2020 级

《物联网数据存储与管理》课程

实 验 报 告

姓　名　朱一平

学　号　U202012475

班　号　物联网 2001 班

日　期　2023.05.29

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，分析性能问题，架设应用实践。

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。Minio 是一个基于 Go 语言的对象存储服务。它实现了大部分亚马逊 S3 云存储服务接口，可以看做是是 S3 的开源版本，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，而一个对象文件可以是任意大小，从几 kb 到最大 5T 不等。区别于分布式存储系统，minio 的特色在于简单、轻量级，对开发者友好，认为存储应该是一个开发问题而不是一个运维问题。Minio Client:mc 提供包括 ls、cat、cp、mirror、diff 等 UNIX 命令。它提供文件系统以及亚马逊 S3 兼容性云存储服务。S3bench 提供了针对兼容 S3 的端点运行非常基本的吞吐量基准测试的功能。它执行一系列的 put 操作，然后执行一系列的 get 操作，并显示相应的统计信息。 该工具使用 AWS Go SDK。

三、实验环境

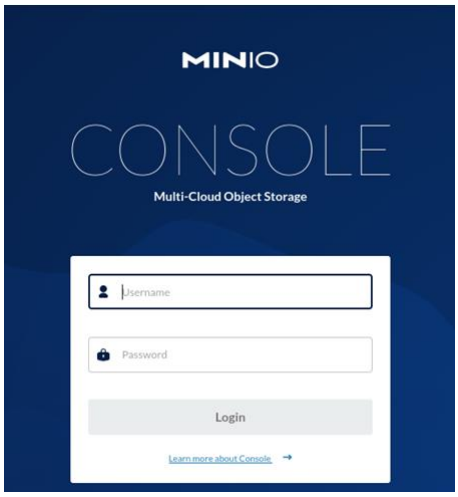
操作系统	Windows10
内存	1T
Python	3
CPU	Intel® Core™ i7-9750H CPU @ 2.60GHz

四、实验内容与实验过程

实验一：系统搭建 实验二：性能观测 实验三：尾延迟挑战

4.1 对象存储技术实践

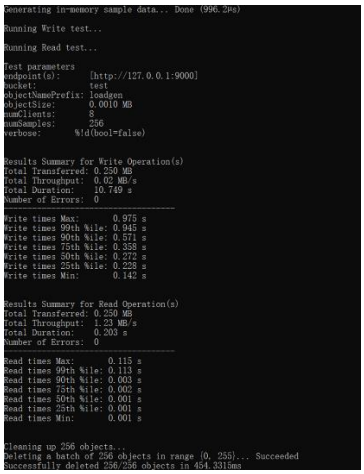
1.配置 Minio 服务端。向配置好的客户端发送文件。
2.配置客户端。在远端利用客户端新建 bucket。
3.下载老师提供的 s3bench 脚本，修改相关参数进行相应的性能测试。
1）下载 Minio。在 Minio 官网 http://www.minio.org.cn/ 下载服务端和客户端。
2）运行 Minio。打开终端，运行 minio，会看到相关入口。
3）在浏览器访问服务器。在浏览器中输入 http://127.0.0.1:9000 可以访问服务器，用户名和密码在上图中已默认给出，登录界面如图 4-2。确认登录后，可以看到界面



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，这里创建一个 test 仓库。
5）运行 MC 客户端进行对服务器的访问。重新打开一个终端，在命令行输入运行服务器之后给的提示，使得可以通过 MC 访问服务器
6）下载 s3bench 源码和相关项目并编 https://github.com/igneoussystems/s3bench，预编译的文件 https://share.weiyun.com/BICMfA4G：
7）使用测试工具 s3-bench 进行测试，先修改脚本，再运行

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



4.2 对象存储性能分析

1.对 s3bench 脚本输出的数据进行相应的处理。
2.改变不同参数观察不同的性能指标。
3.对相应的实验数据做出记录。
首先，测试标准的 s3-bench 结果。可以看到我们改变 object size 等参数时运行时间的变化。

Object size	Num clients	Num samples	写速度	写时间	读速度	读时间
0.001MB	8	256	0.02 MB/s	10.729 s	1.23 MB/s	0.203 s
1MB	8	256	9.05 MB/s s	28.282 s	1816.16 MB/s	0.141 s
10MB	8	256	41.47 MB/s	61.732 s	2237.63 MB/s	1.144 s
20MB	8	256	42.94 MB/s	119.229 s	2269.26 MB/s	2.256 s
10MB	16	256	33.10 MB/s	77.335 s	967.95 MB/s	1.301 s
10MB	24	256	29.39 MB/s	87.110 s	1482.03 MB/s	1.727 s
10MB	8	512	32.28 MB/s	158.594 s	1287.94 MB/s	3.975 s
10MB	8	128	29.01 MB/s	44.116 s	1486.86 MB/s	0.861 s

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。
4）随着 object size 增大，读写速度也会逐渐增大
5）在 object size 相等的情况下，理论上若 client 数量更多则读写速率先升高后降低，即存在一个最优水平，但未观测到
6）在 object size 相等的情况下，理论上若 sample 数量更多则读写速率先降低后升高，即存在一个最低水平，但未观测到

五、实验总结

这次试验让我熟悉了 minio 和 mc 的基本使用方法，也让我懂得了面向对象存储的一些知识。本门课程是一门基础性的课程，本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的基本概念及其优缺点，明白了面向对象存储系统的应用场景。实验中，使用了 minio 作为服务端，使用测试工具 s3-bench 进行了测试，考虑到执行时间，进行操作的容量都比较小，在实际应用中肯定会有大的多的数据量，但是对于基本的因素的观察已经足矣。实验开始时，不知道如何动手完成，后在老师同学指导帮助下完成该实验，总体而言受益匪浅。

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.
（可以根据实际需要更新调整）