



2020 级

《大数据存储系统与管理》课程

实 验 报 告

姓 名 武传航

学 号 U201915020

班 号 计算机 2001 班

日 期 2023.05.29

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.....	3
五、实验过程.....	4
5.1 对象存储技术实践.....	4
5.2 对象存储性能分析.....	6
六、实验总结.....	9
参考文献.....	9

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，分析性能问题，架设应用实践。

二、实验背景

对象存储（Object Storage Service, OSS），也叫基于对象的存储，是一种解决和处理离散单元的方法，可提供基于分布式系统之上的对象形式的数据存储服务。对象存储和我们经常接触到的块和文件系统等存储形态不同，它提供 RESTful API 数据读写接口及丰富的 SDK 接口，并且常以网络服务的形式提供数据的访问。

在我们的实验中，使用到了 MinIO 作为服务端。MinIO 是一款高性能、分布式的对象存储系统。它是一款软件产品，可以 100% 的运行在标准硬件。即 X86 等低成本机器也能够很好的运行 MinIO。MinIO 与传统的存储和其他的对象存储不同的是：它一开始就针对性能要求更高的私有云标准进行软件架构设计。因为 MinIO 一开始就只为对象存储而设计。所以它采用了更易用的方式进行设计，它能实现对象存储所需要的全部功能，在性能上也更加强劲，它不会为了更多的业务功能而妥协，失去 MinIO 的易用性、高效性。这样的结果所带来的好处是：它能够更简单的实现具有弹性伸缩能力的原生对象存储服务。

三、实验环境

CPU	AMD Ryzen 7 6800H with Radeon Graphics 3.20 GHz
内存	16GB
操作系统	Windows11 家庭中文版
Python	3.10.9
服务端	MinIO
客户端	MinIO
评测工具	S3bench

四、实验内容

本次实验主要内容如下：

1. 搭建系统环境，包括代码版本控制系统 `Git`，各种语言运行环境。
2. 对对象存储系统进行实践，采用 `minio/mc` 配置服务器端和客户端，并进行简单的创建或删除 `bucket`、上传或删除文件等操作。
3. 对对象存储系统进行测试，采用 `s3bench` 进行负载测试。

4.1 对象存储技术实践

1. 配置 `minio` server 端，通过浏览器登陆 `127.0.0.1` 查看效果，并熟悉各项操作。
2. 配置 `minio` 客户端 `mc`，在命令行下输入命令实现创建、删除 `bucket` 等操作，并通过访问网址查看新建及删除的结果。

4.2 对象存储性能分析

1. 读写性能对比。
2. 修改脚本范例中 `numClients` 参数的值，从而修改同一时间产生的请求数，即并发数，分析并发数对存储性能的影响。
3. 修改脚本范例中 `numSamples` 参数的值，从而修改 `workers` 的数量，分析 `workers` 数量对存储性能的影响。
4. 修改脚本范例中 `objectSize` 参数的值，从而修改文件的大小，分析块文件大小对存储性能的影响。

五、实验过程

5.1 对象存储技术实践

(1) 下载 Windows 系统对应版本的 minio server 和 minio client。默认启动会显示”Detected default credentials 'minioadmin:minioadmin', we recommend that you change these values with 'MINIO_ROOT_USER' and 'MINIO_ROOT_PASSWORD' environment variables” ， 将 环 境 变 量 'MINIO_ROOT_USER' 和 'MINIO_ROOT_PASSWORD' 设置为 zhigehust，然后运行 minio.exe，得到如图 1 所示结果。

```
D:\minio>minio.exe server minioserver
MinIO Object Storage Server
Copyright: 2015-2023 MinIO, Inc.
License: GNU AGPLv3 <https://www.gnu.org/licenses/agpl-3.0.html>
Version: RELEASE.2023-05-27T05-56-19Z (go1.19.9 windows/amd64)

Status:          1 Online, 0 Offline.
S3-API: http://10.11.177.172:9000 http://127.0.0.1:9000
RootUser: zhigehust
RootPass: zhigehust

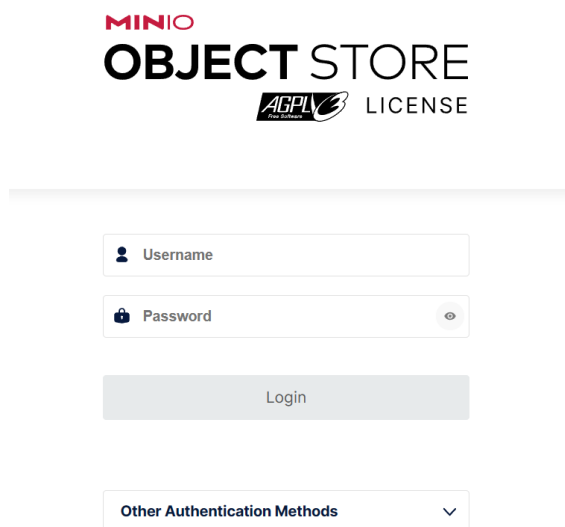
Console: http://10.11.177.172:63403 http://127.0.0.1:63403
RootUser: zhigehust
RootPass: zhigehust

Command-line: https://min.io/docs/minio/linux/reference/minio-mc.html#quickstart
$ mc.exe alias set myminio http://10.11.177.172:9000 zhigehust zhigehust

Documentation: https://min.io/docs/minio/linux/index.html
Warning: The standard parity is set to 0. This can lead to data loss.
```

图 1 minio server

(2) 在浏览器输入 <http://127.0.0.1:9000> 访问服务器，用户名和密码已更改为 zhigehust。登陆界面如图 2。确认登陆后，可以看到界面如图 3。



The image shows the MinIO Object Store login interface. At the top, it displays the 'MINIO OBJECT STORE' logo and the 'AGPLv3 LICENSE' logo. Below the logo, there is a login form with two input fields: 'Username' and 'Password'. The 'Password' field has a toggle icon to show or hide the password. Below the input fields is a 'Login' button. At the bottom, there is a dropdown menu labeled 'Other Authentication Methods' with a downward arrow.

图 2 登录界面

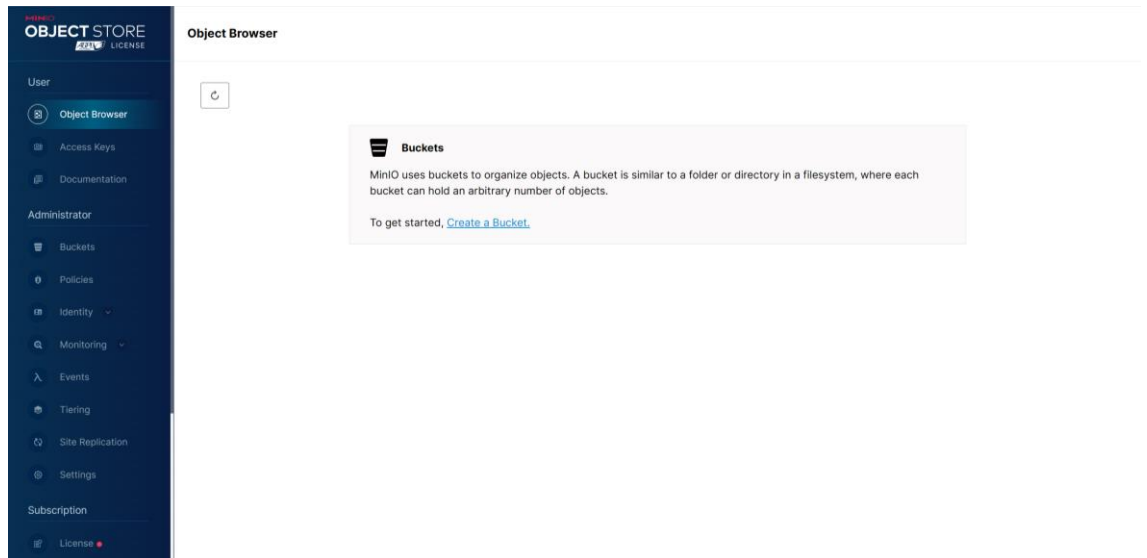


图 3 服务器初始界面

(3) 在 Buckets 界面点击 Create Bucket 创建一个名为 zhige 的 bucket，创建完成后如图 4 所示。

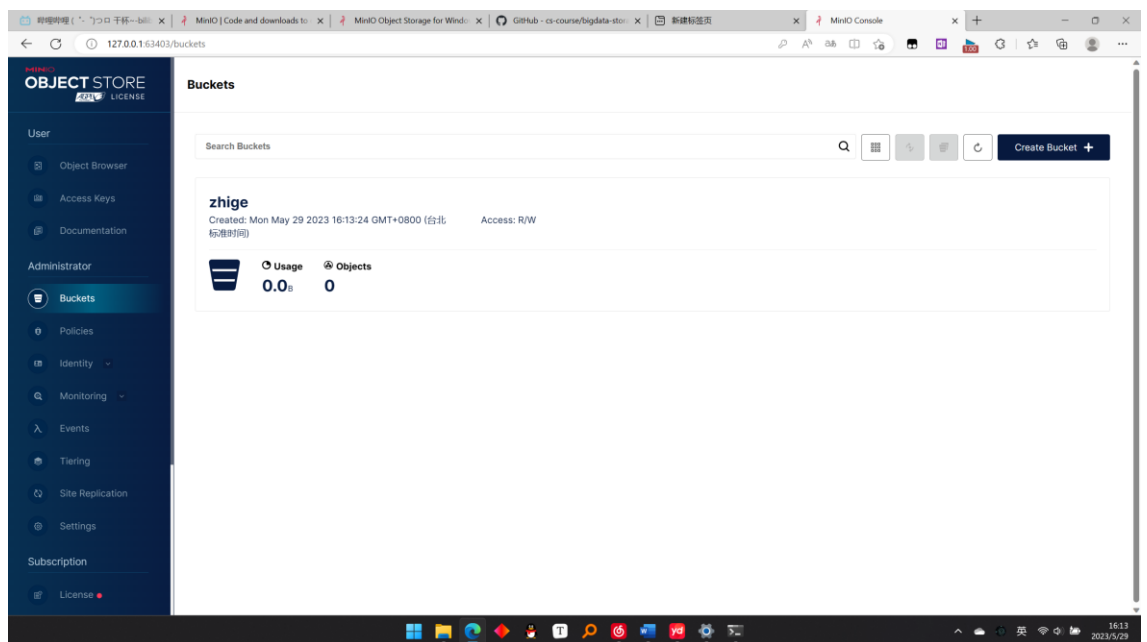


图 4 新建 bucket

(4) 在命令行下运行 mc.exe, 添加一个名为 cs 的存储服务。查看 cs 中的 bucket 和对象，可以看到之前用浏览器创建的 zhige。通过指令，在客户端创建一个名为 hust 的新 bucket，然后可以看到 hust 已被成功创建并加入存储服务中。

```
C:\WINDOWS\system32\cmd
Microsoft Windows [版本 10.0.22621.1702]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\zhige>d:

D:\>cd minio

D:\minio>mc config host add cs http://127.0.0.1:9000 zhigehust zhigehust --api s3v4
Added 'cs' successfully.

D:\minio>mc ls cs
[2023-05-29 16:13:24 CST]    0B zhige/

D:\minio>mc mb cs/hust
Bucket created successfully 'cs/hust'.

D:\minio>mc ls cs
[2023-05-29 16:18:45 CST]    0B hust/
[2023-05-29 16:13:24 CST]    0B zhige/

D:\minio>
```

图 5 运行客户端 mc

回到网页可以看到新创建的名为 hust 的 bucket，如图 6 所示。

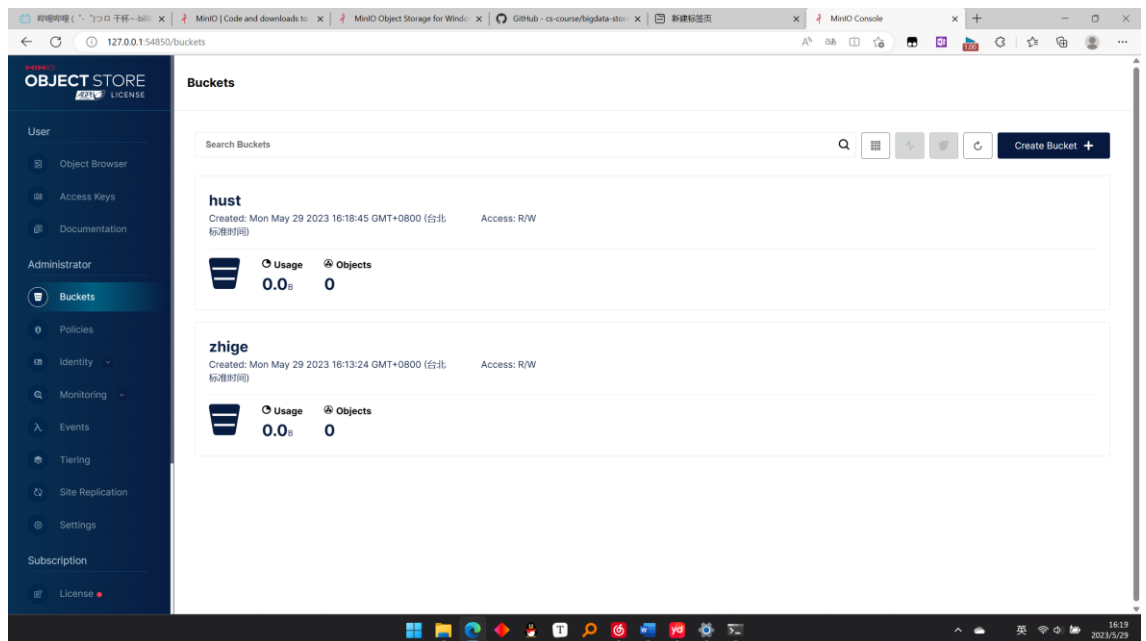
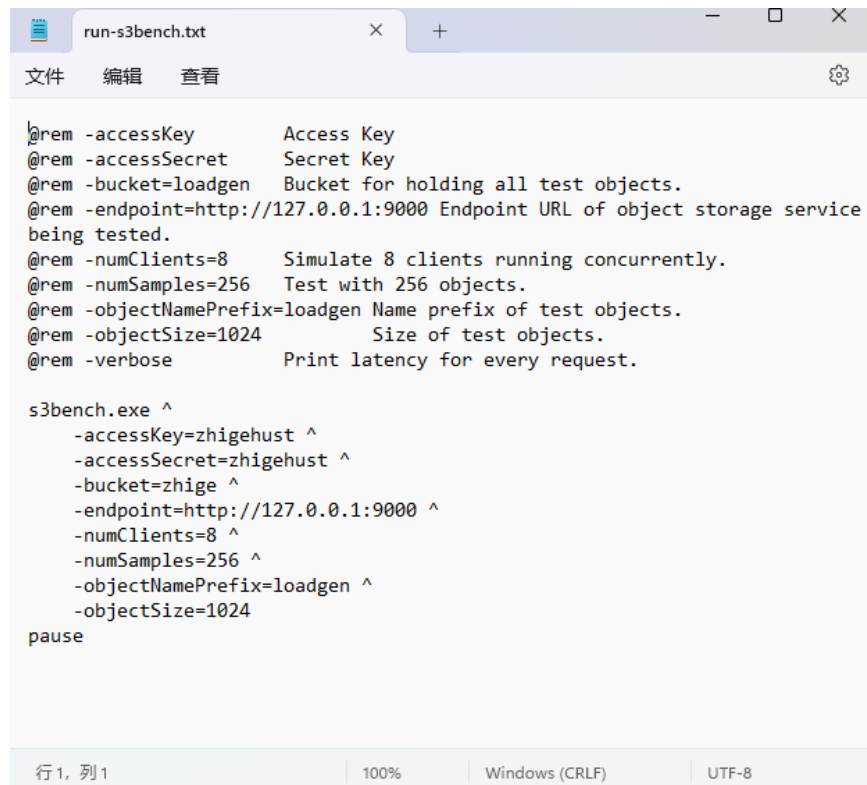


图 6 网页显示结果

5.2 对象存储性能分析

使用测试工具 s3bench 进行测试，先修改脚本，将 accessKey 和 accessSecret 修改为 zhigehust，将 bucket 的值改为 zhige，修改后如图 8 所示。



```
run-s3bench.txt
文件 编辑 查看

@rem -accessKey      Access Key
@rem -accessSecret   Secret Key
@rem -bucket=loadgen  Bucket for holding all test objects.
@rem -endpoint=http://127.0.0.1:9000 Endpoint URL of object storage service
being tested.
@rem -numClients=8    Simulate 8 clients running concurrently.
@rem -numSamples=256  Test with 256 objects.
@rem -objectNamePrefix=loadgen Name prefix of test objects.
@rem -objectSize=1024  Size of test objects.
@rem -verbose        Print latency for every request.

s3bench.exe ^
    -accessKey=zhigehust ^
    -accessSecret=zhigehust ^
    -bucket=zhige ^
    -endpoint=http://127.0.0.1:9000 ^
    -numClients=8 ^
    -numSamples=256 ^
    -objectNamePrefix=loadgen ^
    -objectSize=1024
pause

行 1, 列 1    100%    Windows (CRLF)    UTF-8
```

图 8 测试脚本

将 s3bench.exe 和 run-s3bench.cmd 放在同一目录下。运行 run-s3bench.cmd 脚本，得到结果如图 9 所示。

```
C:\WINDOWS\system32\cmd. X + v

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           zhige
objectNamePrefix: loadgen
objectSize:       0.0010 MB
numClients:       8
numSamples:       256
verbose:          %!d(bool=false)

Results Summary for Write Operation(s)
Total Transferred: 0.250 MB
Total Throughput:  1.75 MB/s
Total Duration:    0.142 s
Number of Errors:  0
-----
Write times Max:      0.010 s
Write times 99th %ile: 0.009 s
Write times 90th %ile: 0.005 s
Write times 75th %ile: 0.005 s
Write times 50th %ile: 0.004 s
Write times 25th %ile: 0.004 s
Write times Min:      0.002 s

Results Summary for Read Operation(s)
Total Transferred: 0.250 MB
Total Throughput:  7.38 MB/s
Total Duration:    0.034 s
Number of Errors:  0
-----
Read times Max:       0.003 s
Read times 99th %ile: 0.003 s
Read times 90th %ile: 0.002 s
Read times 75th %ile: 0.001 s
Read times 50th %ile: 0.001 s
Read times 25th %ile: 0.001 s
Read times Min:       0.000 s
```

图 9 脚本运行结果

将对象大小分别取为 1024, 2048, 3072, 将客户端数分别取为 8, 16, 24, 将样本数量分别取为 256, 512, 768, 得到共如图所示 27 组数据。

	A	B	C	D	E	F	G	H	I	
1	对象大小	客户端数	样本数量	写吞吐率	写99延迟	写90延迟	读吞吐率	读99延迟	读90延迟	
2	1024	8	256	1.76	0.021	0.005	7.83	0.002	0.002	
3	1024	8	512	1.71	0.013	0.006	8.38	0.002	0.002	
4	1024	8	768	1.76	0.018	0.005	8.04	0.003	0.002	
5	1024	16	256	1.7	0.017	0.012	8.17	0.004	0.003	
6	1024	16	512	1.78	0.017	0.01	8.48	0.004	0.003	
7	1024	16	768	1.82	0.02	0.009	7.99	0.006	0.003	
8	1024	24	256	1.66	0.027	0.016	8.46	0.007	0.004	
9	1024	24	512	1.7	0.033	0.014	8.71	0.006	0.004	
10	1024	24	768	1.73	0.025	0.015	8.6	0.008	0.004	
11	2048	8	256	2.85	0.024	0.007	15.51	0.002	0.002	
12	2048	8	512	3.41	0.01	0.006	17.09	0.002	0.001	
13	2048	8	768	3.44	0.018	0.006	16.74	0.002	0.002	
14	2048	16	256	3.54	0.022	0.009	17.21	0.004	0.003	
15	2048	16	512	3.46	0.019	0.01	16.99	0.004	0.003	
16	2048	16	768	3.44	0.021	0.01	18.09	0.004	0.003	
17	2048	24	256	3.47	0.025	0.017	16.8	0.006	0.004	
18	2048	24	512	3.46	0.022	0.014	16.64	0.007	0.004	
19	2048	24	768	3.48	0.021	0.014	17.51	0.012	0.005	
20	3072	8	256	5.58	0.011	0.005	24.34	0.002	0.002	
21	3072	8	512	5.49	0.008	0.005	24.94	0.002	0.002	
22	3072	8	768	4.99	0.021	0.006	25.06	0.002	0.002	
23	3072	16	256	4.68	0.022	0.018	26.54	0.004	0.003	
24	3072	16	512	5.28	0.016	0.01	27.99	0.004	0.002	
25	3072	16	768	5.06	0.028	0.01	20.94	0.006	0.003	
26	3072	24	256	4.85	0.03	0.019	24.42	0.007	0.005	
27	3072	24	512	5.02	0.036	0.016	19.47	0.01	0.006	
28	3072	24	768	5.13	0.026	0.014	24.36	0.007	0.005	
29										

图 10 不同条件下的吞吐率和延迟

从表中可以看到服务器的读取速率大于写入速率；随着客户端数的提高，吞吐率呈下降趋势；吞吐率不因样本数量的变化改变；当对象大小提高时，吞吐率提高，且上升幅度较大。

读写延迟不会因为对象大小以及样本数量的改变而改变，但当客户端数增加时，读写延迟随之提高；

六、实验总结

本次实验是一次面向对象存储的基础性实验。在进行实验的过程中，我对对象存储基础有了基本的了解。

在本次实验中，我是用 minio 搭建了对对象存储系统中的服务端和客户端，使用 s3bench 对存储系统的性能进行了一定程度的分析。缺憾在于测试的数据量相对较少，无法对归纳得到的结果形成有力的佐证。

参考文献

- [1] 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.
- [2] ARNOLD J. OpenStack Swift[M]. O' Reilly Media, 2014.
- [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] Dean J, Barroso L A. Association for Computing Machinery, 2013. The Tail at Scale[J]. Commun. ACM, 2013, 56(2): 74 - 80.

- [5] Delimitrou C, Kozyrakis C. Association for Computing Machinery, 2018. Amdahl's Law for Tail Latency[J]. Commun. ACM, 2018, 61(8): 65 – 72.