.18 MB/s	5828.48 MB/s

图 5.6 样本数为 100，客户端数量为 10 时的吞吐率

可以看到，在任意一组数据中，读操作都要比写操作要快，本次实验将自己的个人 PC 作为服务器，而 PC 中磁盘由于自身结构导致读性能本来就远远高于写性能，这次进行测试时本来想测试读写的吞吐率上限，但由于传输的数据总量过小于是只能看到读写吞吐率一直在递增，还未达到上限，于是将样本数变为 1000，增大了块大小，最后得到的结果如下：

ObjectSize	Total Transferred	write	read
1kb	0.977 MB	0.48 MB/s	9.48 MB/s
2kb	1.953 MB	2.33 MB/s	13.22 MB/s
4kb	3.906 MB	4.16 MB/s	22.17 MB/s
10kb	9.766 MB	11.39 MB/s	67.51 MB/s
20kb	19.531 MB	23.40 MB/s	109.36 MB/s
40kb	39.062 MB	38.01 MB/s	249.62 MB/s
100kb	97.656 MB	93.98 MB/s	560.64 MB/s
200kb	195.312 MB	181.42 MB/s	1181.76 MB/s
400kb	390.625 MB	372.20 MB/s	2367.08 MB/s
1000kb	976.562 MB	810.26 MB/s	4754.98 MB/s
10000kb	9765.625 MB	837.39 MB/s	1290.65 MB/s
20000kb	19531.250 MB	549.29 MB/s	1219.34 MB/s
100000kb	97656.250 MB	551.31 MB/s	1339.25 MB/s

图 5.7 样本数为 1000，客户端数量为 10 时的吞吐率

从本例中可以看到，随着数据总量的不断变大，读写的吞吐率显示不断的上升后开始下降。可以看到，当数据总量很少时，服务器的吞吐率也不会很高，原因是当数据总量很少时，用于磁盘读写的时间其实占比很少，反而是读写的准备工作用时长，而计算吞吐率时这些时间也是要一起算进去的，于是乎会发现当数据总

量很少，服务器吞吐率也很小。当数据总量不断的增大时，磁盘读写所占的时间也越来越多，吞吐率也越来越能接近磁盘真实的读写速率。当读写数据总量超标时，主存放不下这么多数据就需要将数据放入硬盘中，而硬盘的读写速率是远远小于主存的，将数据从硬盘调入主存中需要更多的时间，这就导致了吞吐率的下降。对比了一下其他同学的数据，发现我的电脑读写速率比较快而且数据总量上限是比较高的，应该是因为自己的内存是 16G 的而且 manjaro 装在 SSD 上的原因，但意识到这样跑 S3bench 测试样例我的 SSD 损耗应该会比较大就一阵心疼（电脑刚配不久）。

（2）将客户端数量由 10 个改为 100 个看并发情况

ObjectSize	Total Transferred	write/10	read/10	write/100	read/100
1kb	0.977 MB	0.48 MB/s	9.48 MB/s	6.68 MB/s	8.77 MB/s
2kb	1.953 MB	2.33 MB/s	13.22 MB/s	12.71 MB/s	20.76 MB/s
4kb	3.906 MB	4.16 MB/s	22.17 MB/s	24.39 MB/s	36.24 MB/s
10kb	9.766 MB	11.39 MB/s	67.51 MB/s	62.78 MB/s	93.06 MB/s
20kb	19.531 MB	23.40 MB/s	109.36 MB/s	125.81 MB/s	196.70 MB/s
40kb	39.062 MB	38.01 MB/s	249.62 MB/s	222.56 MB/s	288.22 MB/s
100kb	97.656 MB	93.98 MB/s	560.64 MB/s	360.84 MB/s	587.98 MB/s
200kb	195.312 MB	181.42 MB/s	1181.76 MB/s	617.06 MB/s	1631.99 MB/s
400kb	390.625 MB	372.20 MB/s	2367.08 MB/s	730.26 MB/s	2090.86 MB/s
1000kb	976.562 MB	810.26 MB/s	4754.98 MB/s	1122.85 MB/s	4052.46 MB/s
10000kb	9765.625 MB	837.39 MB/s	1290.65 MB/s	886.26 MB/s	2917.13 MB/s

图 5.8 样本数为 1000，客户端数量为 100 时的吞吐率比较

可以看到，当客户端数量为 100 时，由于并发执行相同数据量情况下写操作的吞吐率显著提高，而读操作的吞吐率却没有显著的变化。当数据总量达到 9765.625MB 时，由于系统资源有限，增加客户端数量对写操作吞吐率并没有太大的提升。

5.2 mock-S3 与 Cosbench 测试

1.配置 mock-S3

配置好 python 环境后执行如下命令

```
git clone https://github.com/Zhan2012/mock-s3.git
cd mock_s3
python main.py --host 127.0.0.1 --port 9000 --root ./root
```

打开服务端后终端如图所示;

图 5.9 打开 mock-S3 服务端

在浏览器中输入网址 127.0.0.1:9000 即可访问

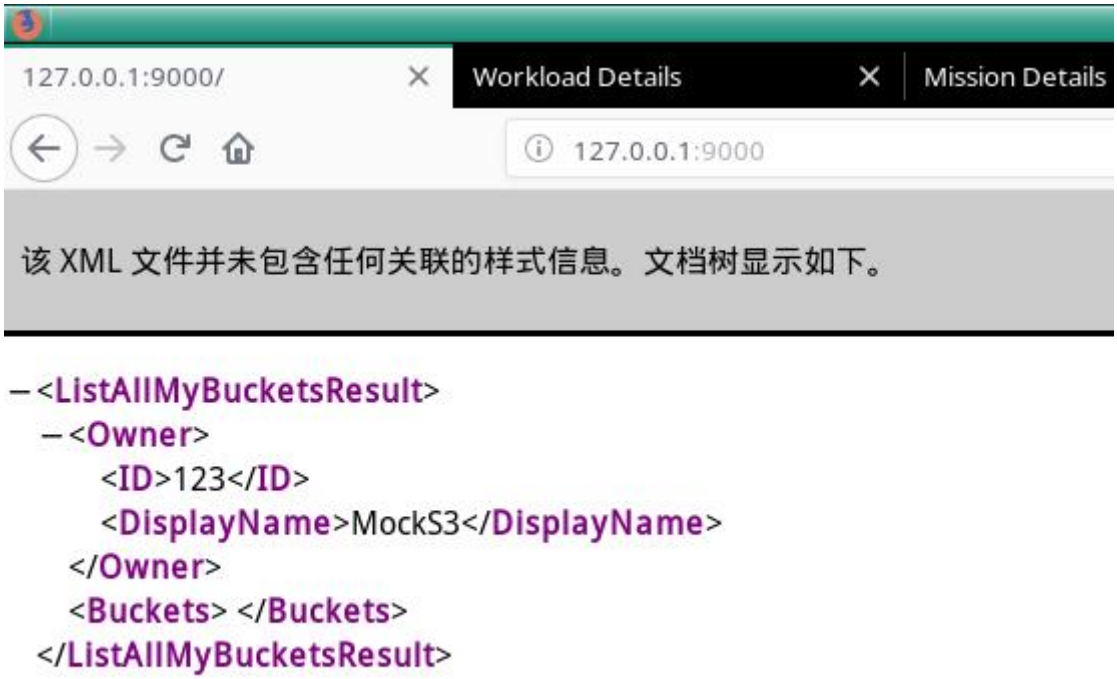

图 5.10 mock-s3 服务端界面

界面无任何功能，模拟服务程序。

将老师提供的 cosbench 解压后输入 unset http_proxy 绕过代理设置，使得控制器和驱动程序可以进行交互；然后运行./start-all.sh 启动驱动程序和控制器，如图所示。

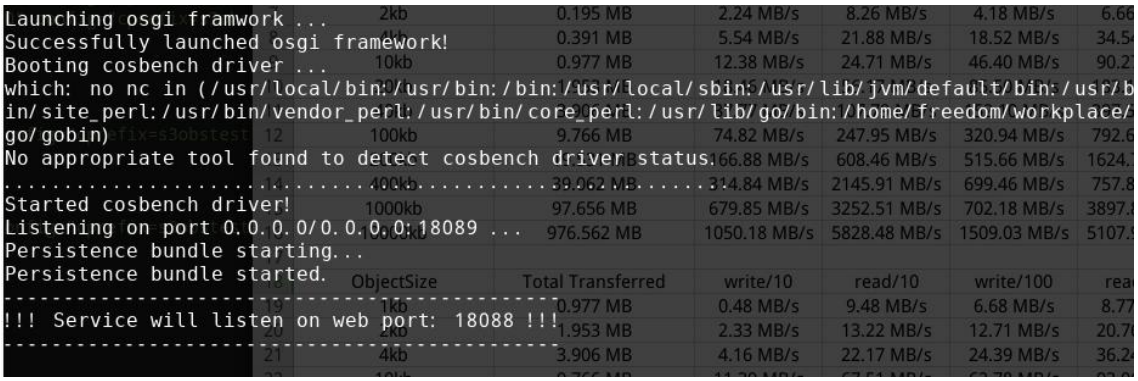

图 5.11 driver 监听端口

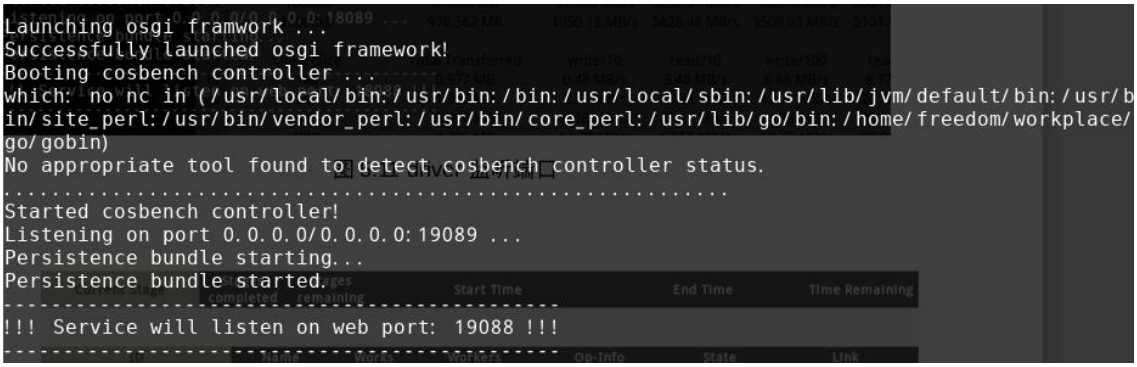

图 5.12 controller 监听端口

输入网址 http://127.0.1.1:19088/controller 后将负载提交，查看测试结果

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

ID	Name	Works	Workers	Op-Info	State	Link
w9-s1-init	init	1 wks	1 wkrs	init	completed	view details
w9-s2-prepare	prepare	8 wks	64 wkrs	prepare	completed	view details
w9-s3-8kb	8kb	1 wks	8 wkrs	read, write	completed	view details
w9-s4-16kb	16kb	1 wks	8 wkrs	read, write	completed	view details
w9-s5-32kb	32kb	1 wks	4 wkrs	read, write	completed	view details
w9-s6-64kb	64kb	1 wks	4 wkrs	read, write	completed	view details
w9-s7-128kb	128kb	1 wks	1 wkrs	read, write	completed	view details
w9-s8-256kb	256kb	1 wks	1 wkrs	read, write	completed	view details
w9-s9-512kb	512kb	1 wks	1 wkrs	read, write	completed	view details
w9-s10-1mb	1mb	1 wks	1 wkrs	read, write	completed	view details
w9-s11-cleanup	cleanup	1 wks	1 wkrs	cleanup	completed	view details
w9-s12-dispose	dispose	1 wks	1 wkrs	dispose	completed	view details

Performance Graph

图 5.13 样本用例测试结果

op1: read	23.21 kops	185.67 MB	7.97 ms	7.24 ms	776.54 op/s	6.21 MB/S	96.87%
op2: write	5.94 kops	47.55 MB	8.14 ms	8.11 ms	198.94 op/s	1.59 MB/S	100%
op1: read	22.36 kops	357.84 MB	7.82 ms	7.03 ms	746.35 op/s	11.94 MB/S	94.87%
op2: write	5.94 kops	94.98 MB	9.2 ms	9.13 ms	198.09 op/s	3.17 MB/S	100%
op1: read	24.53 kops	784.83 MB	3.36 ms	2.81 ms	817.56 op/s	26.16 MB/S	89.11%
op2: write	6.84 kops	218.72 MB	3.88 ms	3.76 ms	227.84 op/s	7.29 MB/S	100%
op1: read	23.41 kops	1.5 GB	3.14 ms	2.56 ms	780.32 op/s	49.94 MB/S	81.14%
op2: write	7.18 kops	459.84 MB	3.8 ms	3.56 ms	239.51 op/s	15.33 MB/S	100%
op1: read	17.98 kops	2.3 GB	1.31 ms	0.92 ms	599.45 op/s	76.73 MB/S	100%
op2: write	4.41 kops	564.99 MB	1.4 ms	0.93 ms	147.14 op/s	18.83 MB/S	100%
op1: read	13.57 kops	3.47 GB	1.67 ms	0.94 ms	452.43 op/s	115.82 MB/S	100%
op2: write	3.4 kops	869.63 MB	2.11 ms	1.2 ms	113.23 op/s	28.99 MB/S	100%
op1: read	9.18 kops	4.7 GB	2.38 ms	1 ms	305.91 op/s	156.63 MB/S	100%
op2: write	2.34 kops	1.2 GB	3.43 ms	1.71 ms	78.07 op/s	39.97 MB/S	100%
op1: read	5.69 kops	5.69 GB	3.76 ms	1.09 ms	189.7 op/s	189.7 MB/S	100%
op2: write	1.4 kops	1.4 GB	6.1 ms	2.76 ms	46.5 op/s	46.5 MB/S	100%
op1: cleanup -delete	128 ops	0 B	0.76 ms	0.76 ms	1319.59 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

图 5.14 样本用例测试结果

比较后可以发现，write 的成功率一直为 100%，而 read 的成功率在 80%-100%之间，而当测试用例的 workers 为 1 时，read 的成功率为 100%，而随着 size 的增大，吞吐率先增大后减小，带宽不断变大。为了测试 workers 数目对 read 成功率的影响，我改变了 workers 的数目以及块的大小，测试当块大小为 108kb 时，workers 数目对成功率的影响。

Stages

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

ID	Name	Works	Workers	Op-Info	State	Link
w25-s1-init	init	1 wks	1 wkrs	init	completed	view details
w25-s2-prepare	prepare	8 wks	64 wkrs	prepare	completed	view details
w25-s3-108kb	108kb	1 wks	1 wkrs	read, write	completed	view details
w25-s4-108kb	108kb	1 wks	2 wkrs	read, write	completed	view details
w25-s5-108kb	108kb	1 wks	4 wkrs	read, write	failed	view details
w25-s6-108kb	108kb	1 wks	8 wkrs	read, write	completed	view details
w25-s7-108kb	108kb	1 wks	16 wkrs	read, write	completed	view details
w25-s8-108kb	108kb	1 wks	32 wkrs	read, write	completed	view details
w25-s9-108kb	108kb	1 wks	64 wkrs	read, write	completed	view details
w25-s10-108kb	108kb	1 wks	128 wkrs	read, write	completed	view details
w25-s11-cleanup	cleanup	1 wks	1 wkrs	cleanup	completed	view details
w25-s12-dispose	dispose	1 wks	1 wkrs	dispose	completed	view details

图 5.15 运行结果 1

op1: read	16.4 kops	1.77 GB	1.43 ms	1.04 ms	546.78 op/s	59.05 MB/S	100%
op2: write	4.12 kops	444.74 MB	1.5 ms	1.02 ms	137.27 op/s	14.83 MB/S	100%
op1: read	21.91 kops	2.37 GB	1.57 ms	1.19 ms	730.46 op/s	78.89 MB/S	75.03%
op2: write	7.32 kops	791.1 MB	1.64 ms	1.25 ms	244.17 op/s	26.37 MB/S	100%
op1: read	20.53 kops	2.22 GB	3.15 ms	2.43 ms	684.21 op/s	73.89 MB/S	72.5%
op2: write	7.05 kops	761.29 MB	3.86 ms	3.43 ms	234.97 op/s	25.38 MB/S	100%
op1: read	18.44 kops	1.99 GB	7.51 ms	6.42 ms	617.17 op/s	66.65 MB/S	76.54%
op2: write	5.95 kops	642.71 MB	9 ms	8.52 ms	199.28 op/s	21.52 MB/S	100%
op1: read	18.68 kops	2.02 GB	14.76 ms	13.65 ms	626.39 op/s	67.65 MB/S	76.71%
op2: write	6.04 kops	652.64 MB	16.6 ms	16.13 ms	202.67 op/s	21.89 MB/S	100%
op1: read	17.89 kops	1.93 GB	33.79 ms	32.6 ms	604.35 op/s	65.27 MB/S	77.01%
op2: write	5.88 kops	634.72 MB	29.39 ms	28.89 ms	198.61 op/s	21.45 MB/S	100%
op1: read	18.27 kops	1.97 GB	54.98 ms	53.86 ms	622.61 op/s	67.24 MB/S	75.87%
op2: write	5.95 kops	642.91 MB	56.36 ms	54.09 ms	202.65 op/s	21.89 MB/S	100%
op1: read	18.19 kops	1.96 GB	117.06 ms	115.92 ms	634.23 op/s	68.5 MB/S	76.05%
op2: write	5.97 kops	645.91 MB	102.77 ms	98.95 ms	208.52 op/s	22.56 MB/S	100%
op1: cleanup -delete	128 ops	0 B	1.73 ms	1.73 ms	576.58 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

图 5.15 运行结果 2

如图可知，块大小均为 108k 时，workers 数目由 1 变为 128，吞吐率的趋势为先快速增加，workers 到达 2 时就已经是峰值，之后开始减小，减小到一定程度后稳定下来；读操作的带宽是先快速增大，后减小，然后趋于稳定；写操作的带宽先快速增加，后增加得越来越慢，到后来甚至出现减小的情况。

在 workers 数比较小时，服务器，也就是 PC 处理能力和带宽相对充足，workers 增加的情况下，读写性能可以快速增加。但是当 workers 达到饱和时，服务器出现拥挤，造成堵塞，此时性能增加开始不明显，但总体对比来说还是比单个 worker 要好的。

六、实验总结

本次实验中自己熟悉了 git 的操作以及对象存储的相关概念，自己动手对对象存储的一些性能指标进行了测试，这里我一开始是按照去年学长的方法用 cosbench 来做实验的，后来老师推荐我们用 s3bench 来进行操作，所以我就用了 s3bench，简单学习了如何写一个 shell 文件，for 循环，声明变量和数组等。也算是一个不大不小的收获。在实验中，自己遇到了一些问题，主要是不自己知道怎么利用测试出来的结果进行分析，所以一开始写报告的时候糊里糊涂的，不知道选哪个方向进行分析，后来确定了一下就好多了，思路也清晰了很多。另一个就是对测试的 workbench 的不了解了，用到的测试样例中一些参数不是很懂，所以必须得磕磕碰碰吃点亏才知道这个东西到底是什么。

好在把这些东西搞懂了之后也加深了自己对对象存储的理解，这次实验虽然时间不长，但收获还是蛮多的，知道了如何写一个简单的 shell 脚本，更加深入的去学习了一些东西，熟练了 github 的操作，老实说之前一直没有仔细的去了解 github 如何建立分支合并成一个大项目，这次实验让我知道了如何去操作。

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.