華中科技大學

数据中心技术课程实验报告

院　系　计算机科学与技术学院

班　级　2106

学　号　M202173732

姓　名　张宇健

2021 年 12 月 31 日

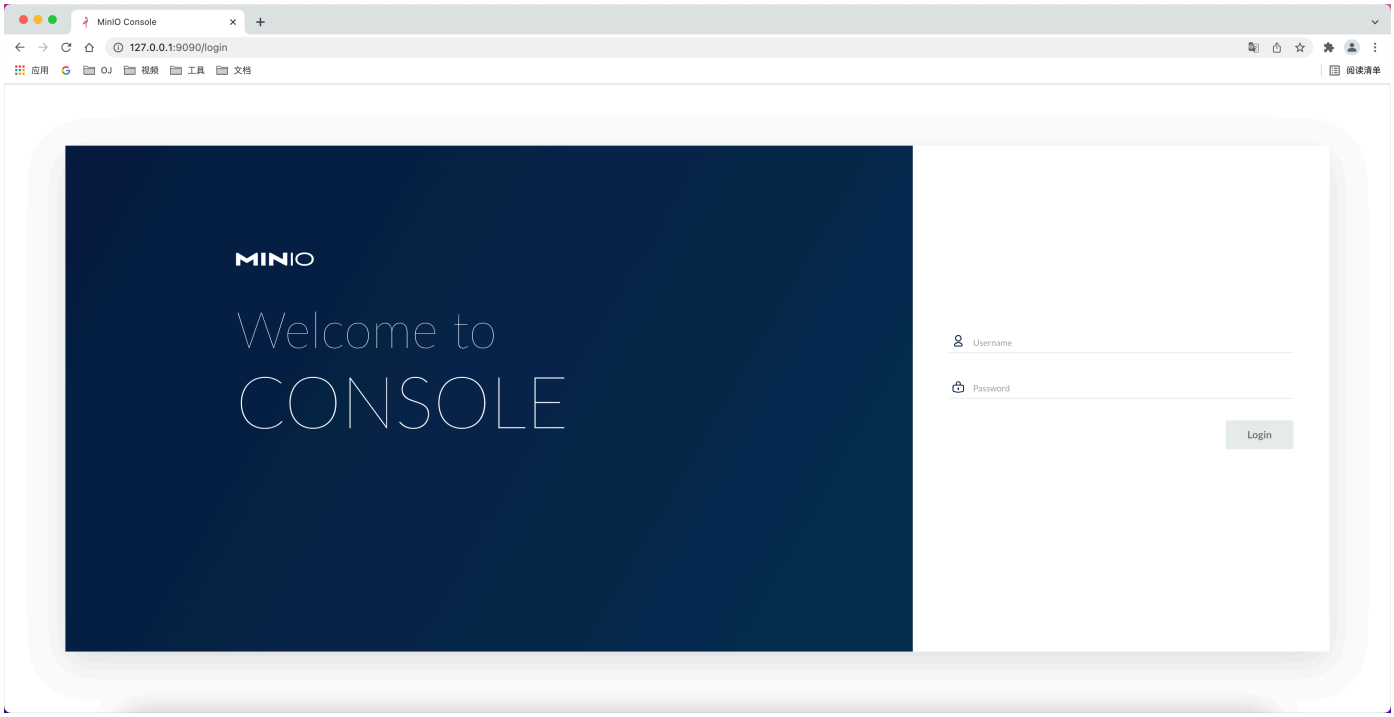
1、系统搭建

1.1 服务端

```
wget https://dl.min.io/server/minio/release/darwin-amd64/minio
chmod +x minio
```

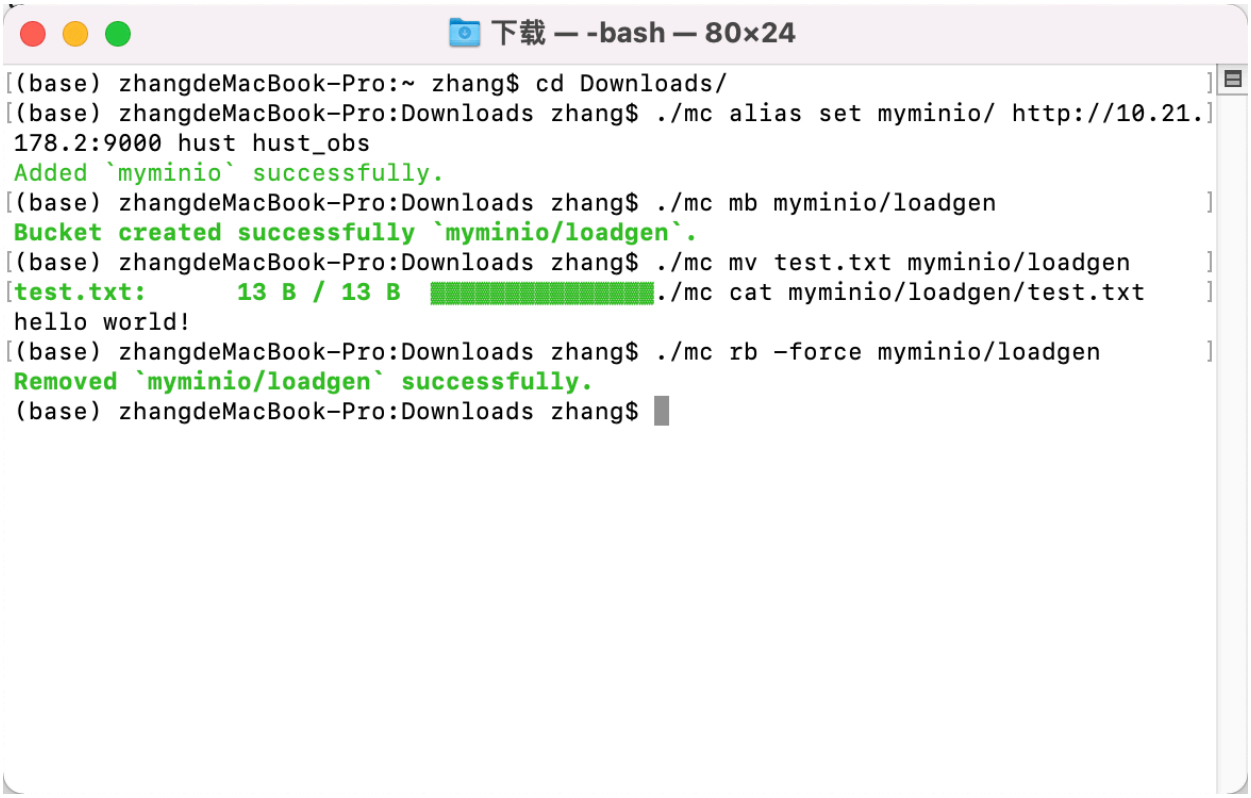
```
# run-minio.sh
#!/bin/bash
export MINIO_ROOT_USER=hust
export MINIO_ROOT_PASSWORD=hust_obs
export MINIO_PROMETHEUS_AUTH_TYPE="public"
./minio -C ./ server ./root --console-address ":9090"
```

环境运行成功之后，在浏览器中输入 http://127.0.0.1:9090，即可见下图



1.2 客户端

```
wget https://dl.min.io/client/mc/release/darwin-amd64/mc
chmod +x mc
./mc alias set myminio/ http://10.21.178.2:9000 hust hust_obs
./mc mb myminio/loadgen # 创建桶
./mc mv test.txt myminio/loadgen # 将本地文件上传到桶中
./mc cat myminio/loadgen/test.txt # 查看文件内容
./mc rb -force myminio/loadgen # 强制删除桶
```



2、性能观测

2.1 安装和运行 s3bench

安装

```
go get -u github.com/igneous-systems/s3bench
```

运行

配置客户端数为 8，样本数为 256，对象大小为 1024 * 32 字节

```
# run-s3bench.sh
#!/bin/sh
s3bench=~/go/bin/s3bench

if [ -n "$GOPATH" ]; then
    s3bench=$GOPATH/bin/s3bench
fi
$s3bench \
    -accessKey=hust \
    -accessSecret=hust_obs \
    -bucket=loadgen \
    -endpoint=http://127.0.0.1:9000 \
    -numClients=8 \
    -numSamples=256 \
    -objectNamePrefix=loadgen \
```

```
    -objectSize=$(( 1024*32 ))
```

```
./run-s3bench.sh
```

2.2 观测结果

2.2.1 写操作

对于写操作而言，吞吐率为 79.65 MB/s，总耗时 0.100 秒。写操作最长耗时 0.009 秒，最短耗时 0.002 秒。99% 的写操作在 0.009 秒内完成，90% 的操作在 0.004 秒内完成

```
Results Summary for Write Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  79.65 MB/s
Total Duration:    0.100 s
Number of Errors:  0
------------------------------------
Write times Max:       0.009 s
Write times 99th %ile: 0.009 s
Write times 90th %ile: 0.004 s
Write times 75th %ile: 0.003 s
Write times 50th %ile: 0.003 s
Write times 25th %ile: 0.002 s
Write times Min:       0.002 s
```

2.2.2 读操作

对于读操作而言，吞吐率为 190.10 MB/s，总耗时 0.042 秒。读操作最长耗时为 0.002 秒，最短耗时为 0.001 秒。 99% 的读操作在 0.002 秒内完成，90% 的操作在 0.002 秒内完成

```
Results Summary for Read Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  190.10 MB/s
Total Duration:    0.042 s
Number of Errors:  0
------------------------------------
Read times Max:       0.002 s
Read times 99th %ile: 0.002 s
Read times 90th %ile: 0.002 s
Read times 75th %ile: 0.001 s
Read times 50th %ile: 0.001 s
Read times 25th %ile: 0.001 s
Read times Min:       0.001 s
```

3、尾延迟挑战

对 latency-collect 和 latency-plot 的代码进行修改以适用于自己搭建的 minio 服务器，通过执行 latency-collect 获取尾延迟分布数据，再执行 latency-plot 画出延迟分布图像和排队论模型预测

3.1 普通请求

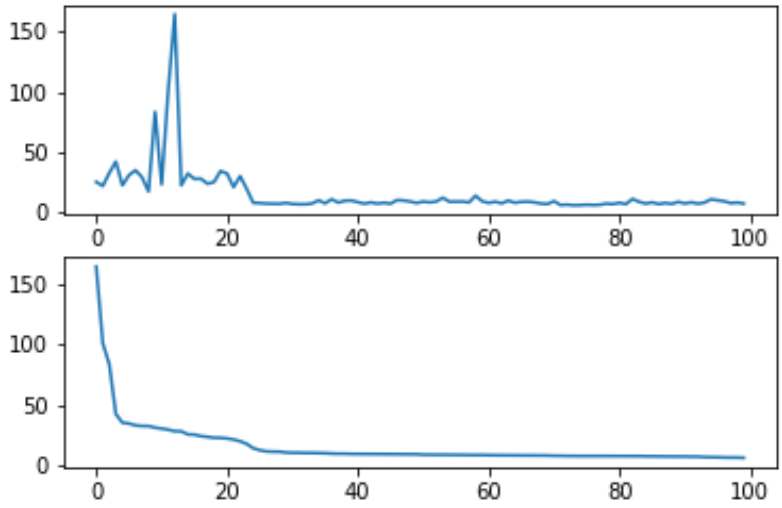
方法

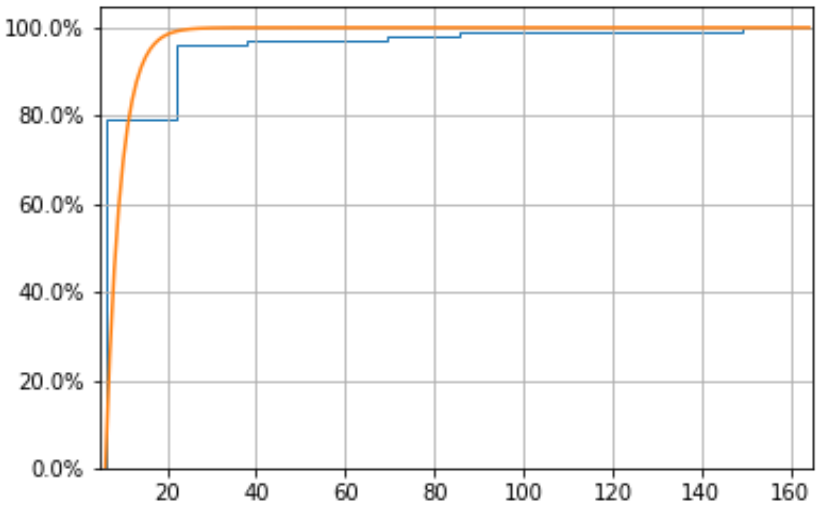
对每一个任务发送一个请求

分析

从以下实验数据可以观测到，最小延迟为 6.19 ms，平均延迟为 15.85 ms，50% 的请求在 8.81 ms 内完成，95% 的请求在 34.66 ms 内完成，99% 的请求在 101.31 ms 内完成，最大延迟为 164.85 ms

```
[min mean p50 p95 p99 max] 6.190061569213867 15.851507186889648 8.809089660644531
34.65604782104492 101.31001472473145 164.85309600830078
```





观察上面两张图可知，低延迟请求占大多数，同时也存在少量尾延迟现象

3.2 对冲请求

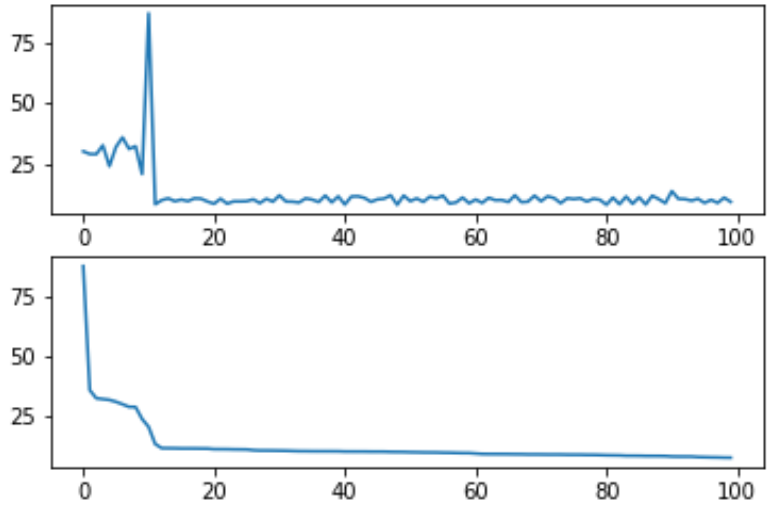
方法

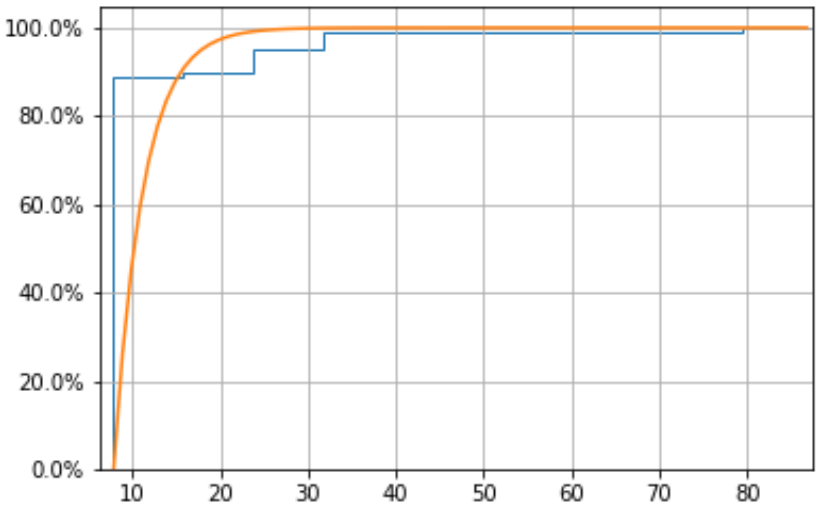
根据普通请求的延迟情况，取适当的延迟阈值，当第一个请求的延迟超过阈值，则发送第二个请求

分析

从以下实验数据可以观测到，最小延迟为 7.83 ms，平均延迟为 12.61 ms，50% 的请求在 10.11 ms 内完成，95% 的请求在 31.05 ms 内完成，99% 的请求在 35.85 ms 内完成，最大延迟降到了 87.42 ms

```
[min mean p50 p95 p99 max] 7.827997207641602 12.605107402801513 10.110139846801758
31.04877471923828 35.852909088134766 87.42195129394531
```





对比普通请求的延迟情况可知，普通请求在头部延迟上有优势，而对冲请求在尾部延迟上有优势。分析原因如下：

- 对冲请求增加了并发请求数，一定程度上增加了服务器的开销，因此导致头部延迟增加

- 对冲请求可以有效地处理尾延迟，因此尾部延迟减少

3.3 关联请求

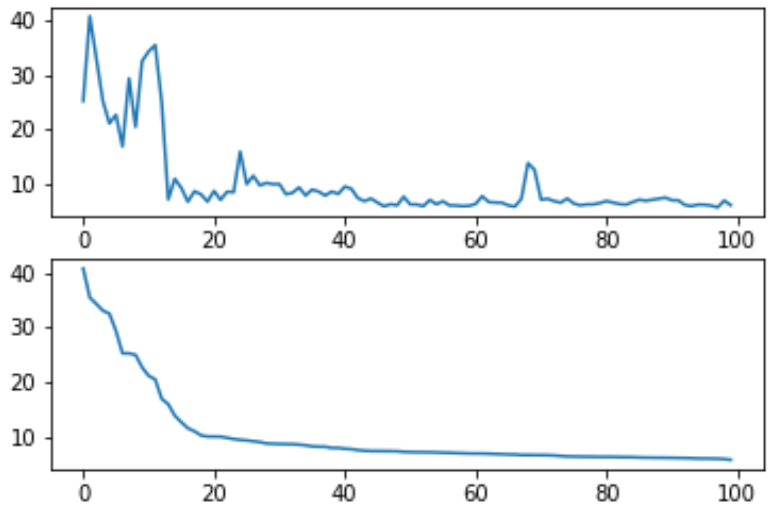
方法

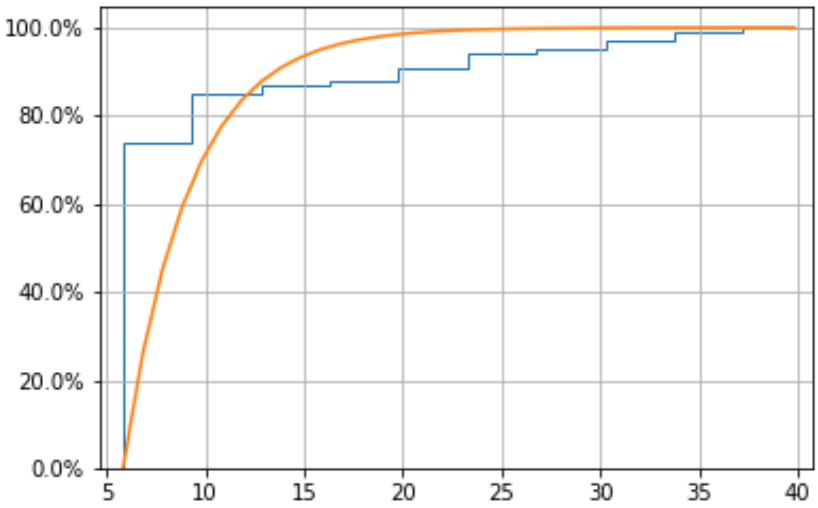
由于只有一台 minio 服务器，如果采用对同一个任务同时扇出两个请求的方法，服务器会由于并发访问的请求增加导致延迟增高。因此，以下采用对同一个任务串行发送两个请求，取其中较低的延迟的方式来模拟关联请求

分析

从以下实验数据可以观测到，最小延迟为 5.83 ms，平均延迟为 10.29 ms，50% 的请求在 7.24 ms 内完成，95% 的请求在 29.36 ms 内完成，99% 的请求在 35.43 ms 内完成，最大延迟降到了 40.72 ms

```
[min mean p50 p95 p99 max] 5.827188491821289 10.28742790222168 7.24482536315918
29.361248016357422 35.43496131896973 40.72284698486328
```





结合实验结果可知，由于低延迟请求占绝大多数，对一个任务发出两次串行的请求可以增大低延迟响应的概率，因此尾延迟情况得到了改善