

华中科技大学

数据中心技术课程实验报告

院 系 计算机科学与技术

班 级 计算机硕 2106

学 号 M202173708

姓 名 李其锟

2022 年 1 月 5 日

一. 运行环境搭建

1. 实验环境

处理器：AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx 2.1 GHz

内存：16GB

系统：Windows10

2. 服务端搭建

在官网按照教程下载并安装 minio, 当前路径下使用 cmd 运行 run-minio 脚本, 进行 minio 服务端搭建, 搭建成功信息如下. 用户名默认为 hust, 密码默认为 hust_obs, web-gui 端口为 9090

```
D:\git_project\data-center-course-experiments\experiments1>run-minio

+-----+
| You are running an older version of MinIO released 1 month ago |
| Update: Run mc admin update                                     |
+-----+

API: http://10.0.2.11:9000 http://192.168.140.1:9000 http://192.168.219.1:9000 http://127.0.0.1:9000
RootUser: hust
RootPass: hust_obs

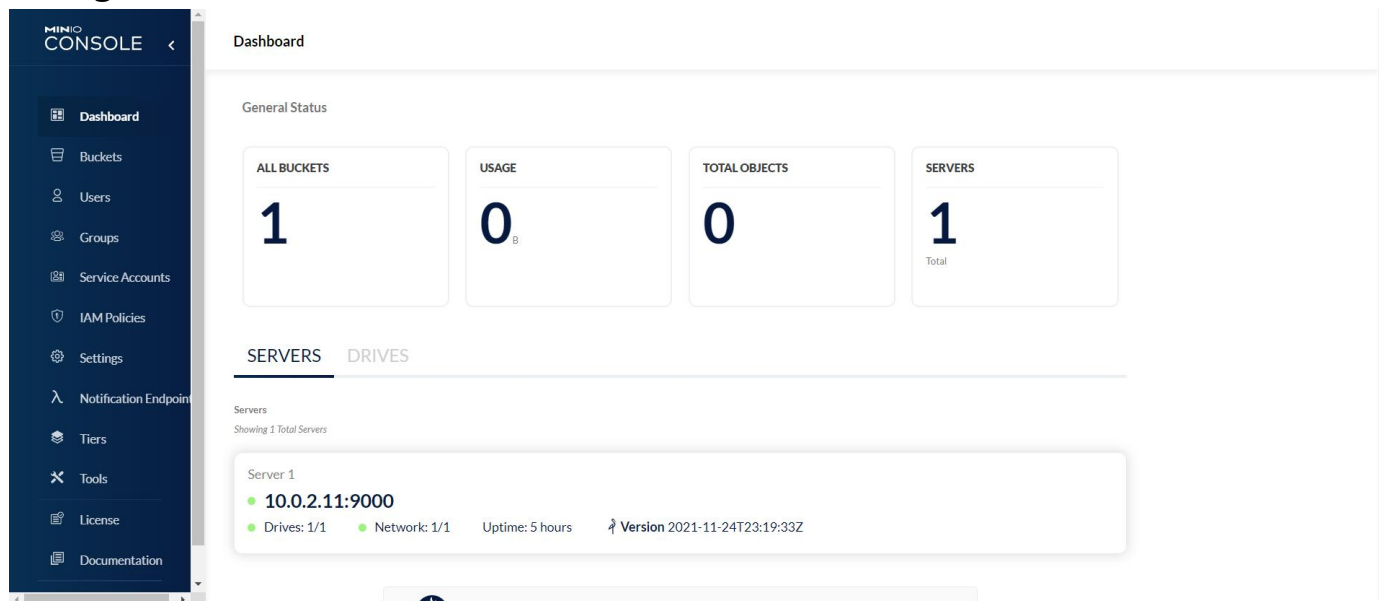
Console: http://10.0.2.11:9090 http://192.168.140.1:9090 http://192.168.219.1:9090 http://127.0.0.1:9090
RootUser: hust
RootPass: hust_obs

Command-line: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc.exe alias set myminio http://10.0.2.11:9000 hust hust_obs

Documentation: https://docs.min.io
```

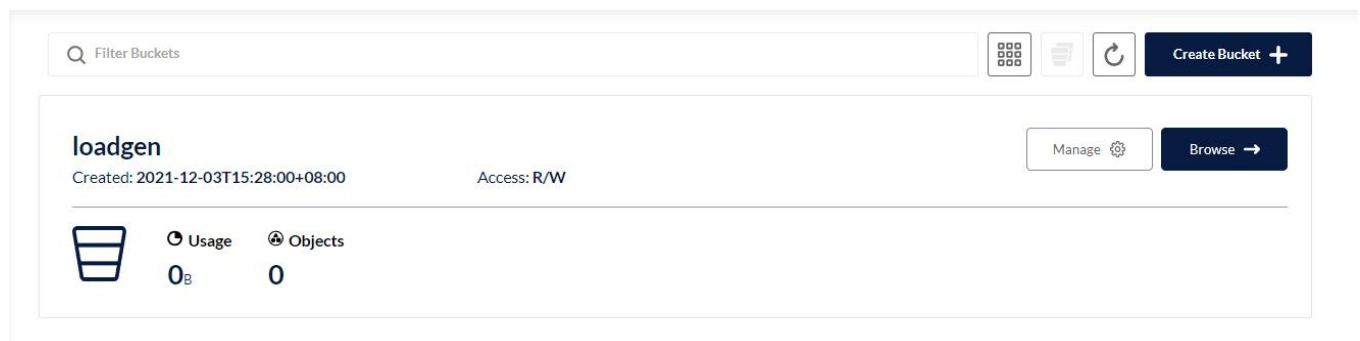
3. 实验配置

浏览器中访问 cmd 输出的网址 <http://127.0.0.1:9090>，进入 minio 服务器端的 gui 界面。



移动到 buckets 栏上创建一个名为“loadgen”的桶，用于接下来的性能测试, 此处已创建

Buckets



二. 性能测试

1. 脚本测试

下载 run-s3bench.sh 脚本, 并修改其中参数如下

```
s3bench.exe ^
-accessKey=hust ^
-accessSecret=hust_obs ^
-bucket=loadgen ^
-endpoint=http://127.0.0.1:9000 ^
-numClients=8 ^
-numSamples=256 ^
-objectNamePrefix=loadgen ^
-objectSize=4096
pause
```

在 cmd 中运行脚本进行 s3bench 基准测试, 得到以下数据

```
Test parameters
endpoint(s): [http://127.0.0.1:9000]
bucket: loadgen
objectNamePrefix: loadgen
objectSize: 0.0039 MB
numClients: 8
numSamples: 256
verbose: %!d(bool=false)

Generating in-memory sample data... Done (4.0008ms)

Running Write test...

Running Read test...

Test parameters
endpoint(s): [http://127.0.0.1:9000]
bucket: loadgen
objectNamePrefix: loadgen
objectSize: 0.0039 MB
numClients: 8
numSamples: 256
verbose: %!d(bool=false)

Results Summary for Write Operation(s)
Total Transferred: 1.000 MB
Total Throughput: 0.51 MB/s
Total Duration: 1.954 s
Number of Errors: 0
```

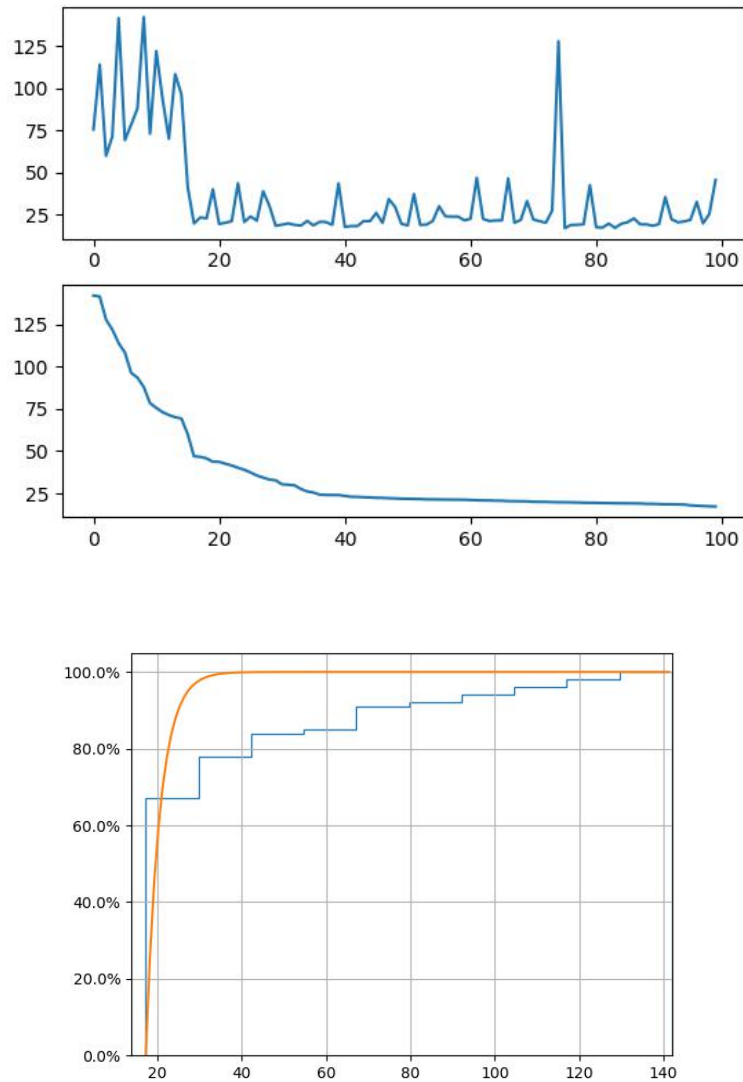
```
-----  
Write times Max:      0.135 s  
Write times 99th %ile: 0.119 s  
Write times 90th %ile: 0.102 s  
Write times 75th %ile: 0.082 s  
Write times 50th %ile: 0.057 s  
Write times 25th %ile: 0.039 s  
Write times Min:      0.013 s  
  
Results Summary for Read Operation(s)  
Total Transferred: 1.000 MB  
Total Throughput:  5.11 MB/s  
Total Duration:    0.196 s  
Number of Errors:  0  
-----  
Read times Max:      0.016 s  
Read times 99th %ile: 0.015 s  
Read times 90th %ile: 0.009 s  
Read times 75th %ile: 0.007 s  
Read times 50th %ile: 0.006 s  
Read times 25th %ile: 0.004 s  
Read times Min:      0.002 s  
  
Cleaning up 256 objects...  
Deleting a batch of 256 objects in range {0, 255}... Succeeded  
Successfully deleted 256/256 objects in 594.9748ms
```

分析图中数据可知, 最大时长往往比最小时长高出许多倍, 同时也大大超出 90%基准线的标准, 这些占比很小的数据传输很多时候会造成额外的时间成本

二. 尾延迟挑战

1. 数据采集

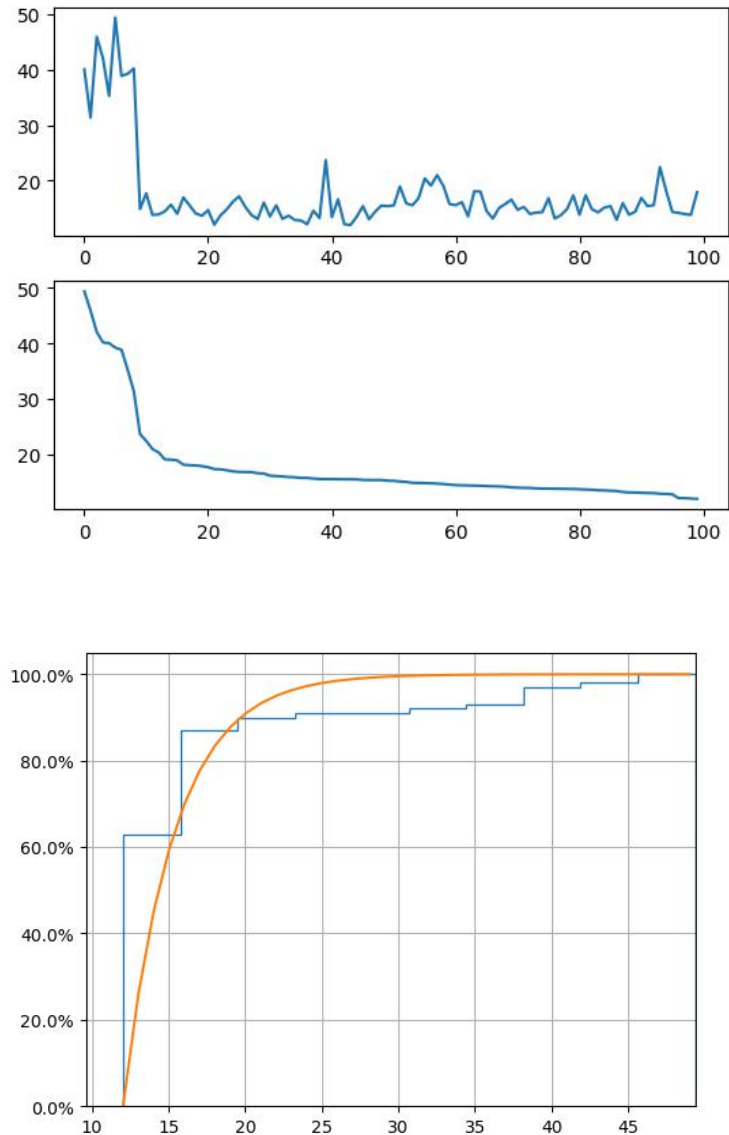
修改 latency-collect 和 latency-plot 文件中的部分参数以连接到搭建的 minio 服务器, 执行测试获得以下尾延迟分布图



由数据可知, 存在少部分的写请求耗时远远高于其他请求的平均值, 即观测到尾延迟现象.

2. 对冲优化

分析上图数据知, 90%的基准线大致为 0.025 秒, 因此将超时重发的限制制定为 0.025 秒并重新收集数据如下



分析数据可发现, 延迟分布相比之前更加均匀, 数据的方差和极值均显著降低, 可见尾延迟优化生效