2019 级

《大数据存储系统与管理》课程

实 验 报 告

姓　名　钟逸

学　号　U201914978

班　号　计算机 1903 班

日　期　2023.05.28

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

当今世界，物联网快速演化，持续制造着各种各样的数据。面对庞大且复杂的大数据，对象存储技术应运而生，基于对象存储技术的设备就是对象存储设备（Object-based Storage Device，OSD），其在当前工业界得到了广泛应用。

对象存储提供了具有高性能、高可靠性、高安全性且跨平台的数据共享的存储体系结构。对象存储的核心就是将数据的读写和元数据分离，存储系统的每个对象存储设备都能够自动管理它的数据分布。

MinIO 是一个高性能的分布式对象存储系统。在行业标准硬件上运行，并且100%开源。

三、实验环境

表 1 实验环境配置

	配　置	版 本
硬件环境	OS	Windows 10
	CPU	Intel(R) Core(TM) i7-9750H
	RAM	16.0 GB
软件环境	Server	MINIO SERVER
	Client	MINIO CLIENT
	Tool	S3 Bench

本实验具体环境配置如表 1 所示。

四、实验内容

实验分为三个部分：1. 熟悉基础环境；2. 实践对象存储；3. 进行初步研究。下面分为两个小节对实验内容进行具体阐述。

4.1 对象存储技术实践

（1）下载 windows 版本 MinIO 服务端以及客户端；
（2）利用命令脚本配置服务端以及客户端；
（3）基础功能尝试运行；
（4）下载 S3 Bench 以及命令脚本；
（5）编辑命令脚本，准备进行性能测试。

4.2 对象存储性能分析

（1）测试客户端数量对传输的影响，记录实验结果；
（2）测试对象数量对传输的影响，记录实验结果；
（3）测试对象大小对传输的影响，记录实验结果；
（4）根据实验结果进行性能分析。

五、实验过程

5.1 对象存储技术实践

（1）MinIO Server：

首先安装 win 版 MinIO Server，然后下载 run-minio.cmd，运行命令脚本。然后可加载网站 https://localhost:9090。

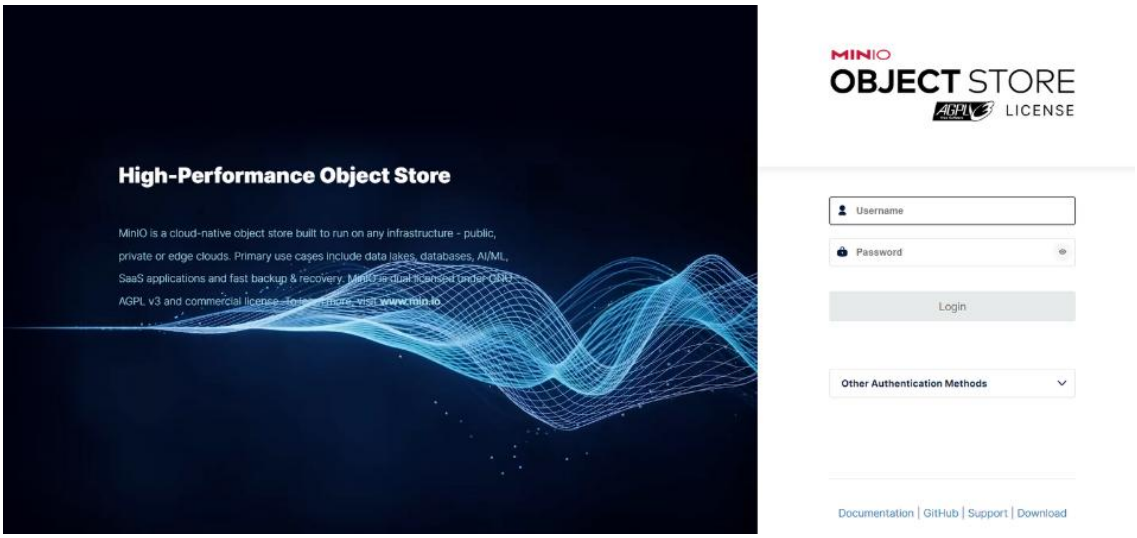

图 1 MinIO Server 登录界面

查看命令脚本，发现用户名和用户密码分别设置为 hust 和 hust_obs，输入即可登录进入使用界面。

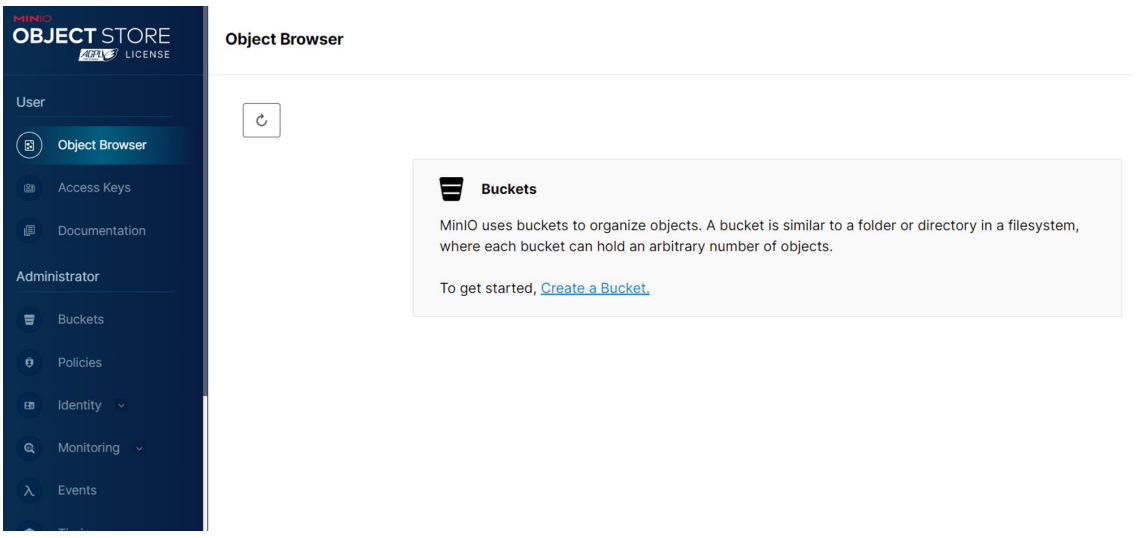

图 2 MinIO Server 使用界面

可以点击 Create a Bucket 按钮，创建一个新的 bucket，名命为 test，注意不能有大写。

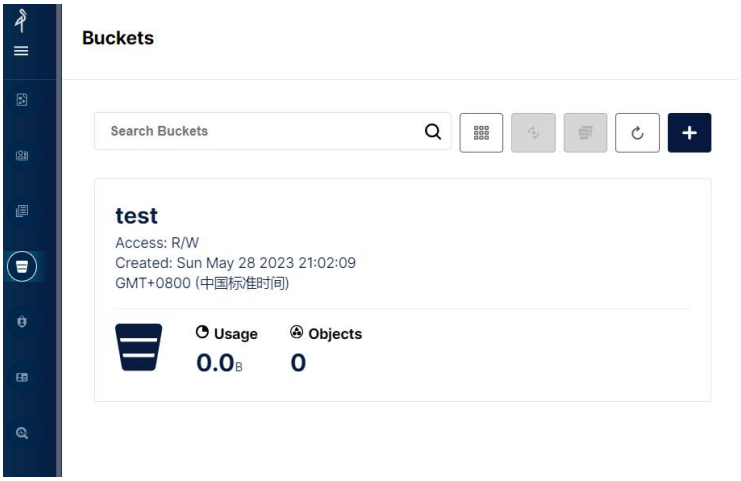

图 3 创建新 bucket “test”

（2）MinIO Client：

接下来，安装 win 版本的 MinIO Client，然后绑定至 Server。其操作为 mc config host add [hostname] [url] [username] [userpassword] --api s3v4。

```
D:\BigData\exp>mc config host add root http://127.0.0.1:9000 hust hust_obs -api s3v4
Added `root` successfully.
```

图 4 绑定成功

绑定成功之后便可以利用客户端，对服务端进行增删操作。使用 mc mb [hostname]/[bucketname]，新建一个 bucket。

```
D:\BigData\exp>mc mb root/test2
Bucket created successfully `root/test2`.
```

图 5 成功新建 bucket

之后再看服务端上，便会多出一个名为 test2 的 bucket。

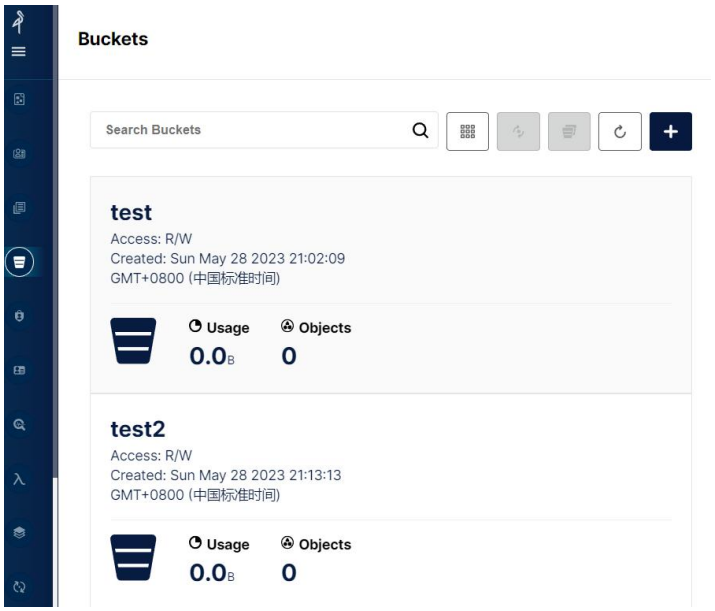

图 6 新建 bucket 后的 Server 界面

使用 mc rb [hostname]/[bucketname]，删除一个 bucket。

```
D:\BigData\exp>mc rb root/test2
Removed `root/test2` successfully.
```

图 7 成功删除 bucket

再看服务端上，之前创建的名为 test2 的 bucket 已经被删除。

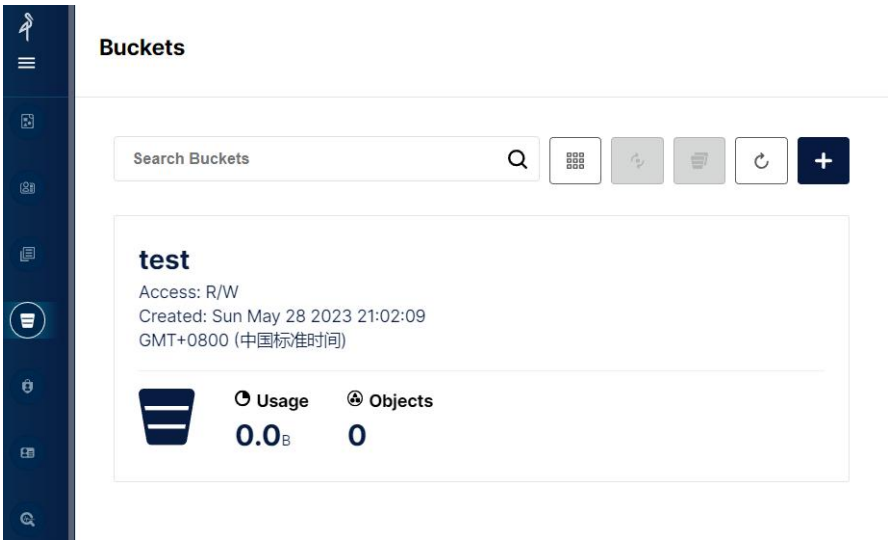

图 8 删除 bucket 后的 Server 界面

需要注意的是，在绑定服务端时使用的 url 应是 S3-API 对应的 url，而非 console 对应的 url，否则会导致错误，无法进行操作。如之前我误操作之后，使用 mc config host ls，查询 mc 绑定的服务信息。

```
root
  URL       : http://localhost:9090
  AccessKey : hust
  SecretKey : hust_obs
  API       : s3v4
  Path      : auto

s3
  URL       : https://s3.amazonaws.com
  AccessKey : YOUR-ACCESS-KEY-HERE
  SecretKey : YOUR-SECRET-KEY-HERE
  API       : S3v4
  Path      : dns


D:\BigData\exp>mc config host remove root
Removed `root` successfully.
```

图 9 错误绑定之后的服务信息

然后使用 mc config host remove [hostname]，移除错误绑定的内容。重新绑定即可正常通过客户端对服务端进行操作。

（3）S3 Bench：

最后下载 s3bench.exe 和命令脚本 run-s3bench.cmd，查看命令脚本中的信息是

否与 MinIO 中的相同，修改用户名、密码以及 bucket 名。

```
@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 b
@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=hust ^
   -accessSecret=hust_obs ^
   -bucket=test ^
   -endpoint=http://127.0.0.1:9000 ^
   -numClients=8 ^
   -numSamples=256 ^
   -objectNamePrefix=loadgen ^
   -objectSize=1024
pause
```

图 10 编辑 run-s3bench.cmd

编辑之后运行脚本文件，查看输出内容。

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


Results Summary for Write Operation(s)
Total Transferred: 0.250 MB
Total Throughput:  1.22 MB/s
Total Duration:    0.205 s
Number of Errors:  0
------------------------------------
Write times Max:       0.019 s
Write times 99th %ile: 0.018 s
Write times 90th %ile: 0.008 s
Write times 75th %ile: 0.007 s
Write times 50th %ile: 0.006 s
Write times 25th %ile: 0.005 s
Write times Min:       0.004 s


Results Summary for Read Operation(s)
Total Transferred: 0.250 MB
Total Throughput:  5.03 MB/s
Total Duration:    0.050 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: 0.002 s
Read times 50th %ile: 0.002 s
Read times 25th %ile: 0.001 s
Read times Min:       0.001 s


Cleaning up 256 objects...
Deleting a batch of 256 objects in range {0, 255}... Succeeded
Successfully deleted 256/256 objects in 353.8455ms
```

图 11 测试输出内容

5.2 对象存储性能分析

（1）测试客户端数量对传输的影响：

控制对象数量为 256，对象大小为 1024KB，改变客户端数量，进行测试，结果如表 2 所示。

表 2 不同 numClients 的测试结果

numClients (个)	numSamples (个)	objectSize (MB)	Write		Read	
			Throughput (MB/s)	Duration (s)	Throughput (MB/s)	Duration (s)
4	256	0.0010	1.01	0.248	4.06	0.062
8	256	0.0010	1.22	0.205	5.03	0.050
16	256	0.0010	1.55	0.161	5.18	0.048
32	256	0.0010	1.46	0.171	4.65	0.054
64	256	0.0010	1.45	0.173	4.28	0.058

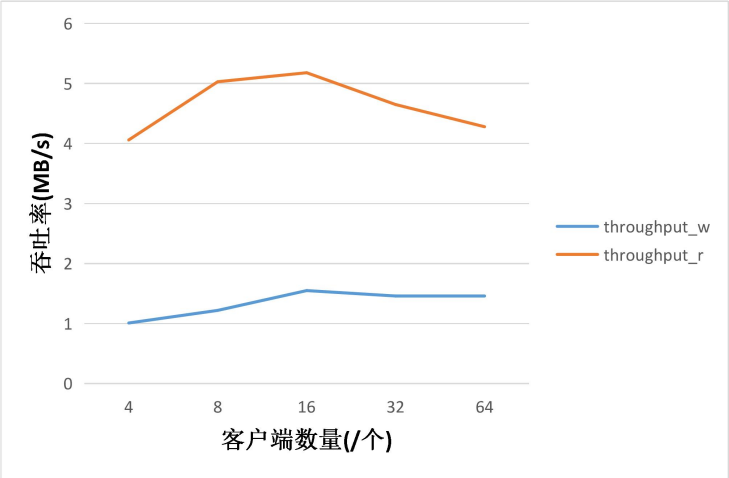

图 12 读写吞吐量随客户端数量的变化曲线

可见，随着客户端数量的增加，在一定范围内，读写吞吐量逐步增加。用户越多，发出的请求也越多，应用程序处理的请求数越多，吞吐量也就越大。

但当客户端数量大于 16 之后，读写吞吐量开始缓慢下降。估计是请求用户超负载，应用程序处理不过来，导致性能下降。

而总体来说，读的吞吐量比写吞吐量高出 1~2 倍，速度更加快。

（2）测试对象数量对传输的影响：

控制客户端数量为 8，对象大小为 1024KB，改变对象数量，进行测试，结果如表 3 所示。

表 3 不同 numSamples 的测试结果

numClients (个)	numSamples (个)	objectSize (MB)	Write		Read	
			Throughput (MB/s)	Duration (s)	Throughput (MB/s)	Duration (s)
8	64	0.0010	1.07	0.058	4.66	0.013
8	128	0.0010	1.08	0.116	4.48	0.028
8	256	0.0010	1.22	0.205	5.03	0.050
8	512	0.0010	1.51	0.332	5.06	0.099

8	1024	0.0010	1.48	0.676	4.95	0.202

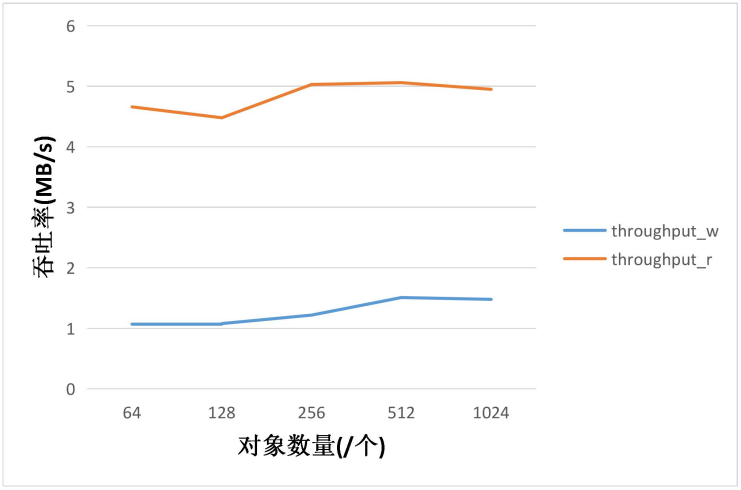

图 13 读写吞吐量随对象数量的变化曲线

可以看到，对象个数增大时，吞吐率有所变化，但无明显变化趋势。猜测对象数量对吞吐率的影响较小。

（3）测试对象大小对传输的影响：

控制客户端数量为 8，对象数量为 256，改变对象大小，进行测试，结果如表 3 所示。

表 4 不同 objectSize 的测试结果

numClients (个)	numSamples (个)	objectSize (MB)	Write		Read	
			Throughput (MB/s)	Duration (s)	Throughput (MB/s)	Duration (s)
8	256	0.0002	0.32	0.196	0.80	0.078
8	256	0.0005	0.68	0.184	2.54	0.049
8	256	0.0010	1.22	0.205	5.03	0.050
8	256	0.0020	2.43	0.206	9.86	0.051
8	256	0.0040	5.43	0.184	20.20	0.050

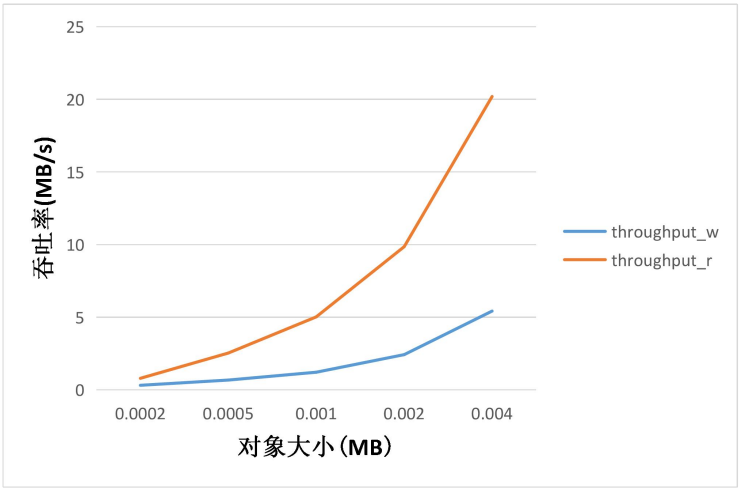

图 14 读写吞吐量随对象大小的变化曲线

由图表可知，随着对象大小的增长，吞吐率会随之增长。可以推测，对象的大小将影响吞吐率。

六、实验总结

本次实验，我使用了 MinIO 作为服务端以及客户端，构建了一个基础的对象存储系统，并使用 S3 Bench 去进行性能测试，根据测试结果，分析了各种因素对于对象存储系统的性能影响。

总的来说，实验较为简单，毕竟我没有选用较难的服务端以及客户端，老师提供的运行脚本也很方便地就能够使用。但是虽然简单，但也能够通过实验很好地加深对对象存储系统的理解。并且，由于之前并没有了解过类似知识，容易上手也能让我更放心地进行探究实验。

可惜由于各种原因，留给实验的时间较少，不能再进行更多的尝试，若是以后有机会，会进一步探索。最后感谢老师的实验编排，思路清晰、指引明确，经过实验确实能过学到许多。

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.