



2021 级

《数据中心技术》课程 实 验 报 告

姓 名 王 源 博

学 号 M202173707

班 号 计算机技术 2106

指导老师 施 展 童 薇

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和处理离散单元的方法的通用术语。对象在一个层结构中不会再有层级结构，是以扩展元数据为特征的。

对象存储，也叫做基于对象的存储，是用来描述解决和处理离散单元的方法的通用术语，这些离散单元被称作为对象。

就像文件一样，对象包含数据，但是和文件不同的是，对象在一个层结构中不会再有层级结构。每个对象都在一个被称作存储池的扁平地址空间的同一级别里，一个对象不会属于另一个对象的下一级。

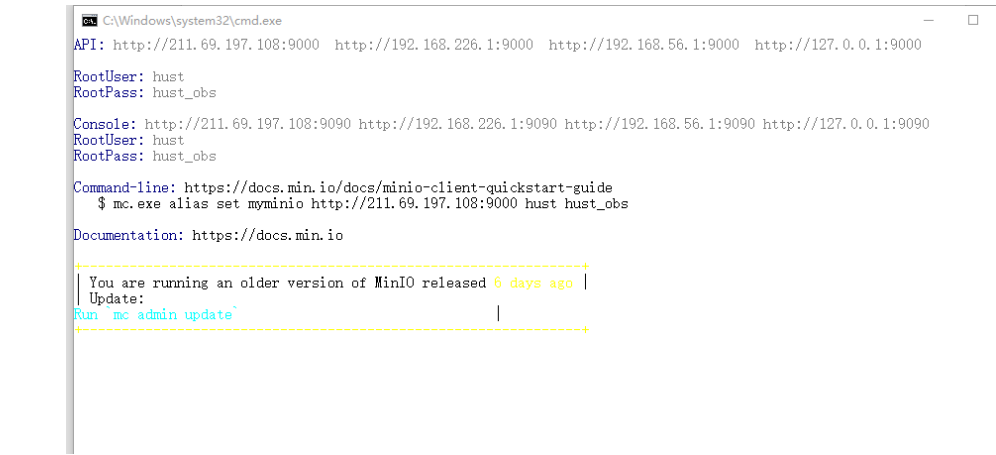
文件和对象都有与它们所包含的数据相关的元数据，但是对象是以扩展元数据为特征的。每个对象都被分配一个唯一的标识符，允许一个服务器或者最终用户来检索对象，而不必知道数据的物理地址。这种方法对于在云计算环境中自动化和简化数据存储有帮助

三、实验环境

| | |
|-----------|---|
| 操作系统 | Windows 10 64 位 |
| 处理器 | Intel(R) Core(TM) i5-10400 CPU @ 2.90GHz 2.90 GHz |
| 内存 | 16G |
| Python 版本 | 3.6.7 |
| Server | Minio |
| Client | Minio Client |
| Test | s3bench |

四、实验过程

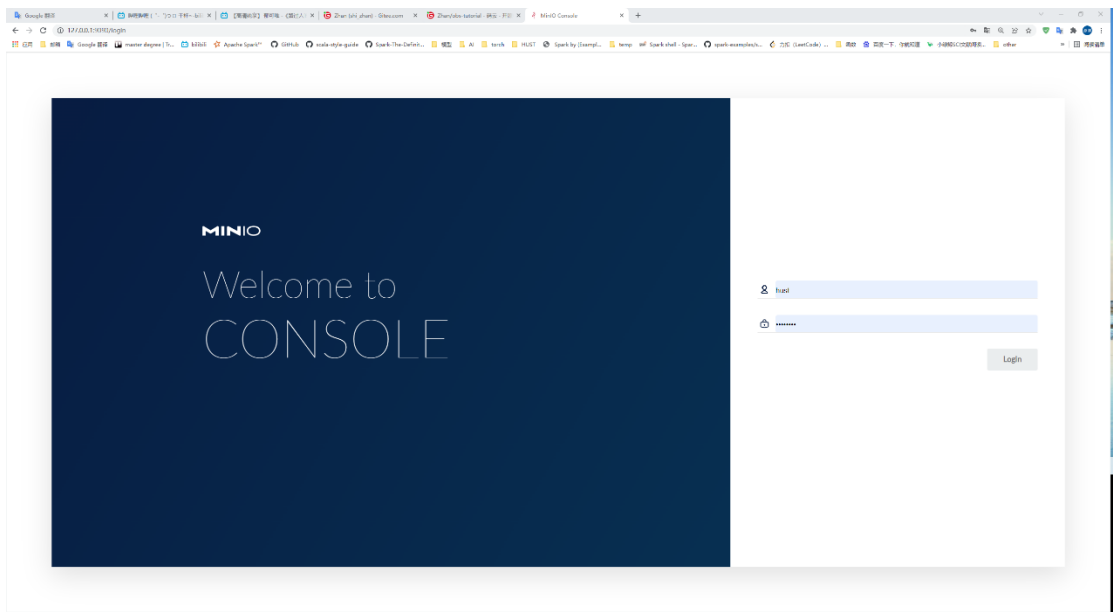
第一步首先搭建实验系统，安装 Python 环境以及 Minio 环境作为服务端。过程截图如下：



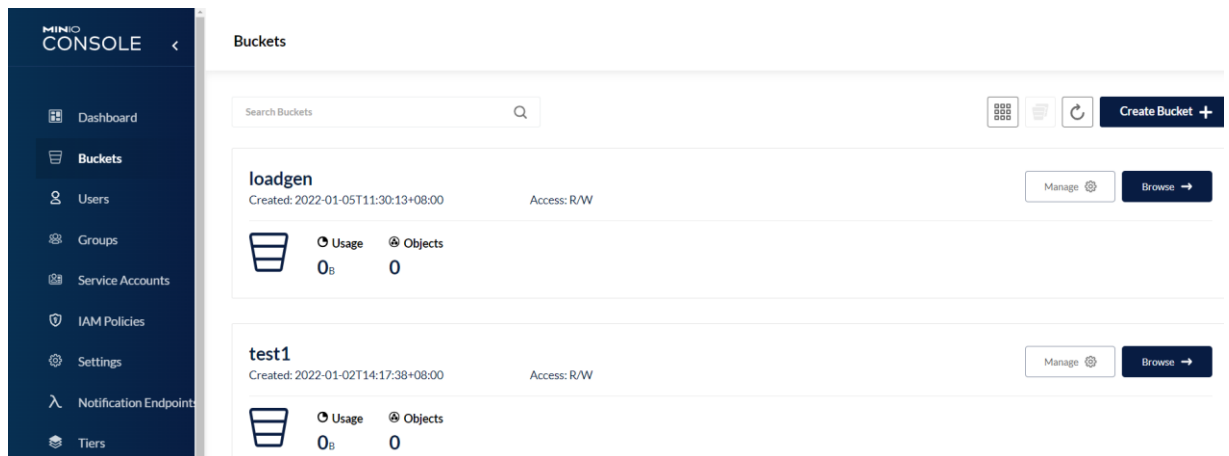
```
C:\Windows\system32\cmd.exe
API: http://211.69.197.108:9000 http://192.168.226.1:9000 http://192.168.56.1:9000 http://127.0.0.1:9000
RootUser: hust
RootPass: hust_obs
Console: http://211.69.197.108:9090 http://192.168.226.1:9090 http://192.168.56.1:9090 http://127.0.0.1:9090
RootUser: hust
RootPass: hust_obs
Command-line: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc.exe alias set myminio http://211.69.197.108:9000 hust hust_obs
Documentation: https://docs.min.io

You are running an older version of MinIO released 6 days ago |
Update:
Run mc admin update |
```

图一 启动 Minio 服务

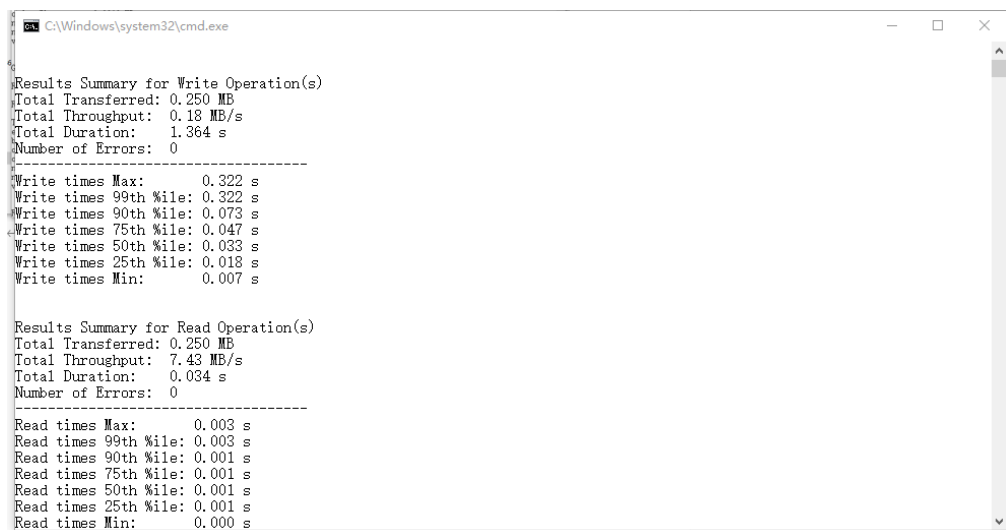


图二 minio 登录界面



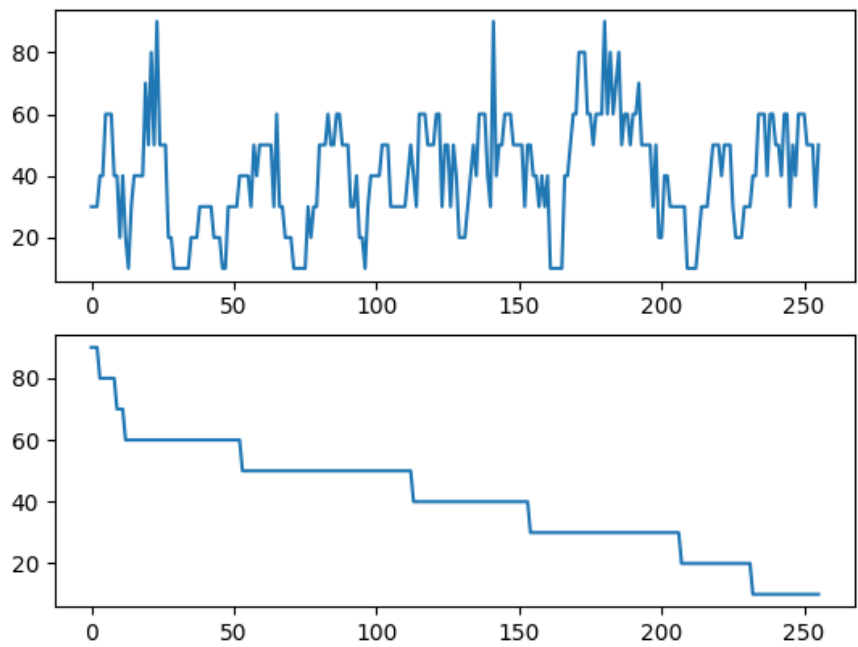
图三 minio 主界面

第二步，观察性能指标。设置参数模拟 8 个客户端同时运行，共有 256 个对象，每个大小为 1024B。直接运行 `run-s3bench.cmd` 脚本，观察性能指标如下



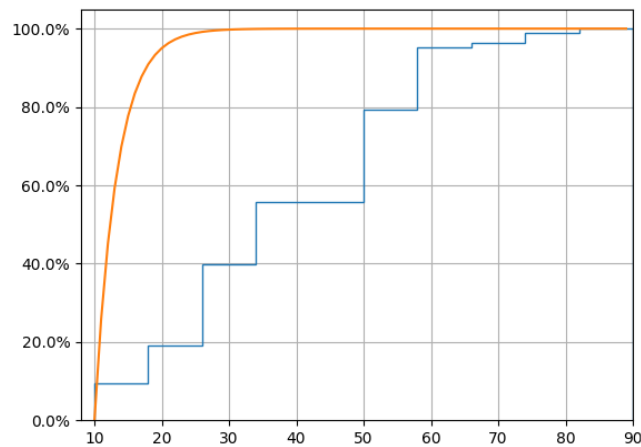
图四 读写性能指标

可以观察到，对于写操作，总共写入 0.25MB，总耗时是 1.364s，吞吐量为 0.18MB/s。对于读操作，总共读取 0.25MB，总耗时是 0.34s，总的吞吐量是 7.43MB/s。在脚本里面添加 verbose 参数可以输出每一项执行的时间，但是输出的时间是以 s 为单位，会导致大量项目的耗时一样，所以绘图的如下：



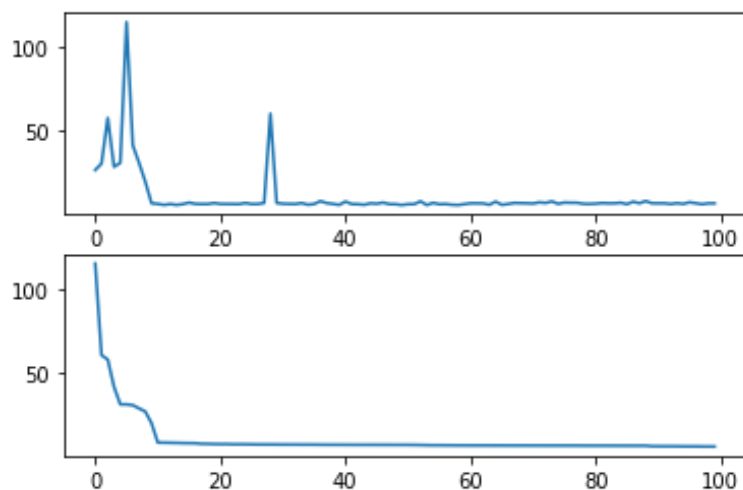
图五 写请求的延迟分布情况

观察这个延迟分布图可以看到大部写请求的延时在 40ms 左右，只有少部分超过 70ms。

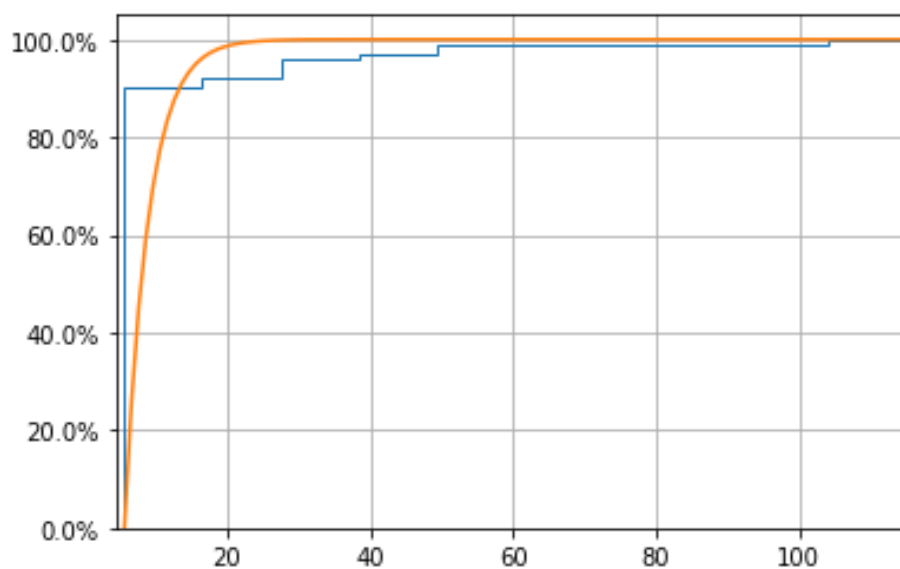


图六 排队论拟合结果

然后我用 python 的 mock-s3 模仿的 Amazon S3 跑了一遍实验，得到延迟分布图像比上一个更加丝滑，用也是 minio 的服务端。上传的是 1024*4 大小的文件，上传 100 次。得到的图像如下：



图八 mock-s3 写请求的延迟分布

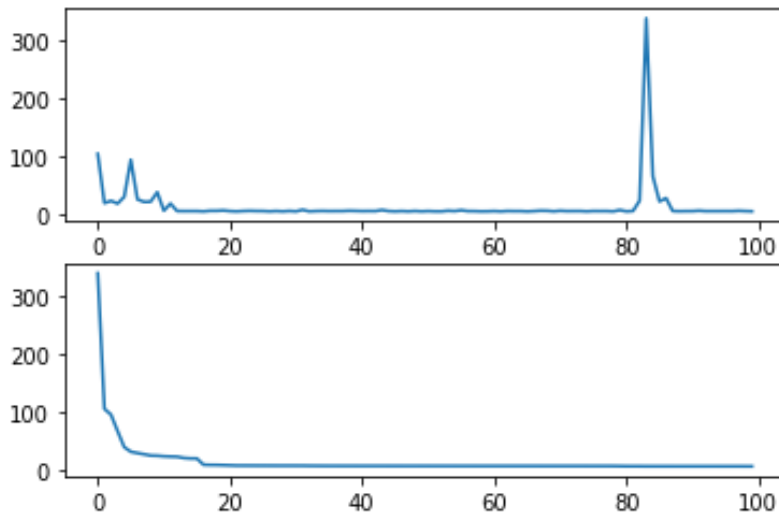


图九 排队论拟合

在这次实验中明显可以看出有不十个左右最耗时的请求显然高于平均水平。如果降低这些请求的耗时，将会使整个系统的性能提升。

然后进行挑战尾延迟的实验。对冲请求是指在指定时间间隔没有返回时，会发起对冲请求，继续等待，如果依然没有返回，则重复发送直到接收结果或者超时取消。

通过分析上面的数据可以看到，大部分写请求的延迟都低于 30ms，如果一个请求 30ms 之后还没有回应，那么这个延迟有可能远大于 50ms，就应该重发这个请求，直到回应。



图十 对冲请求执行效果

并没有观察到很明显的优化效果。

五、实验总结

通过对象存储实验，熟悉了对对象存储的方法，学会了搭建 minio 服务器，我了解了尾延迟的概念，以及自己尝试了如何去挑战优化尾延迟问题。通过自己动手实验和观察，对尾延迟有了更深的理解。

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.

（可以根据实际需要更新调整）