



2018 级

《物联网数据存储与管理》课程

实 验 报 告

姓 名 张骞

学 号 U201814573

班 号 物联网 1801 班

日 期 2021.06.27

目 录

| | |
|-------------------|----|
| 一、实验目的..... | 1 |
| 二、实验背景..... | 1 |
| 三、实验环境..... | 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..... | 8 |
| 五、实验总结..... | 9 |
| 参考文献..... | 10 |

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，是用来描述解决问题和处理离散单元的方法的通用术语，这些离散单元被称作对象。而对象存储系统，提供了高可靠、跨平台以及安全的数据共享的存储体系结构。

目前已经有了大量的基于块和基于文件的存储系统可供选择，基于块的存储系统，磁盘块通过底层存储协议访问，所有高级别的任务，像共享、锁定和安全通常由操作系统负责，即基于块的存储系统关心所有的底层的问题。而文件存储以文件为传输协议，以 TCP/IP 实现网络化存储，可扩展性好、价格便宜、用户易管理。但缺点在于读写速率低，传输速率慢。而对象存储，克服块存储与文件存储各自的缺点，发扬它俩各自的优点。块存储读写快，不利于共享，文件存储读写慢，利于共享。这就是我们在已有基于块和基于文件的存储系统的情况下，还需要对象存储的原因。

在我们的实验中，使用到了 Minio 作为服务端。Minio 是一个基于 Apache License v2.0 开源协议的对象存储服务。它兼容亚马逊 S3 云存储服务接口，非常适合于存储大容量非结构化的数据，例如图片、视频、日志文件、备份数据和容器/虚拟机镜像等，而一个对象文件从数 KB 到 5TB 都能够得到很好的支持。

三、实验环境

本实验的环境如下：

实验所用的操作系统为 Ubuntu20.04.2 LTS 版 64 位虚拟机环境，如图 3-1.



图 3-2 实验环境

软件环境：

对象存储客户端：采用 MC 进行测试

对象存储服务器端：采用 Minio

对象存储测试工具：采用 cosbench 进行测试

四、实验内容

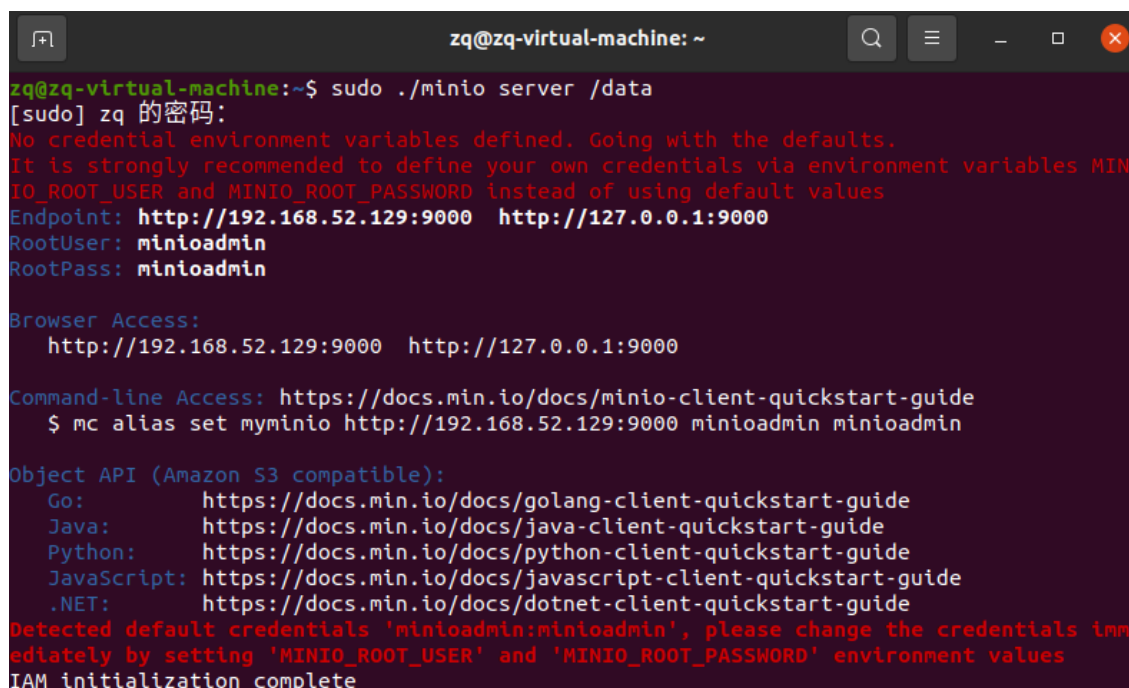
本次实验是对象存储实验入门实践，准备工作有 Git 与 GitHub 的学习，linux 虚拟机的安装以及 Java 或 Python 环境的准备。然后是选定对象存储服务端与客户端，选择 Minio 和 MC，在 linux 中运行 Minio 后用测试工具 Cosbench 进行测试。

4.1 对象存储技术实践

1. 采用 Minio 作为服务端

1) 下载 Minio 作为服务端。首先在 Minio 官网 <https://www.minio.io/downloads.html> 下载 Minio 和客户端 MC，然后使用 `chmod +x minio` 命令行添加权限。

2) 运行 Minio。在 linux 打开终端，在 root 权限下运行：`./minio server /data`。如图 4-1，可以看到服务器已经打开，并且可以通过端口 9000 访问。此时可以看到用户名和密码。

A terminal window titled 'zq@zq-virtual-machine: ~' showing the output of the command 'sudo ./minio server /data'. The output includes a password prompt for 'zq', a warning about missing credentials, endpoint information for http://192.168.52.129:9000 and http://127.0.0.1:9000, root user 'minioadmin', root password 'minioadmin', browser access URLs, command-line access instructions, and a list of object API quickstart guides for Go, Java, Python, JavaScript, and .NET. It also shows a warning about default credentials and 'IAM initialization complete'.

```
zq@zq-virtual-machine:~$ sudo ./minio server /data
[sudo] zq 的密码:
No credential environment variables defined. Going with the defaults.
It is strongly recommended to define your own credentials via environment variables MINIO_ROOT_USER and MINIO_ROOT_PASSWORD instead of using default values
Endpoint: http://192.168.52.129:9000 http://127.0.0.1:9000
RootUser: minioadmin
RootPass: minioadmin

Browser Access:
http://192.168.52.129:9000 http://127.0.0.1:9000

Command-line Access: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc alias set myminio http://192.168.52.129:9000 minioadmin minioadmin

Object API (Amazon S3 compatible):
Go: https://docs.min.io/docs/golang-client-quickstart-guide
Java: https://docs.min.io/docs/java-client-quickstart-guide
Python: https://docs.min.io/docs/python-client-quickstart-guide
JavaScript: https://docs.min.io/docs/javascript-client-quickstart-guide
.NET: https://docs.min.io/docs/dotnet-client-quickstart-guide
Detected default credentials 'minioadmin:minioadmin', please change the credentials immediately by setting 'MINIO_ROOT_USER' and 'MINIO_ROOT_PASSWORD' environment values
IAM initialization complete
```

图 4-1 运行 Minio

3) 在浏览器访问服务器。在浏览器中输入 `http://127.0.0.1:9000` 可以访问服务器，登录界面如图 4-2。确认登录后，可以看到界面如图 4-3。

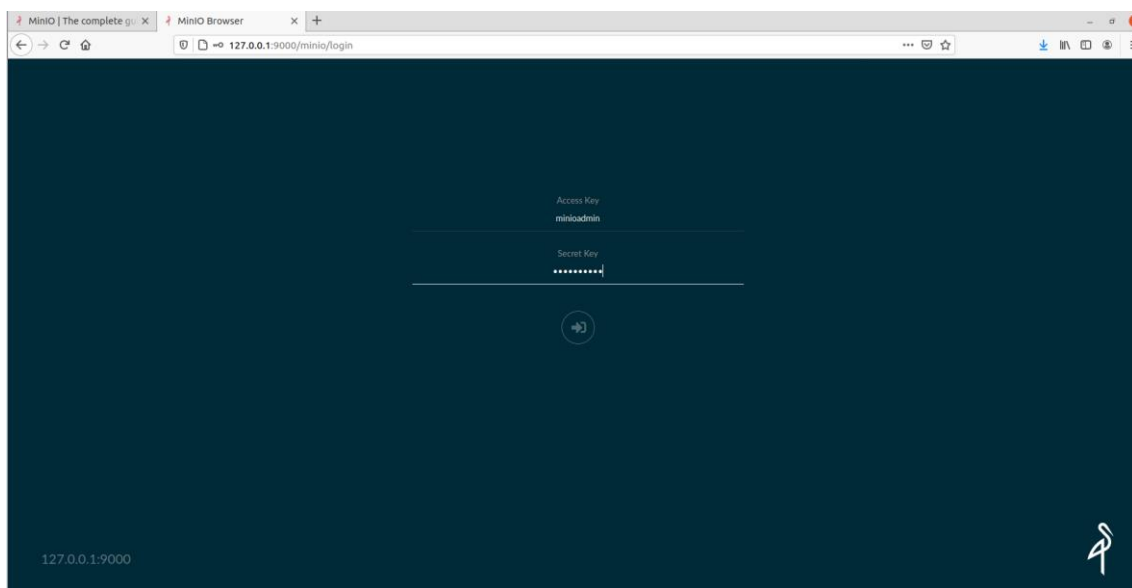


图 4-2 登录服务器

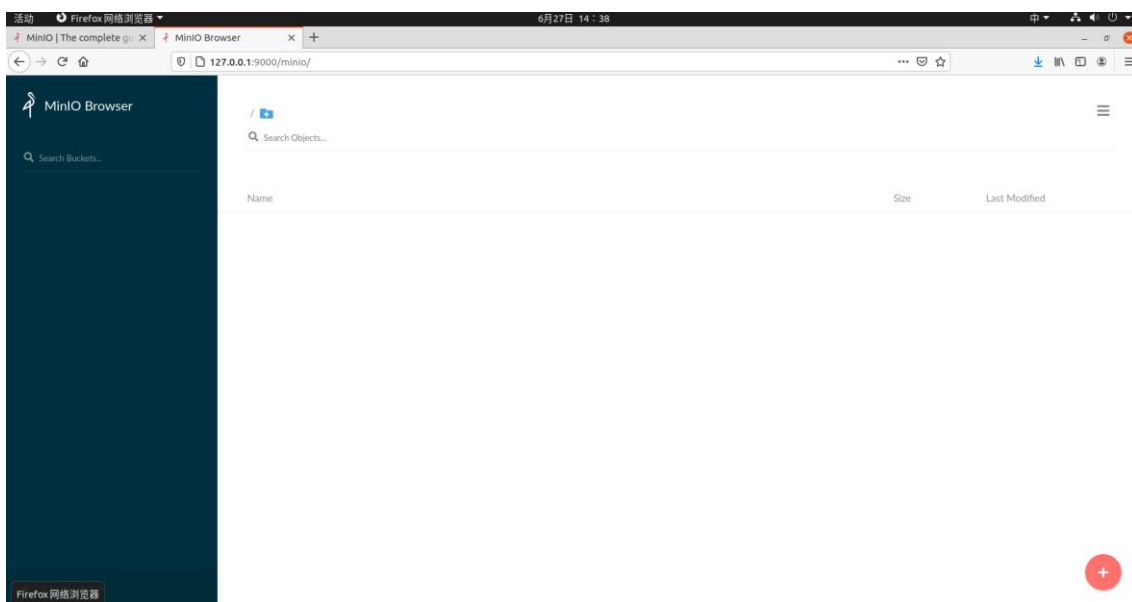


图 4-3 在浏览器中访问服务器

4) 在浏览器中可以添加存储对象。点击页面的+号按钮，可以选择新建一个仓库或者上传一个新的存储文件。在这里我们展示上传一个新的存储对象，如图 4-4.

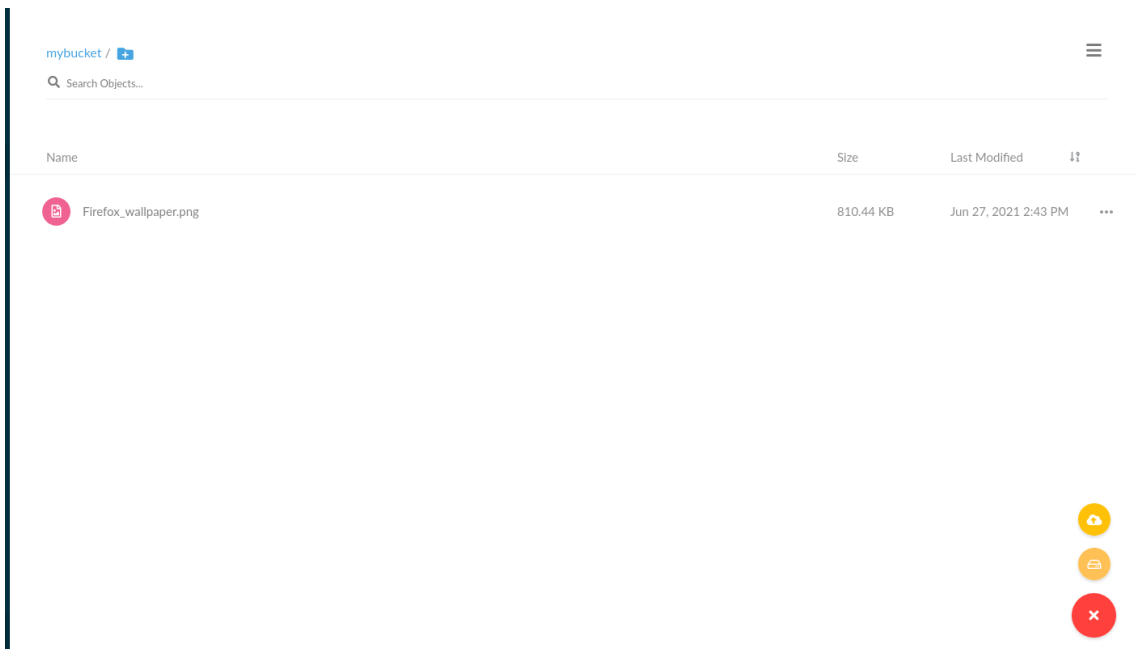


图 4-4 添加存储对象

5) 运行 MC 客户端进行对服务器的访问。重新打开一个终端，输入命令行：`./mc config host add myminio http://127.0.0.1:9000 minioadmin minioadmin`，使得可以通过 MC 访问服务器，如图 4-5。

```
zq@zq-virtual-machine:~$ chmod 777 mc
zq@zq-virtual-machine:~$ ./mc config host add myminio http://127.0.0.1:9000 minioadmin minioadmin
mc: Configuration written to `/home/zq/.mc/config.json`. Please update your access credentials.
mc: Successfully created `/home/zq/.mc/share`.
mc: Initialized share uploads `/home/zq/.mc/share/uploads.json` file.
mc: Initialized share downloads `/home/zq/.mc/share/downloads.json` file.
Added `myminio` successfully.
zq@zq-virtual-machine:~$
```

图 4-5 通过 MC 访问服务器

6) 使用 MC 对服务器进行增删。在命令行中输入`./mc ls myminio/mybucket` 可以查看服务器中的文件，如图 4-6。其中第一个文件是上面第四步中添加的。

```
zq@zq-virtual-machine:~$ ./mc ls myminio/mybucket
[2021-06-27 14:43:30 CST] 810KiB Firefox_wallpaper.png
zq@zq-virtual-machine:~$
```

图 4-6 MC 的 ls 命令

可以利用 MC 的 `rm` 命令删除服务器中的文件，如图 4-7，使用命令删除了这个 png 文件。如图 4-8，还可以使用 `mb` 命令来添加仓库或文件。

```
zq@zq-virtual-machine:~$ ./mc rm myminio/mybucket/Firefox_wallpaper.png
Removing `myminio/mybucket/Firefox_wallpaper.png`.
zq@zq-virtual-machine:~$
```

图 4-7 MC 的 rm 命令

```
zq@zq-virtual-machine:~$ ./mc mb myminio/mybucket2
Bucket created successfully 'myminio/mybucket2'.
```

图 4-8 MC 的 mb 命令

我们再通过 ls 命令查看，看是否真正的对服务器进行了删改。如图 4-9，分别对 test 和 test1 都使用了 ls 命令。可以看到，在 test 目录下的图片文件已经被成功的删除，而在服务器中成功的添加了 mybucket2 这个包。

```
zq@zq-virtual-machine:~$ ./mc ls myminio
[2021-06-27 14:43:14 CST]      0B mybucket/
[2021-06-27 15:02:18 CST]      0B mybucket2/
zq@zq-virtual-machine:~$
```

图 4-9 通过 ls 命令验证

最终我们在浏览器中访问服务器，可以看到当前的状态如图 4-10，这与我们在命令行中访问的结果是一致的。

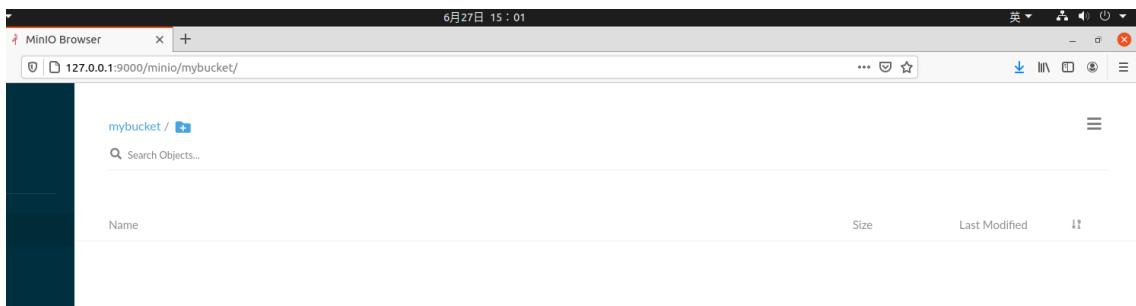


图 4-10 通过浏览器访问服务器

7) 使用测试工具 cosbench 进行测试。首先使用命令行 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；然后运行脚本启动驱动程序和控制器，使用端口 19088 进行监听，如图 4-11。

```
Launching osgi framework ...
Successfully launched osgi framework!
Booting cosbench controller ...
...
Starting    cosbench-log_0.4.2      [OK]
.
Starting    cosbench-tomcat_0.4.2    [OK]
Starting    cosbench-config_0.4.2    [OK]
Starting    cosbench-core_0.4.2      [OK]
Starting    cosbench-core-web_0.4.2  [OK]
Starting    cosbench-controller_0.4.2 [OK]
Starting    cosbench-controller-web_0.4.2 [OK]
Successfully started cosbench controller!
Listening on port 0.0.0.0/0.0.0.0:19089 ...
Persistence bundle starting...
Persistence bundle started.
-----
!!! Service will listen on web port: 19088 !!!
-----
```

图 4-11 启动驱动程序与控制器

然后用浏览器访问 <http://127.0.0.1:19088/controller/index.html> 可以进入 cosbench 的测试页面，如图 4-12。

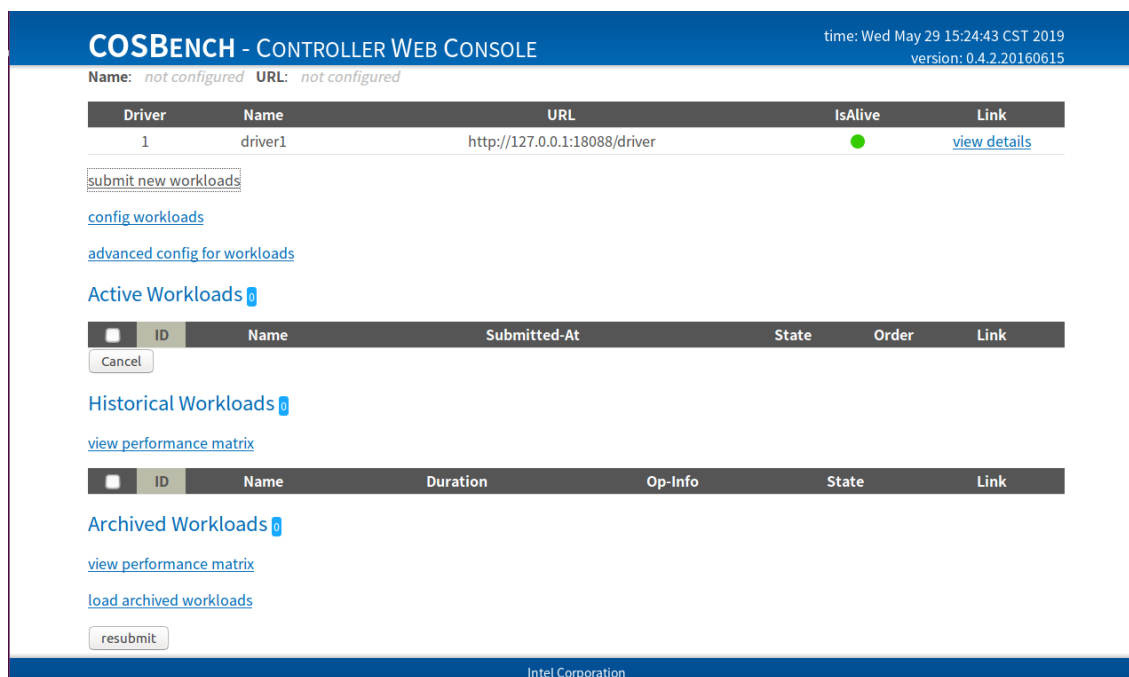


图 4-12 cosbench 测试页面

然后利用标准的测试样例，进行对服务器的标准测试，测试结果如图 4-13，具体的分析在下一节中给出。

| 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 | 8 ops | 64 KB | 487.5 ms | 487.5 ms | 16.79 op/s | 134.33 KB/s | 100% |
| op2: prepare -write | 8 ops | 128 KB | 349 ms | 348.25 ms | 23.4 op/s | 374.35 KB/s | 100% |
| op3: prepare -write | 8 ops | 256 KB | 466.12 ms | 465.25 ms | 17.48 op/s | 559.41 KB/s | 100% |
| op4: prepare -write | 8 ops | 512 KB | 524.62 ms | 521.25 ms | 16.02 op/s | 1.03 MB/s | 100% |
| op5: prepare -write | 8 ops | 1.02 MB | 654.12 ms | 632.88 ms | 12.54 op/s | 1.61 MB/s | 100% |
| op6: prepare -write | 8 ops | 2.05 MB | 759.75 ms | 729.25 ms | 11.12 op/s | 2.85 MB/s | 100% |
| op7: prepare -write | 8 ops | 4.1 MB | 604.38 ms | 573.88 ms | 14.14 op/s | 7.24 MB/s | 100% |
| op8: prepare -write | 8 ops | 8 MB | 693 ms | 610.5 ms | 12.58 op/s | 12.58 MB/s | 100% |
| op1: read | 13.51 kops | 108.06 MB | 12.77 ms | 12.61 ms | 450.35 op/s | 3.6 MB/s | 100% |
| op2: write | 3.29 kops | 26.29 MB | 19.98 ms | 17.57 ms | 109.55 op/s | 876.43 KB/s | 100% |
| op1: read | 14.05 kops | 224.74 MB | 11.65 ms | 11.1 ms | 468.32 op/s | 7.49 MB/s | 100% |
| op2: write | 3.47 kops | 55.57 MB | 21.7 ms | 20.08 ms | 115.8 op/s | 1.85 MB/s | 100% |
| op1: read | 13.97 kops | 447.1 MB | 5.59 ms | 5.12 ms | 465.94 op/s | 14.91 MB/s | 100% |
| op2: write | 3.47 kops | 111.07 MB | 11.96 ms | 11.53 ms | 115.75 op/s | 3.7 MB/s | 100% |
| op1: read | 13.04 kops | 834.56 MB | 5.92 ms | 5.39 ms | 434.78 op/s | 27.83 MB/s | 100% |
| op2: write | 3.14 kops | 201.28 MB | 13.5 ms | 12.26 ms | 104.86 op/s | 6.71 MB/s | 100% |
| op1: read | 5.92 kops | 758.27 MB | 3.27 ms | 3.15 ms | 197.47 op/s | 25.28 MB/s | 100% |
| op2: write | 1.47 kops | 187.65 MB | 7.13 ms | 6.41 ms | 48.87 op/s | 6.26 MB/s | 100% |
| op1: read | 6.41 kops | 1.64 GB | 2.79 ms | 2.62 ms | 213.7 op/s | 54.71 MB/s | 100% |
| op2: write | 1.61 kops | 412.16 MB | 7.46 ms | 5.8 ms | 53.67 op/s | 13.74 MB/s | 100% |
| op1: read | 5.48 kops | 2.8 GB | 2.97 ms | 2.72 ms | 182.58 op/s | 93.48 MB/s | 100% |
| op2: write | 1.41 kops | 720.38 MB | 9.7 ms | 6.09 ms | 46.9 op/s | 24.02 MB/s | 100% |
| op1: read | 3.97 kops | 3.97 GB | 3.3 ms | 2.86 ms | 132.54 op/s | 132.54 MB/s | 100% |
| op2: write | 1.08 kops | 1.08 GB | 15.52 ms | 6.1 ms | 35.98 op/s | 35.98 MB/s | 100% |
| op1: cleanup -delete | 128 ops | 0 B | 3.62 ms | 3.62 ms | 274.68 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 4-13 cosbench 测试结果

4.2 对象存储性能分析

1. 采用 Minio 作为服务端

首先，测试标准的 cosbench 结果如图 4-18。可以看到全部 12 个任务都完成了。

| ID | Name | Works | Workers | Op-Info | State | Link |
|-----------------|---------|-------|---------|-------------|-----------|------------------------------|
| w13-s1-init | init | 1 wks | 1 wkrs | init | completed | view details |
| w13-s2-prepare | prepare | 8 wks | 64 wkrs | prepare | completed | view details |
| w13-s3-8kb | 8kb | 1 wks | 8 wkrs | read, write | completed | view details |
| w13-s4-16kb | 16kb | 1 wks | 8 wkrs | read, write | completed | view details |
| w13-s5-32kb | 32kb | 1 wks | 4 wkrs | read, write | completed | view details |
| w13-s6-64kb | 64kb | 1 wks | 4 wkrs | read, write | completed | view details |
| w13-s7-128kb | 128kb | 1 wks | 1 wkrs | read, write | completed | view details |
| w13-s8-256kb | 256kb | 1 wks | 1 wkrs | read, write | completed | view details |
| w13-s9-512kb | 512kb | 1 wks | 1 wkrs | read, write | completed | view details |
| w13-s10-1mb | 1mb | 1 wks | 1 wkrs | read, write | completed | view details |
| w13-s11-cleanup | cleanup | 1 wks | 1 wkrs | cleanup | completed | view details |
| w13-s12-dispose | dispose | 1 wks | 1 wkrs | dispose | completed | view details |

There are 12 stages in this workload.

图 4-18 cosbench 测试结果（minio 作为服务端）

更加详细的情况如图 4-19，我们可以看到：

- 1) 写入和读取的成功率一直都是 100%；
- 2) 读取的 Bandwidth 比写入的 Bandwidth 大，这和我们平时了解的读取速度大于写入速度是一致的；
- 3) 随着每次读取与写入 size 的增大，Bandwidth 渐渐增大，而 Throughput 渐渐减小，相应的平均的休息时间与工作时间也减小。

| | | | | | | | |
|------------|------------|-----------|----------|----------|-------------|-------------|------|
| op1: read | 13.51 kops | 108.06 MB | 12.77 ms | 12.61 ms | 450.35 op/s | 3.6 MB/S | 100% |
| op2: write | 3.29 kops | 26.29 MB | 19.98 ms | 17.57 ms | 109.55 op/s | 876.43 KB/S | 100% |
| op1: read | 14.05 kops | 224.74 MB | 11.65 ms | 11.1 ms | 468.32 op/s | 7.49 MB/S | 100% |
| op2: write | 3.47 kops | 55.57 MB | 21.7 ms | 20.08 ms | 115.8 op/s | 1.85 MB/S | 100% |
| op1: read | 13.97 kops | 447.1 MB | 5.59 ms | 5.12 ms | 465.94 op/s | 14.91 MB/S | 100% |
| op2: write | 3.47 kops | 111.07 MB | 11.96 ms | 11.53 ms | 115.75 op/s | 3.7 MB/S | 100% |
| op1: read | 13.04 kops | 834.56 MB | 5.92 ms | 5.39 ms | 434.78 op/s | 27.83 MB/S | 100% |
| op2: write | 3.14 kops | 201.28 MB | 13.5 ms | 12.26 ms | 104.86 op/s | 6.71 MB/S | 100% |
| op1: read | 5.92 kops | 758.27 MB | 3.27 ms | 3.15 ms | 197.47 op/s | 25.28 MB/S | 100% |
| op2: write | 1.47 kops | 187.65 MB | 7.13 ms | 6.41 ms | 48.87 op/s | 6.26 MB/S | 100% |
| op1: read | 6.41 kops | 1.64 GB | 2.79 ms | 2.62 ms | 213.7 op/s | 54.71 MB/S | 100% |
| op2: write | 1.61 kops | 412.16 MB | 7.46 ms | 5.8 ms | 53.67 op/s | 13.74 MB/S | 100% |
| op1: read | 5.48 kops | 2.8 GB | 2.97 ms | 2.72 ms | 182.58 op/s | 93.48 MB/S | 100% |
| op2: write | 1.41 kops | 720.38 MB | 9.7 ms | 6.09 ms | 46.9 op/s | 24.02 MB/S | 100% |
| op1: read | 3.97 kops | 3.97 GB | 3.3 ms | 2.86 ms | 132.54 op/s | 132.54 MB/S | 100% |
| op2: write | 1.08 kops | 1.08 GB | 15.52 ms | 6.1 ms | 35.98 op/s | 35.98 MB/S | 100% |

图 4-19 详细测试结果（minio 作为服务端）

五、实验总结

此次试验是面向对象存储的入门实验，在实验中我了解了对象存储技术，明白了在已有基于块和基于文件的存储系统的情况下，我们仍然需要面向对象存储系统的原因。

实验中，我在 linux 环境中使用了 minio 作为服务端，使用测试工具 cosbench 进行了测试，受限于实验条件，进行操作的容量都比较小，在实际应用中肯定会有更大的存储吞吐量，不过总体来看速度和成功率都十分可观。

环境的配置有之前学长学姐的经验，照葫芦画瓢其实也还好，分析的过程使用控制变量的方法，对比分析 bit 和 worker 的变化对存取结果和带宽的影响。随着 workers 数目的增加，带宽增加，并发性提高带宽，数据吞吐率会减小，所需要的时间增多。最后采用 cosbench 测试是因为 cosbench 有图形界面，可视化程度较高

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.