数据中心技术——对象存储实验报告

姓名：韩耀东
学号：202173490

实验一 系统搭建

一、实验目标

完成基础环境搭建，完成对象存储实验平台搭建（Minio）。

二、实验过程

系统环境：Ubuntu20.02，2 核 CPU，内存 2G，硬盘 20G
基础环境搭建包括：Git、Python、Go 等
服务端环境搭建包括：Minio

三、实验结果

1. 服务端运行 run-minio.sh 脚本，以用户名为 hust、密码为 hust_obs 启动 minio，并指定 API 端口为 9000，控制台端口为 9090。

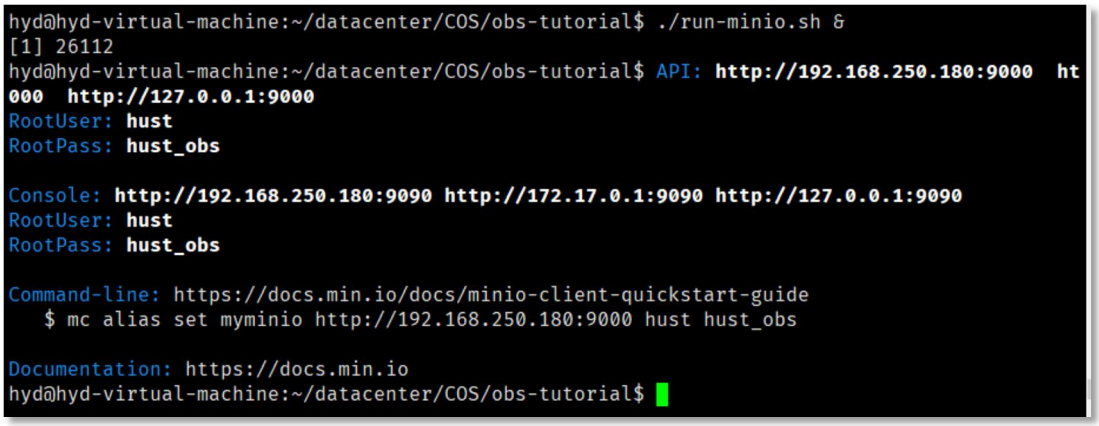

图 1 服务端启动

2. 输入控制台的地址，可以看到 minio 能够正常运行。

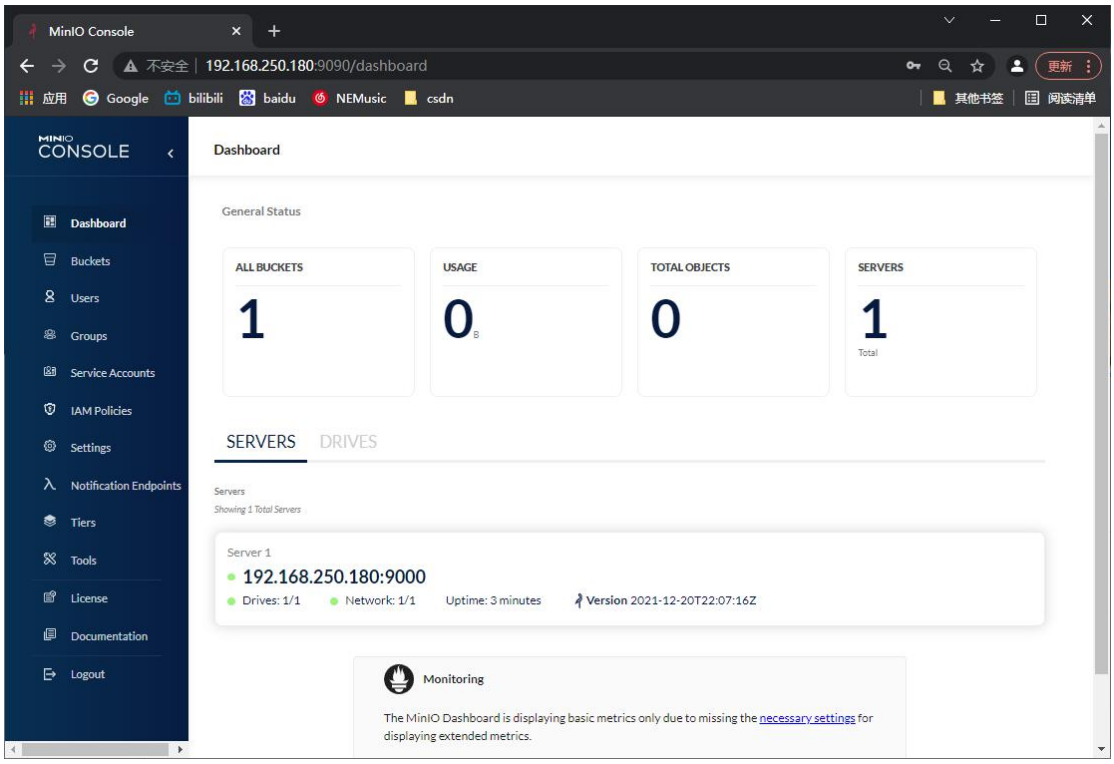

图 2 服务端正常运行

3. 创建名为 loadgen 的存储桶，并使用 S3 Bench 测试工具进行测试。执行 run-s3bench.sh 脚本，主要执行命令如下：

```
~/go/bin/s3bench \
    -accessKey=hust \
    -accessSecret=hust_obs \
    -bucket=loadgen \
    -endpoint=http://192.168.250.180:9000 \
    -numClients=8 \
    -numSamples=256 \
    -objectNamePrefix=loadgen \
    -objectSize=$(( 1024*32 ))
```

其中，指定客户端数量为 8，对象个数为 256，对象大小为 1024*32 字节

输出结果为：

```
Test parameters
endpoint(s):        [http://192.168.250.180:9000]
bucket:             loadgen
objectNamePrefix: loadgen
objectSize:         0.0312 MB
numClients:         8
numSamples:         256
verbose:        %!d(bool=false)
```

Results Summary for Write Operation(s)
Total Transferred: 8.000 MB
Total Throughput: 27.74 MB/s
Total Duration: 0.288 s
Number of Errors: 0

Write times Max: 0.029 s
Write times 99th %ile: 0.023 s
Write times 90th %ile: 0.014 s
Write times 75th %ile: 0.011 s
Write times 50th %ile: 0.008 s
Write times 25th %ile: 0.005 s
Write times Min: 0.002 s

Results Summary for Read Operation(s)
Total Transferred: 8.000 MB
Total Throughput: 56.51 MB/s
Total Duration: 0.142 s
Number of Errors: 0

Read times Max: 0.017 s
Read times 99th %ile: 0.016 s
Read times 90th %ile: 0.010 s
Read times 75th %ile: 0.006 s
Read times 50th %ile: 0.003 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 153.68334ms

实验二 性能观测

一、实验目标

安装评测工具 S3 Bench，并使用脚本，以不同的参数运行测试获得百分位延迟以及吞吐率数据，并进行分析。

二、实验过程

实验的观测指标有吞吐率 Throughtput 以及延迟 Latency，环境中支持调整的参数有对象尺寸 object size、并发数 Clients。

首先在 object size 为 1KB、8KB、64KB、1024KB、8192KB、16384KB 的情况下进行测试，获得相应的百分位延迟数据以及吞吐率。

然后对 Clients 为 1、8、64、128、256 的情况下进行测试，获得相应的百分位延迟数据以及吞吐率。

三、实验结果

1. 对象尺寸对性能的影响

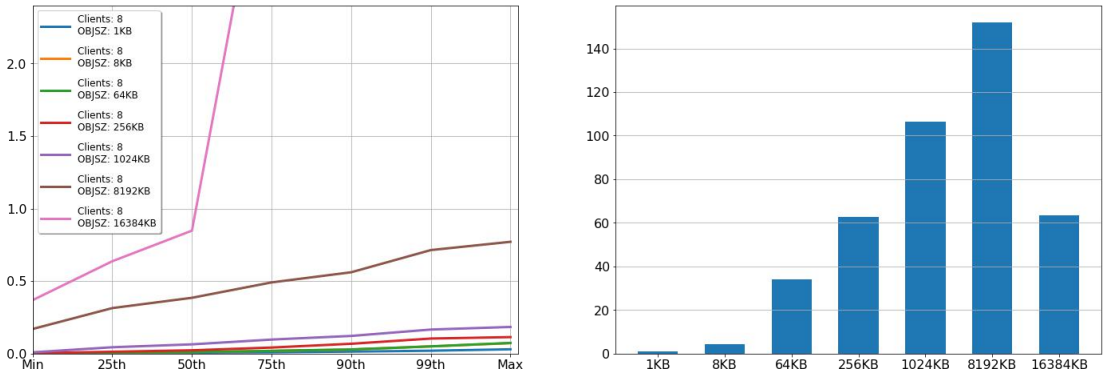

图 3 对象尺寸对写性能的影响

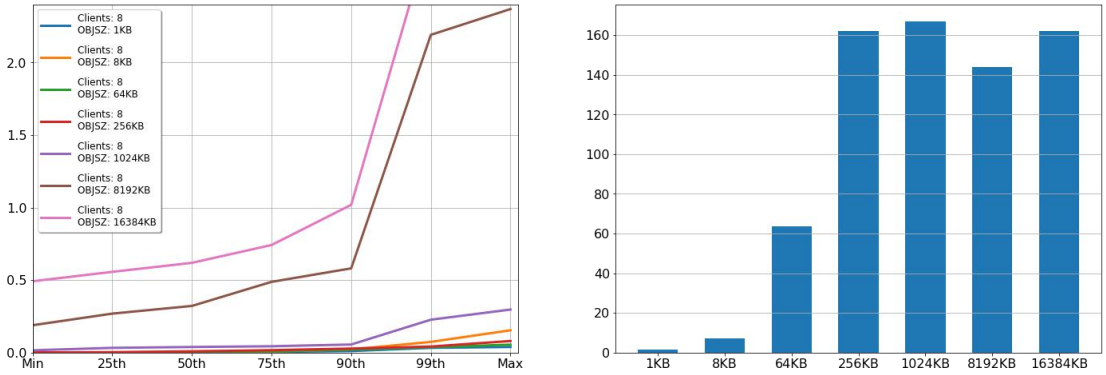

图 4 对象尺寸对读性能的影响

对象尺寸对写性能的影响如图 3 所示，左图是各种尺寸下百分位延迟数据，可以发现当对象尺寸达到 8192KB 时，整体延迟都变得很大，而吞吐率达到最大；当尺寸达到 16384KB 时，50th 延迟就已经接近 1s，整体最大延迟 7.045s，尾延迟现象严重，吞吐率反而减小。

对象尺寸对读性能的影响如图 4 所示，1024KB 及以下的延迟分布区别不大，这与写性能的变化是类似的，但是当大小达到 256KB 时，读吞吐率已经接近饱和。

所以在当前的系统环境下，对象大小在 256KB 左右应该是一个比较不错的选择。当然，如果系统更多的处理写请求，而读请求并不多，那么我们可以将这个大小上调至 1024KB，甚至更大。

2. **并发数量对性能的影响**

图 5 展示了并发数量对写性能的影响，在对象大小为 64KB 的情况下测试。结果显示当并发数在 16 以下时，延迟表示很好；并发数为 32 时，90th 延迟在 0.1s 以下，这是一个还可以接受的结果，但并发数如果仍然增加，结果将变得无法忍受，有较为严重的尾延迟现象。在吞吐率方面，并发数达到 8 时就已经接近饱和。所以在本实验环境中，需要对并发数量尽量控制在 64 以下。

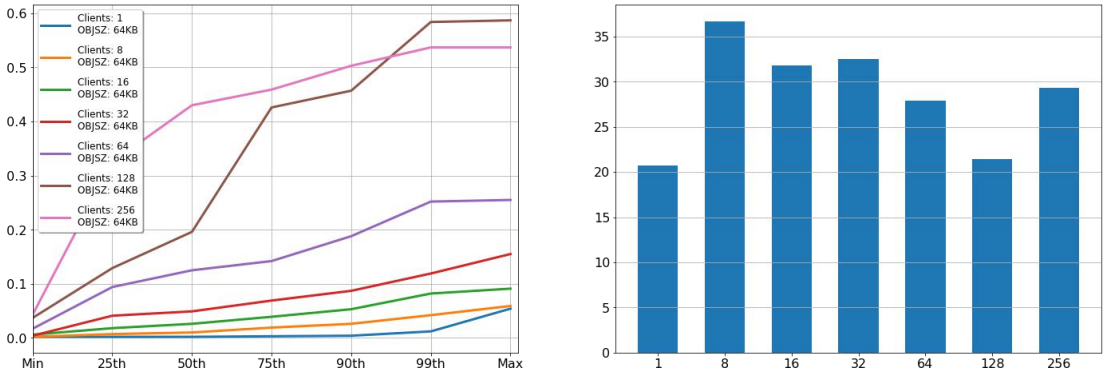

图 5 并发数量对写性能的影响

实验三 尾延迟挑战

一、实验目标

观察尾延迟现象，并使用对冲请求的方法进行优化。

二、实验过程

运行 obs-tutorial 中的 python 脚本，观察尾延迟现象。图 6 为实测数据的延迟分布，可以发现有小部分的测试数据延迟很大。图 7 是使用排队论模型拟合后的结果，拟合函数为 $F(t) = 1 - e^{(-\alpha * t)}$，其中$\alpha = 0.3$，表示单位时间平均请求数。

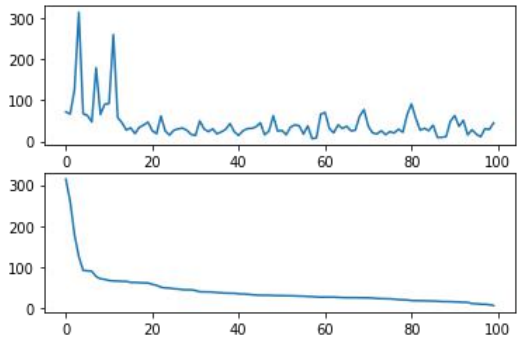

图 6 延迟分布

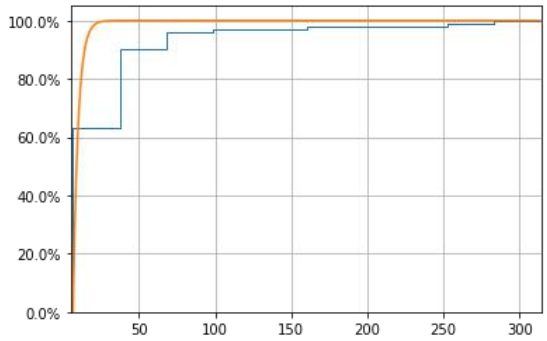

图 7 使用排队论模型进行拟合

从上图可以看出，约有 90%的请求在 70ms 内完成，所以在对冲请求中设置 70ms 作为分界，当请求时间大于 70ms，就重新发送一个相同的请求。

三、实验结果

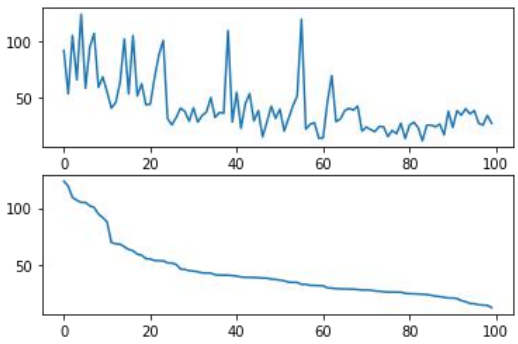

图 8 延迟分布

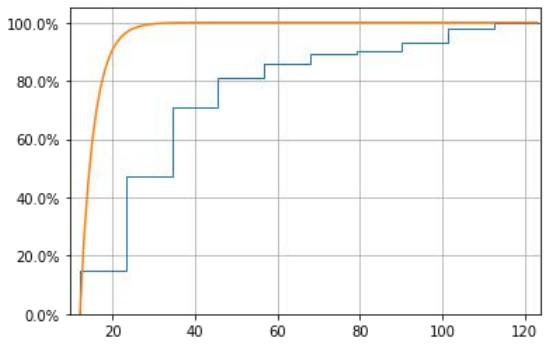

图 9 使用排队论模型进行拟合

将 70ms 设置为对冲请求分界，测试尾延迟分布如图 8 所示，尽管仍约有 90%的请求在 70ms 内完成，但是最大延迟由 300 多 ms 减小至 120ms 左右，可以得出结论，对冲请求极大地缓解了尾延迟现象。