2019 级

《物联网数据存储与管理》课程

实 验 报 告

姓　名 黄宇杰

学　号 U201914861

班　号 CS1902 班

日　期 2023.5.29

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，分析性能问题，架设应用实践。

二、实验背景

MinIO 是一个开源的高性能对象存储服务器，适用于大规模无结构数据的存储。它的设计充分考虑了云环境的适应性，并提供了简单的 API 和一些附加的功能，如数据保护、数据版本化、事件通知等。

MinIO 采用了分布式架构，支持跨多个硬件设备和位置分布存储，从而提供了高可用性和数据冗余。这使得在硬件故障或者网络中断的情况下，数据仍然可以被访问。

MinIO 的特性主要包括：
1. 高性能：MinIO 针对大规模数据的读写操作进行了优化，可以提供高吞吐量和低延迟的数据访问。
2. 兼容 S3：MinIO 兼容亚马逊的 S3 API，可以无缝对接大多数使用 S3 的应用。
3. 安全性：MinIO 提供了细粒度的访问控制和数据加密功能，保证了数据的安全。
4. 易用性：MinIO 提供了一个易于使用的界面，并可以方便地通过 Docker 和 Kubernetes 进行部署。

MinIO 的应用场景非常广泛，可以用于：
1. 备份和归档：存储大量的备份数据，如数据库备份，系统备份等。
2. 大数据分析：提供了高性能的数据存储，适用于大数据处理和分析，如 Hadoop 和 Spark 等。
3. AI 和机器学习：用于存储大规模的训练数据，以及机器学习模型。
4. 内容分发：提供了高性能的数据传输，可以用于视频流、音频流、图片和其他大型文件的分发。
5. Web 应用：提供了一个可扩展的后端存储，可以用于存储用户生成的内容，如图片、视频等。
6. DevOps：可以为持续集成/持续部署（CI/CD）提供可靠的对象存储。

三、实验环境

硬件环境	CPU	12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz
	内存	16.0 GB
软件环境	操作系统	Windows Subsystem Linux Ubuntu 20.04 LTS
	测试工具	MinIO mc s3bench

四、实验内容

1.熟练运用 Git 和 GitHub
2.运用 MinIO 和 mc 创建服务端和客户端，访问和管理数据。
3.使用 s3bench 进行性能测试。

4.1 对象存储技术实践

使用 MinIO 搭建服务端，访问浏览器查看效果。用 mc 配置客户端进行简单操作并查看效果。

4.2 对象存储性能分析

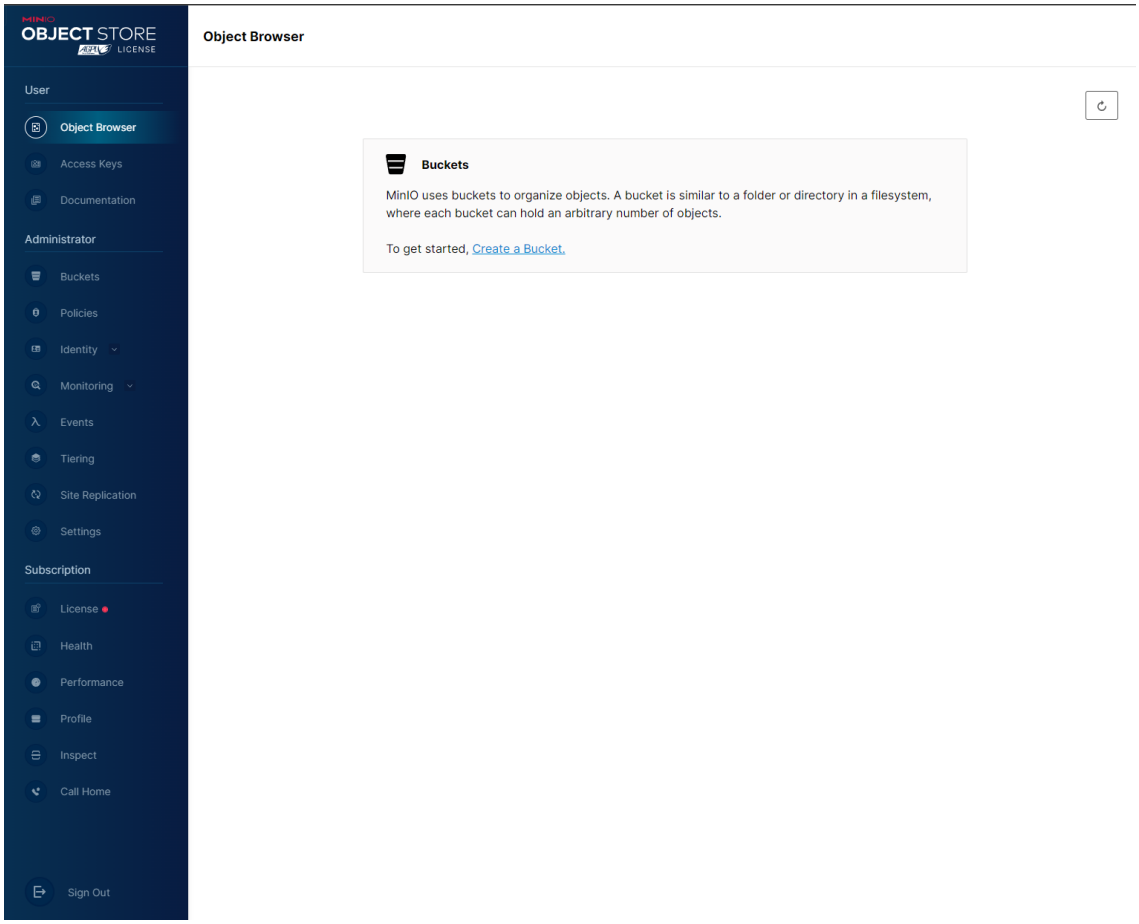
修改脚本参数，分析脚本参数对存储性能的影响。

五、实验过程

启动 Windows Terminal，打开 WSL。
下载并安装 MinIO server 和 MinIO client。
运行 MinIO server，显示如下画面：

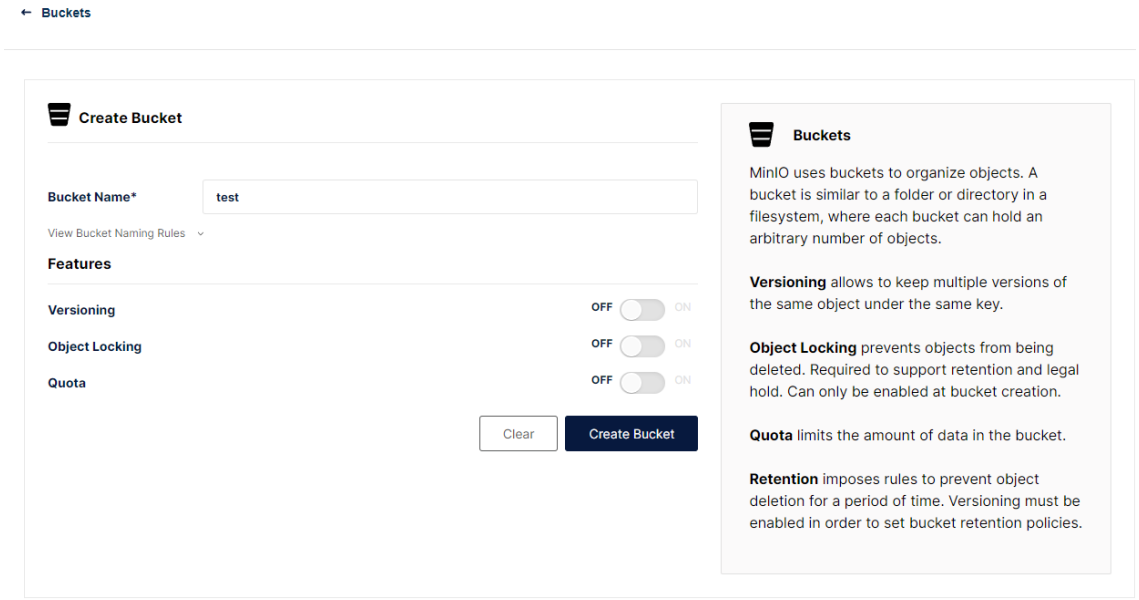
```
 root@Y9000P  ~  minio server /data
Formatting 1st pool, 1 set(s), 1 drives per set.
WARNING: Host local has more than 0 drives of set. A host failure will result in data becomi
ng unavailable.
WARNING: Detected default credentials 'minioadmin:minioadmin', we recommend that you change
these values with 'MINIO_ROOT_USER' and 'MINIO_ROOT_PASSWORD' environment variables
MinIO Object Storage Server
Copyright: 2015-2023 MinIO, Inc.
License: GNU AGPLv3 <https://www.gnu.org/licenses/agpl-3.0.html>
Version: RELEASE.2023-04-20T17-56-55Z (go1.20.3 linux/amd64)

Status:         1 Online, 0 Offline.
S3-API: http://172.26.62.194:9000  http://127.0.0.1:9000
RootUser: minioadmin
RootPass: minioadmin

Console: http://172.26.62.194:35875 http://127.0.0.1:35875
RootUser: minioadmin
RootPass: minioadmin

Command-line: https://min.io/docs/minio/linux/reference/minio-mc.html#quickstart
   $ mc alias set myminio http://172.26.62.194:9000 minioadmin minioadmin

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

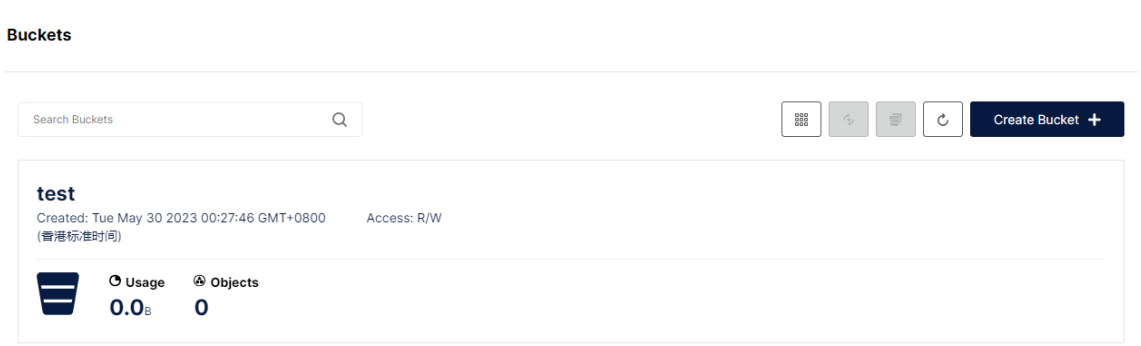
在浏览器输入 http://127.0.0.1:9000 进入登录界面并使用初始用户名和密码进行登录。



界面如上图所示。

创建 bucket：





另起一个 WSL 终端，为存储服务创建好别名。

```
root@Y9000P ~ mc alias set myminio http://127.0.0.1:9000 minioadmin minioadmin
Added `myminio` successfully.
```

展示所有的 bucket：

```
root@Y9000P ~ mc ls myminio
[2023-05-30 00:27:46 HKT]     0B test/
```

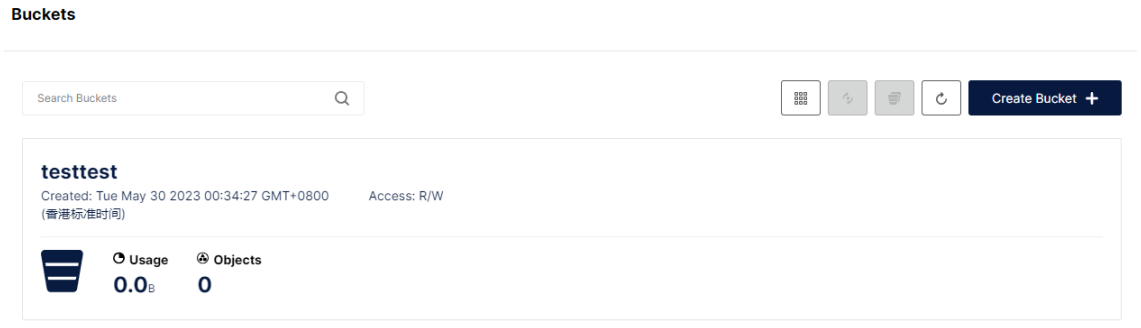
使用命令 mb，创建新的 bucket：

```
root@Y9000P ~ mc mb myminio/testtest
Bucket created successfully `myminio/testtest`.
```

再使用 lb 对第一个创建的 bucket 进行删除：

```
root@Y9000P ~ mc rb myminio/test
Removed `myminio/test` successfully.
```

网页控制台可见结果如下：



接下来使用 s3bench 进行存储性能测试。

首先下载 s3bench，并编写相应的脚本：

```
root@Y9000P ~ cat bench.sh
s3bench -accessKey=minioadmin -accessSecret=minioadmin -bucket=loadgen -endpoint=http://127.
0.0.1:9000 -numClients=4 -numSamples=1024 -objectNamePrefix=loadgen -objectSize=1024#
```

为脚本创建所需的 loadgen bucket 之后，运行该脚本可以看到结果如下图所示：

```
 root@Y9000P  ~  sh ./bench.sh
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.0010 MB
numClients:       4
numSamples:       1024
verbose:       %!d(bool=false)


Generating in-memory sample data ... Done (38.129µs)

Running Write test ...

Running Read test ...

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.0010 MB
numClients:       4
numSamples:       1024
verbose:       %!d(bool=false)


Results Summary for Write Operation(s)
Total Transferred: 1.000 MB
Total Throughput:  1.48 MB/s
Total Duration:    0.673 s
Number of Errors:  0
--------------------------------------
Write times Max:       0.010 s
Write times 99th %ile: 0.007 s
Write times 90th %ile: 0.003 s
Write times 75th %ile: 0.003 s
Write times 50th %ile: 0.002 s
Write times 25th %ile: 0.002 s
Write times Min:       0.002 s


Results Summary for Read Operation(s)
Total Transferred: 1.000 MB
Total Throughput:  5.19 MB/s
Total Duration:    0.193 s
Number of Errors:  0
--------------------------------------
Read times Max:       0.003 s
```

接下来所需要做的仅仅只需要改变参数测得对应数据的变化即可。为方便测量，采用 throughput 和 duration 作为性能观测指标。

(1) 改变 Clients（并发数）（numsamples=1024,objectsize=1024）测量性能：

Clients	Total Throughput Write (MB/s)	Total Duration Write (s)	Total Throughput Read (MB/s)	Total Duration Read (s)
4	1.48	0.673	5.19	0.193
8	2.37	0.423	8.13	0.123
16	3.37	0.297	10.28	0.097
32	3.80	0.263	12.28	0.081
64	3.06	0.327	15.64	0.064

可以看出随着 clients 的增长，吞吐量不断上升，总用时不断下降。但是，当并发量达到 64 时，写性能却反而下降了，推测是因为进程间切换导致的开销问题。

(2) 修改 numSamples(numclients=8,objectsize=1024)进行性能测试：

NumSamples	Total Throughput Write (MB/s)	Total Duration Write (s)	Total Throughput Read (MB/s)	Total Duration Read (s)
512	2.35	0.212	7.94	0.063
1024	2.37	0.423	8.13	0.123
2048	2.61	0.765	8.37	0.239
4096	2.45	1.631	8.40	0.476

可以看出随着 samples 的增长，吞吐率基本不变，总耗时约等于线性上升。

(3) 修改 objectSize（numclients=8,numsamples=1024)进行性能测试：

ObjectSize	Total Throughput Write (MB/s)	Total Duration Write (s)	Total Throughput Read (MB/s)	Total Duration Read (s)
512	1.24	0.405	3.93	0.127
1024	2.37	0.423	8.13	0.123
2048	4.78	0.418	17.44	0.115
4096	9.42	0.425	32.43	0.123

可以看出随着文件大小的提升，总耗时基本不变，吞吐量约等于线性上升。

六、实验总结

本次实验在 WSL 上进行，选择的是相对而言更为简单的 MinIO 和 s3bench，实验较为顺利。不过在 WSL 的使用上以及相关工具的下载上遇到了一些问题，除此之外还在 Windows 宿主机与 WSL 传文件时遇到了困难，好在最后都顺利解决了。

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.