对象存储实验报告

姓名：狄时禹

学号：M202173489

实验一 环境搭建

系统环境：虚拟机 Ubuntu18.04,i7-8700 CPU@3.20GHz,4GB RAM

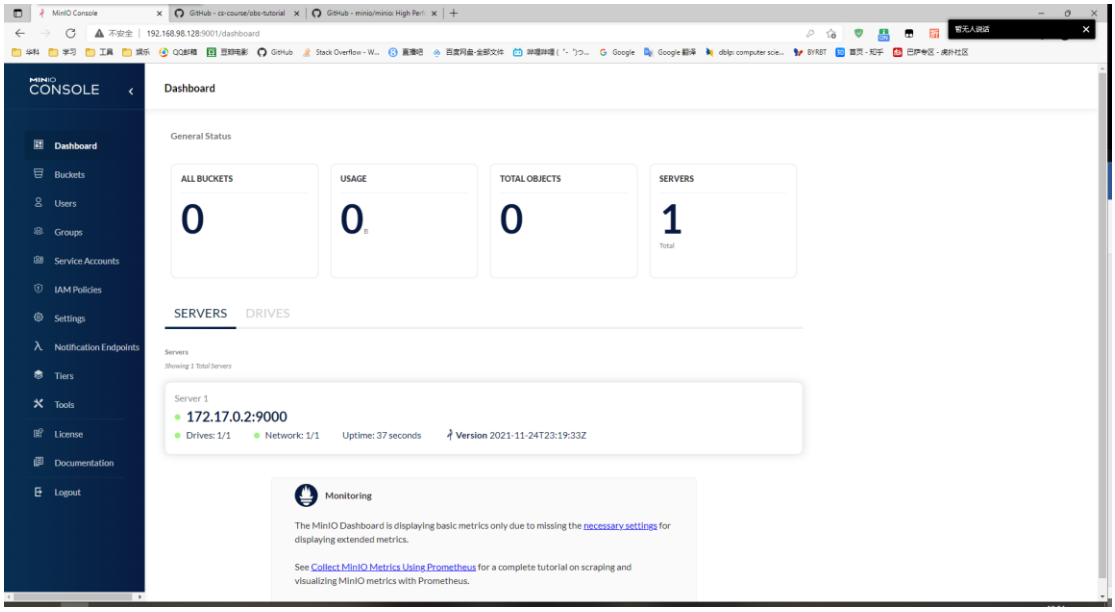
前置环境：java、docker 等

搭建过程

1、服务器端启动 minio，设置用户名和密码均为 minioadmin

```
sparrow@ubuntu: $ sudo docker run -d -v miniovolume:/home/sparrow/dsy/minio/data/  -p 9000:9000 -p 9001:9001 quay.io/minio/minio server ./data --console-address ":9001"
aa6844ed74aa59a8382bc1799a20357dab7cf361c537c6f3c317b8a22d7801f7
sparrow@ubuntu: $ sudo docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED         STATUS         PORTS                                                                                  NAMES
aa6844ed74aa   quay.io/minio/minio   "/usr/bin/docker-ent…"   22 seconds ago   Up 21 seconds   0.0.0.0:9000-9001->9000-9001/tcp, :::9000-9001->9000-9001/tcp   wonderful_hugle
```

2、客户端登录后台管理



3、安装 minio 客户端 mc，创建一个名为 mybucket 的桶，上传 test.txt，查看内容并删除

```
sparrow@ubuntu:~$ sudo docker pull minio/mc
Using default tag: latest
latest: Pulling from minio/mc
Digest: sha256:53c2063206840f2be4ba1cbcd71728cfedc334c3b0c22660fe9862d938b0cea3
Status: Image is up to date for minio/mc:latest
docker.io/minio/mc:latest
sparrow@ubuntu:~$ sudo docker run -it --entrypoint=/bin/sh minio/mc
sh-4.4# mc config host add myminio http://192.168.98.128:9000 minioadmin minioadmin
mc: Configuration written to `/root/.mc/config.json`. Please update your access credentials.
mc: Successfully created `/root/.mc/share`.
mc: Initialized share uploads `/root/.mc/share/uploads.json` file.
mc: Initialized share downloads `/root/.mc/share/downloads.json` file.
Added `myminio` successfully.
sh-4.4# mc mb myminio/mybucket
Bucket created successfully `myminio/mybucket`.
sh-4.4#
```

```
sh-4.4# mc cat myminio/mybucket/test.txt
hello myminio/mybucket/test.txtsh-4.4#
sh-4.4# mc rm myminio/mybucket/test.txt
Removing `myminio/mybucket/test.txt`.
sh-4.4#
```

实验二 性能测试

实验目标：观察在不同的对象大小、并发度下吞吐量和延迟的变化

实验准备：安装测试工具 S3 Bench

实验步骤：先测试对象大小分别为 1KB、8KB、64KB、1024KB、8192KB、16384KB 的情况下延迟及吞吐率的情况，然后测试客户端数量分别为 1、8、64、128、256 的情况下延迟及吞吐率的情况。

实验结果：

1、对象尺寸对写性能的影响

图 1 中左图为各对象大小下百分位延迟数据(单位 s)，右图为吞吐率数据(单位 MB/s)，由图可知当对象大小达到 8196KB 时整体延迟增大，当对象大小不小于 1024KB，尤其是不小于 8192KB 时，尾延迟现象严重；吞吐率随对象大小增大而增大，但增速渐缓，对象大小不小于 1024KB 时吞吐率变化不大。

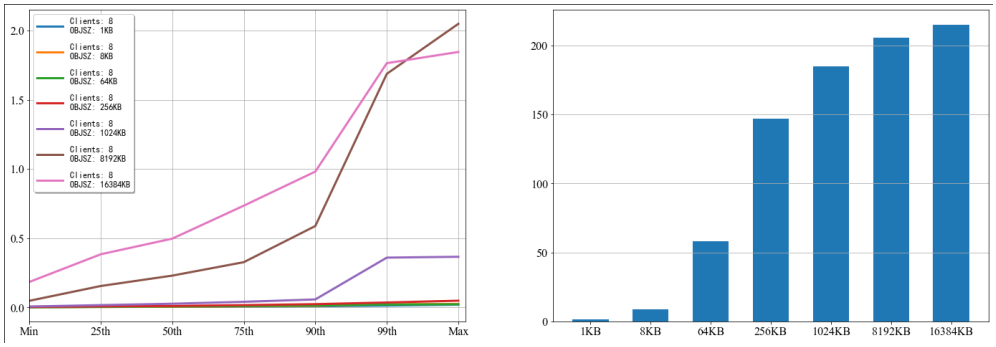


图 1 对象尺寸对写性能的影响

2、对象尺寸对读性能的影响

类似图 1，由图 2 可知当对象大小达到 8196KB 时整体延迟增大，当对象大小不小于 8192KB 时，尾延迟现象严重；吞吐率随对象大小增大而先增大后减小，对象大小为 1024KB 时吞吐率最大。

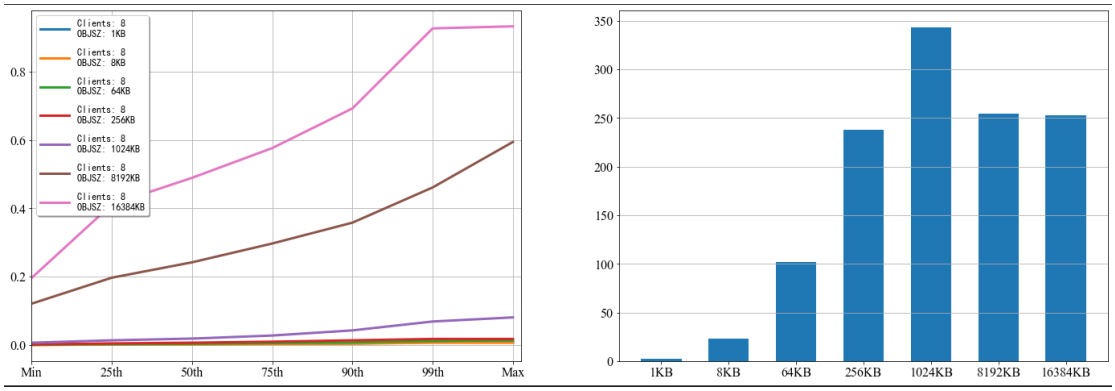


图 2 对象尺寸对读性能的影响

3、并发数量对性能的影响

图 3 显示了对象大小为 64KB 时，在测试的各并发数量下延迟和吞吐率的变化情况。由图可知，当并发数不大于 64 时整体延迟均低于 0.2s，但超过 64 则整体延迟大幅升高且尾延迟现象较严重，第 75 百分位后延迟明显高于第 75 百分位前。吞吐率方面，并发数为 16 时吞吐率最高，并发数不小于 128 时吞吐率大幅降低。

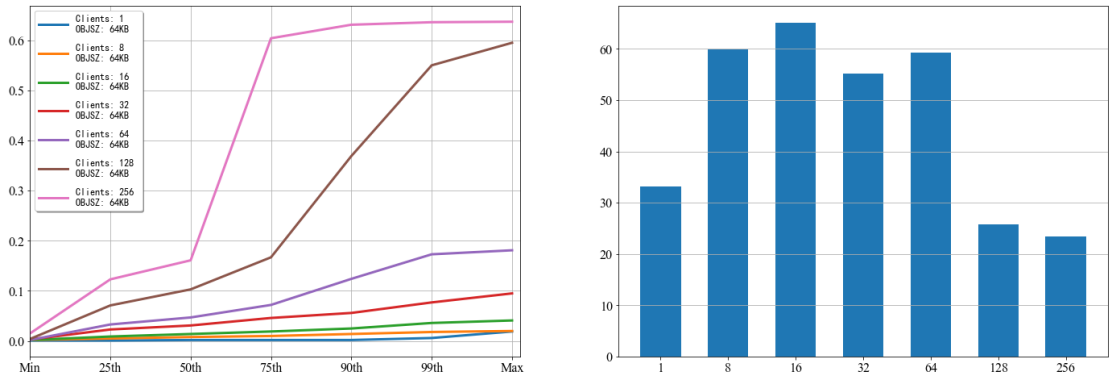


图 3 并发数量对性能的影响

实验结论：有实验结果可知，在当前系统环境下，对象大小为 1024KB 能够同时获得较好的读写性能；在对象大小为 64KB 时，并发数控制在 8 到 32 之间能够有较低的延迟和较高的吞吐率。

实验三 尾延迟挑战

实验设置：

上传文件大小	4KB
上传任务数	100
线程数	2

1、使用对冲请求前实验数据如图 6 所示，有少量请求的延迟很大，即存在着明显的尾延迟现象，其中有 90%的请求延迟小于 47ms。图 5 是使用排队论模型拟合后的结果，拟合函数为$F(t) = 1 - e^{-at}$，其中$a = 0.3$，表示单位时间评价请求数。

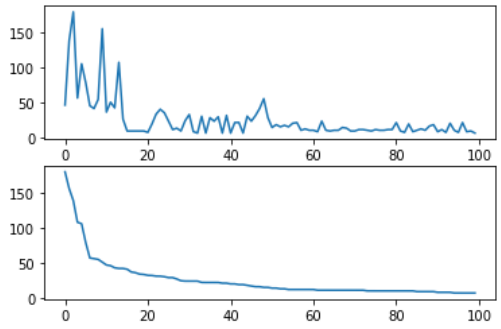


图 4 原请求延迟

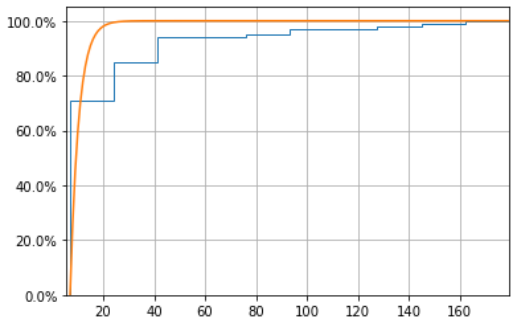


图 5 使用排队论模型拟合

2、使用对冲请求，当请求在 47ms 内没有返回时重新发送相同请求，以原请求和新请求返回时间最小值作为该请求响应时间。实验数据如图 6 所示，可以看到，最大延迟由 180ms 左右降为 65ms 左右，平均延迟下降的情况下尾延迟现象也得到了改善。

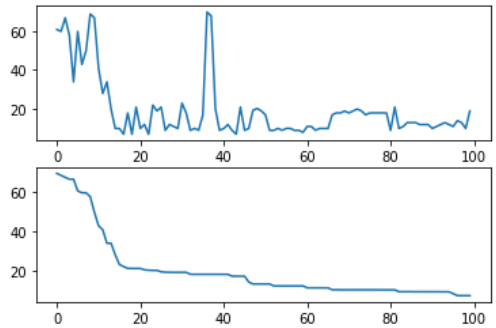


图 6 使用对冲后请求延迟

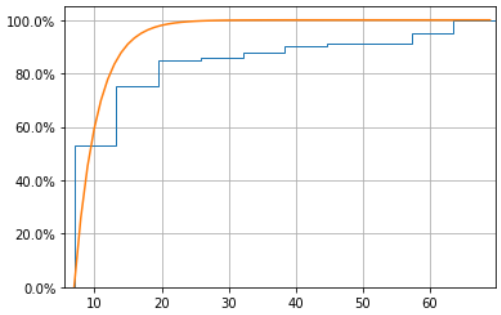


图 7 使用排队论模型拟合