华中科技大学

数据中心技术课程实验报告

院　系 计算机科学与技术学院

班　级 2110

学　号 M202173874

姓　名 戴秋爽

2021 年 1 月 2 日

实验一：系统搭建

系统搭建部分选择了 Apache License v2.0 下发布的对象存储服务器 MinIO，服务端可直接通过 web 访问，避免复杂的环境配置。完成 minio 下载后，启动 windows 环境中的命令行程序，启动 run-minio.cmd 搭建 MinIO 服务器，如图 1 所示。根据展示信息可以获取服务器地址和密钥进行网页登录。在新创建的 MinIO 服务器中新建一个名为“loadgen”的 Bucket，见图 2，完成系统搭建和预备操作。

图 1 minio 运行结果

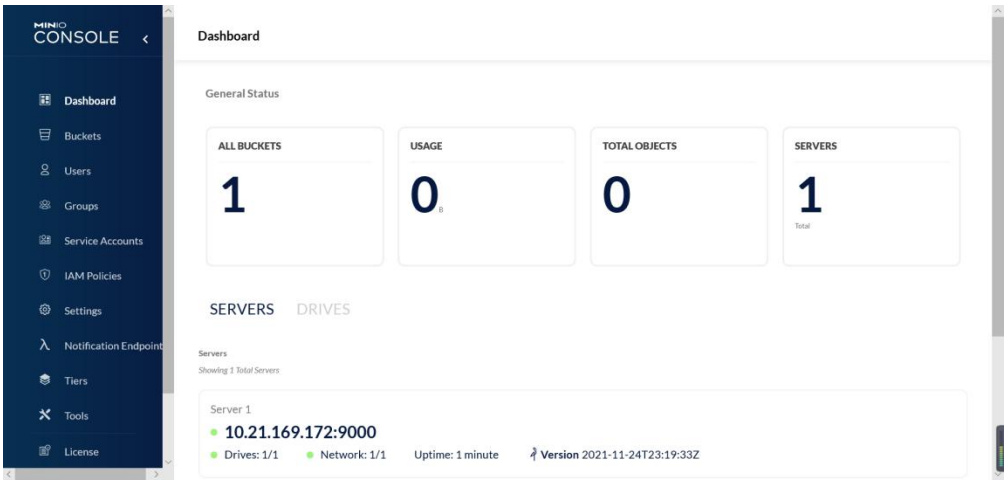


图 2 minio 服务器端 web 界面

实验二：性能观测

性能观测部分使用 S3 Bench 评测工具进行基准测试。直接运行 run-s3bench.cmd 命令可以观测到运行数据，见图 3.。初始设置客户端数量为 8，被测数量为 216，大小为 1024KB。


C:\Windows\system32\cmd.exe
D:\obs-tutorial-master>s3bench.exe -accessKey=hust -accessSecret=hust_
-objectNamePrefix=loadgen -objectSize=1024
Test parameters
endpoint(s): [http://127.0.0.1:9000]
bucket: loadgen
objectNamePrefix: loadgen
objectSize: 0.0010 MB
numClients: 8
numSamples: 256
verbose: %!d(bool=false)
Generating in-memory sample data... Done (1.9933ms)
Running Write test...
Running Read test...
Test parameters
endpoint(s): [http://127.0.0.1:9000]
bucket: loadgen
objectNamePrefix: loadgen
objectSize: 0.0010 MB
numClients: 8
numSamples: 256
verbose: %!d(bool=false)
Results Summary for Write Operation(s)
Total Transferred: 0.250 MB
Total Throughput: 0.18 MB/s
Total Duration: 1.409 s
Number of Errors: 0
Write times Max: 0.137 s
Write times 99th %ile: 0.091 s
Write times 90th %ile: 0.077 s
Write times 75th %ile: 0.060 s
Write times 50th %ile: 0.040 s
Write times 25th %ile: 0.024 s
Write times Min: 0.010 s
Results Summary for Read Operation(s)
Total Transferred: 0.250 MB
Total Throughput: 0.74 MB/s
Total Duration: 0.337 s
Number of Errors: 0
Read times Max: 0.026 s
Read times 99th %ile: 0.020 s
Read times 90th %ile: 0.015 s
Read times 75th %ile: 0.012 s
Read times 50th %ile: 0.010 s
Read times 25th %ile: 0.008 s
Read times Min: 0.004 s
Cleaning up 256 objects...
Deleting a batch of 256 objects in range {0, 255}... Succeeded
Successfully deleted 256/256 objects in 415.9833ms
D:\obs-tutorial-master>pause

图 3 性能测试结果

在初始设定下，在写操作的吞吐率为 0.18MB/s，总耗时 1.409 秒。写操作最长耗时 0.137 秒，最短耗时 0.010 秒。99%的写操作在 0.91 秒内完成，90%的操作在 0.077 秒内完成；对于读操作而言：吞吐率为 0.74MB/s，总耗时 0.337 秒。写操作最长耗时 0.026 秒，最短耗时 0.004 秒。99%的写操作在 0.020 秒内完成，90%的操作在 0.015 秒内完成。

设置不同的负载和被测数量，观测性能变化。

图 4 更改被测数量和负载后性能变化对比图

由第一行数据对比可以发现，随着被测数量增加，读写操作的总文件大小、吞吐率为和总耗时增加，其他性能变化不明显；由第二行数据对比可以发现，随着负载增加，读写的总数值都在增加，最长最短读写时间增加、相同百分比的读写完成率所用时间增加，尾延迟现象明显。

实验三：尾延迟挑战

1.尾延迟观测

运行 latency-collect 和 latency-plot 的代码获取尾延迟分布数据并绘制延迟分布图像（图 5）并用排队论模型拟合实测数据（图 6）。

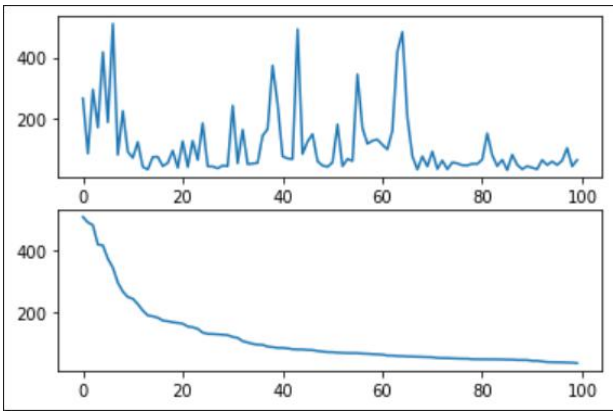


图 5 请求延迟分布情况

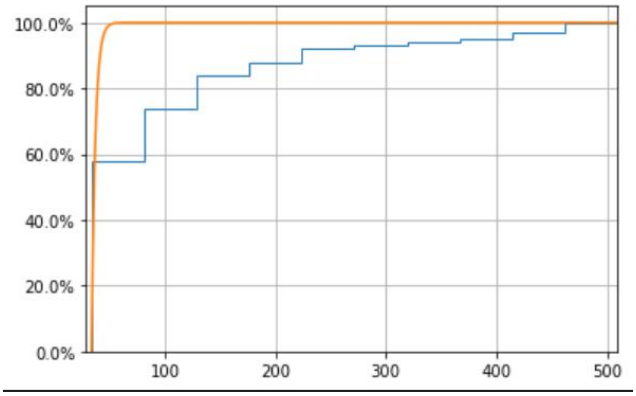
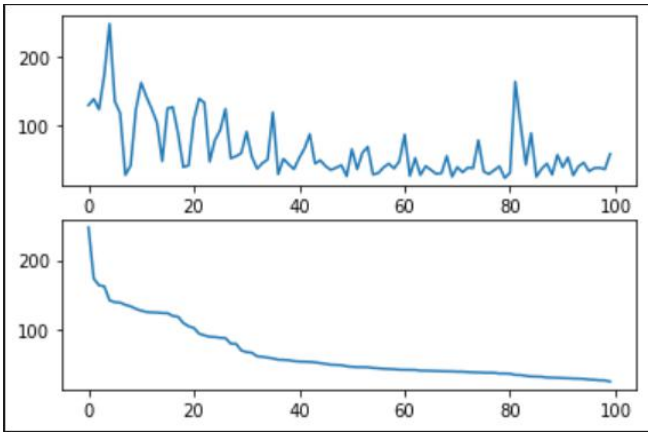


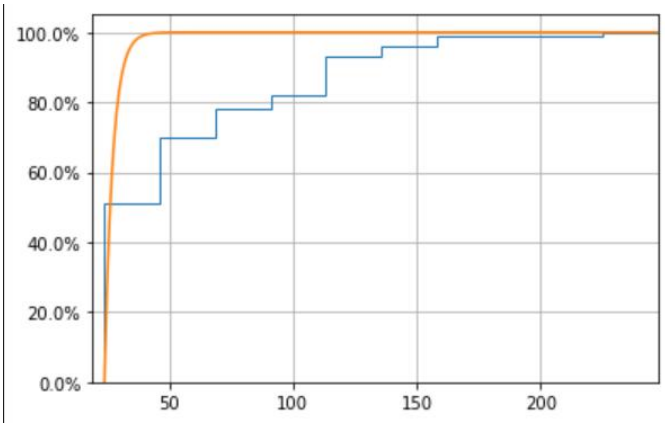
图 6 预测结果

从图 5 可以看出，延迟的波动较明显，但多数延迟偏低，只有部分数据产生了较高延迟影响整体的效果，出现了尾延迟现象。用排队论拟合实测数据得到图六，观察蓝色的折线可以发现，在 110ms 时有超过 80%数据完成，在 220ms 时有 90%的数据完成，410ms 时有 95%的数据完成。

2.对冲请求

为了降低尾延迟带来的影响尝试对冲请求，以 220ms 作为界限，如果请求超过 220ms 则重新发送请求。





可以发现，再加入对冲请求后，99%的请求都可以在 160ms 内发出，尾延迟现象得到了明显的改善。