

对象存储技术实验报告

姓名	宋诚
班级	数据中心技术 1 班
指导教师	施展
学院	计算机科学与技术学院
学号	M202173765

技术方案

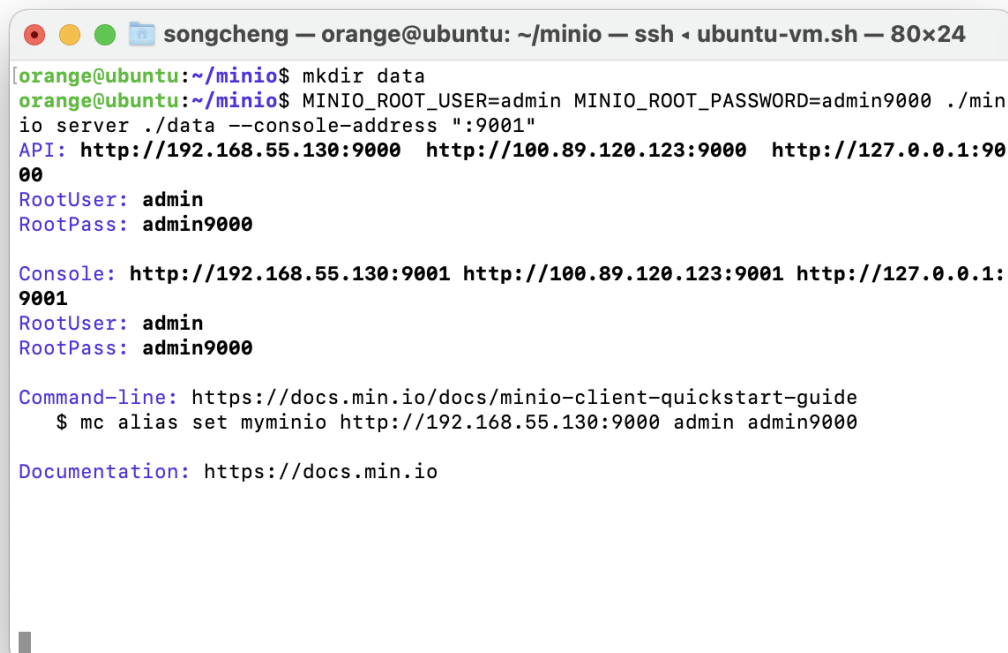
实验采用 MinIO, s3bench 和 Java 版 AWS S3 SDK。

实验一：系统搭建

这里选用能够最快速搭建的 MinIO 作为对象存储 server。server 系统环境为安装于 vmware 虚拟机中的 ubuntu 20 LTS。特别的事，为了连接 ubuntu 客户机和 Windows 宿主机系统，我使用 vmware 为客户机额外分配了一张虚拟的‘仅主机’网卡，实验中可以使用这张额外的虚拟网卡在主机和客户机之间直接通信。另外，为了在 MacBook 上能够通过 SSH 直接连接 ubuntu 系统，ubuntu 系统和 macOS 中均安装了 tailscale 用于搭建虚拟局域网。

安装 MinIO 的官方文档，在 Ubuntu 系统的终端里运行下面的命令安装 MinIO 并启动：

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
chmod +x minio
MINIO_ROOT_USER=admin MINIO_ROOT_PASSWORD=admin9000 ./minio
server ./data --console-address ":9001"
```



```
songcheng — orange@ubuntu: ~/minio — ssh - ubuntu-vm.sh — 80x24
[orange@ubuntu:~/minio$ mkdir data
orange@ubuntu:~/minio$ MINIO_ROOT_USER=admin MINIO_ROOT_PASSWORD=admin9000 ./minio
io server ./data --console-address ":9001"
API: http://192.168.55.130:9000 http://100.89.120.123:9000 http://127.0.0.1:9000
RootUser: admin
RootPass: admin9000

Console: http://192.168.55.130:9001 http://100.89.120.123:9001 http://127.0.0.1:9001
RootUser: admin
RootPass: admin9000

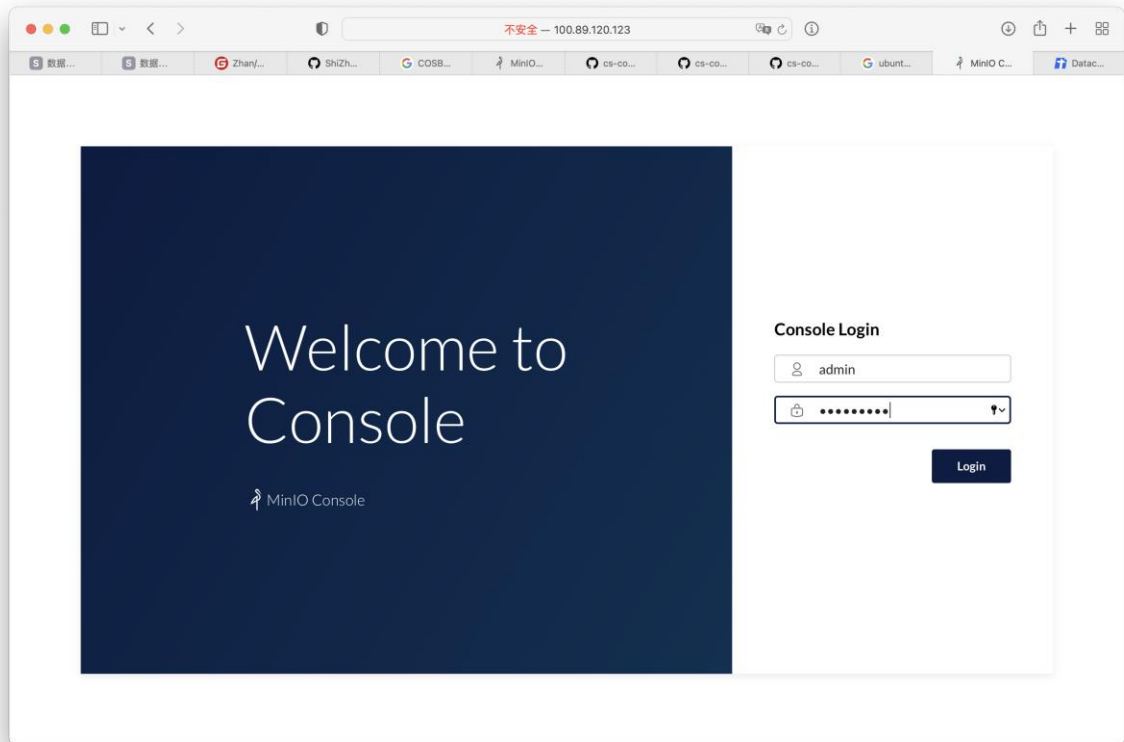
Command-line: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc alias set myminio http://192.168.55.130:9000 admin admin9000

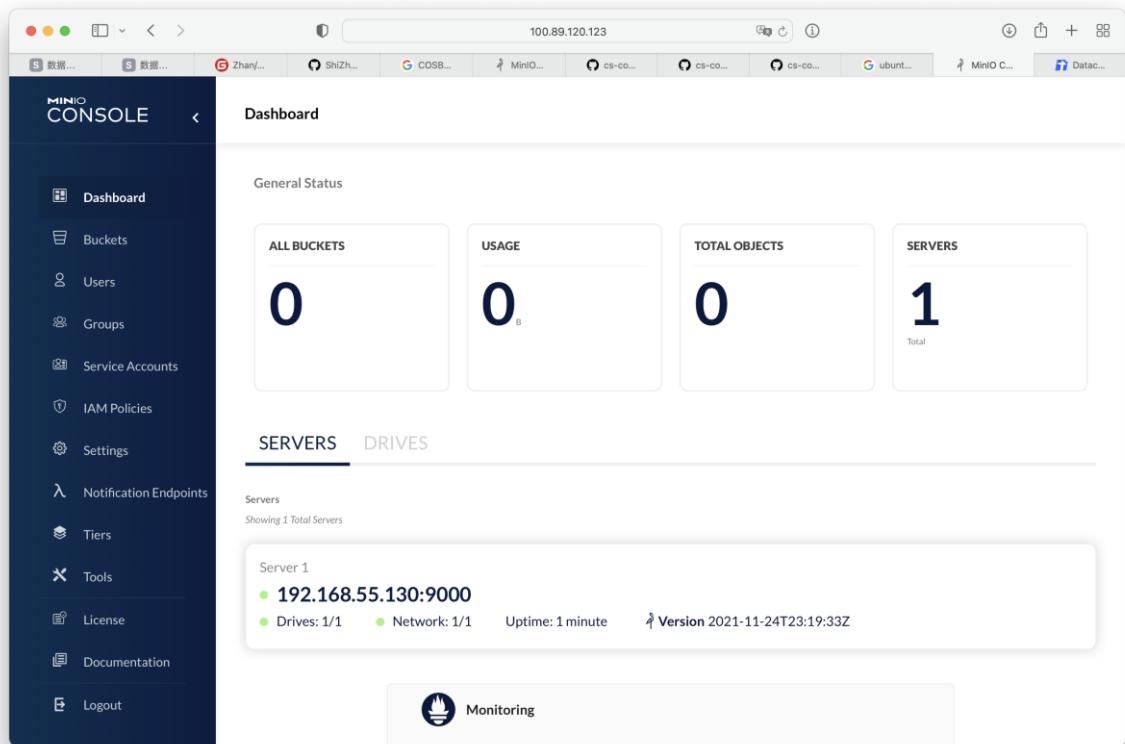
Documentation: https://docs.min.io
```

此外需要确保关闭 ubuntu 系统的防火墙。使用下面的命令关闭防火墙。

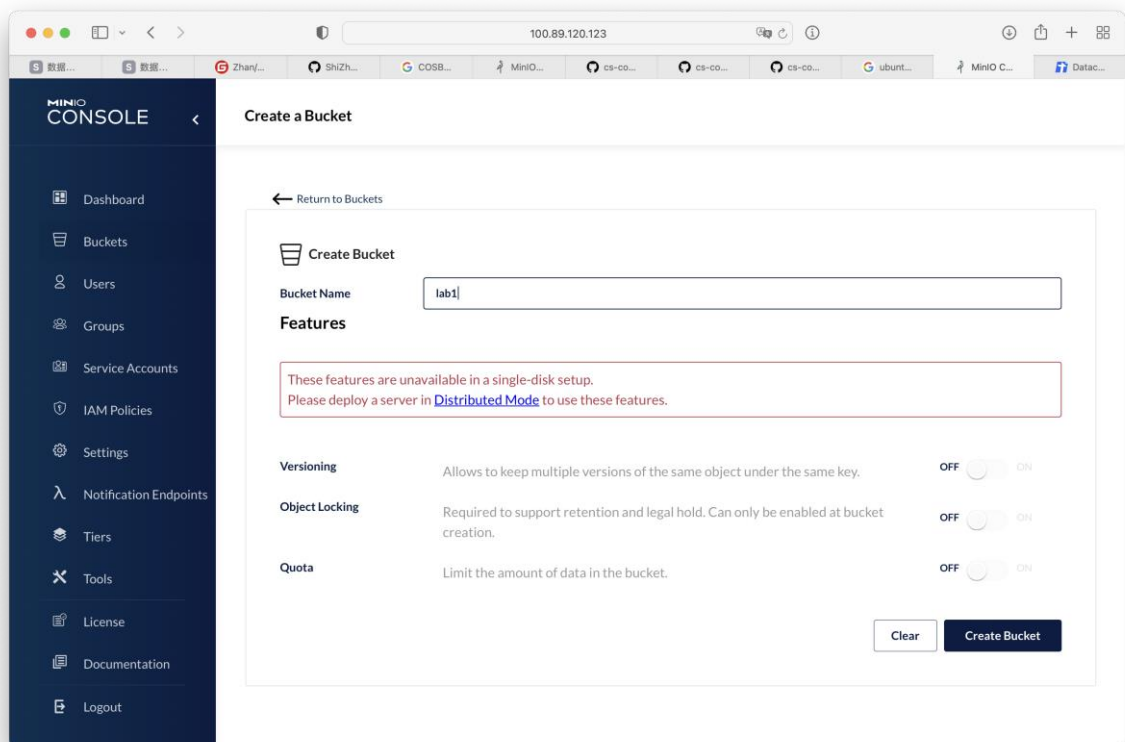
```
sudo ufw disable
```

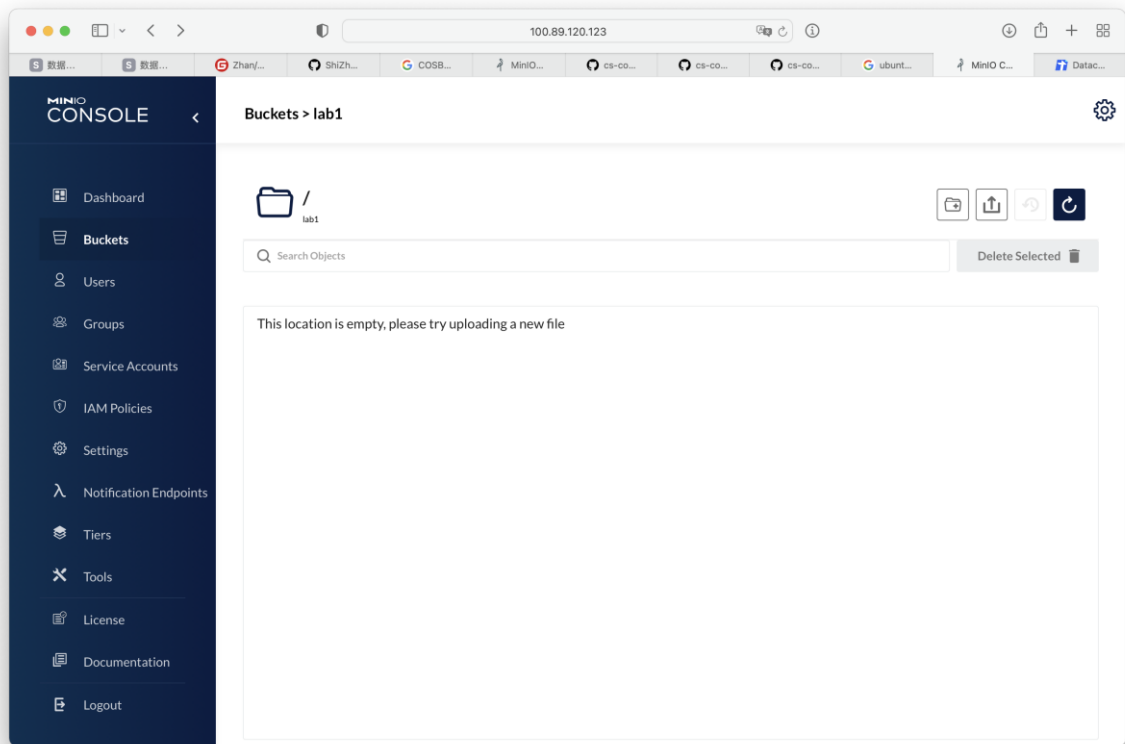
此时在 ubuntu 系统中使用 firefox 浏览器打开 <http://127.0.0.1:9001>/或者在宿主机 Windows 系统中使用 chrome 浏览器打开 ubuntu 客户机的‘仅主机’网卡地址的 9001 端口，或者在 macOS 中用 safari 浏览器打开 ubuntu 系统在虚拟局域网中的 IP 地址和端口 9001，可以看到 MinIO 提供的 Web 管理界面，并可以使用启动时命令中的用户名和密码登录。





登录后创建一个名为 lab1 的 bucket。





至此，测试使用的对象存储 **server** 搭建完毕。

实验二：性能观测

这里我使用 `s3bench` 作为性能测试工具。由于编译 `s3bench` 时遇到网络困难，这里采用预编译的 Windows 版本。测试工具运行在 Windows 主机上，通过虚拟网卡与 Ubuntu 虚拟机内的 MinIO server 通信。

测试时 `s3bench` 使用的配置如下：

```
E:\temp\s3bench>s3bench.exe          -accessKey=admin          -accessSecret=admin9000
-bucket=lab1          -endpoint=http://100.89.120.123:9000          -numClients=80
-numSamples=2560          -objectNamePrefix=loadgen          -objectSize=10240
    -verbose=true
Test parameters
endpoint(s):          [http://100.89.120.123:9000]
bucket:              lab1
objectNamePrefix: loadgen
objectSize:          0.0098 MB
numClients:          80
numSamples:          2560
verbose:             %!d(bool=true)
```

在如上配置的测试方案下，`s3bench` 的读写测试结果如下。90%的写入操作在 131 毫秒左右执行完毕，最大的写入耗时为 235 毫秒。最小写入耗时为 6 毫秒，写入操作的总体吞吐率为 9.32MB/s。90%的读取操作在 47 毫秒以内完成，最大读取耗时 502 毫秒，最小读取耗时为 3 毫秒，读取的总体吞吐率为 18.24MB/s。读取操作无论是在延迟还是吞吐率均明显好于写入操作，这是符合预期的。此时已经可以感受到读取操作有明显的长尾效应。

```
Results Summary for Write Operation(s)
Total Transferred: 25.000 MB
Total Throughput:  9.32
Total Duration:    2.682 s
Number of Errors:  0
-----
Write times Max:      0.235 s
Write times 99th %ile: 0.192 s
Write times 90th %ile: 0.131 s
Write times 75th %ile: 0.103 s
Write times 50th %ile: 0.077 s
Write times 25th %ile: 0.058 s
Write times Min:      0.006 s

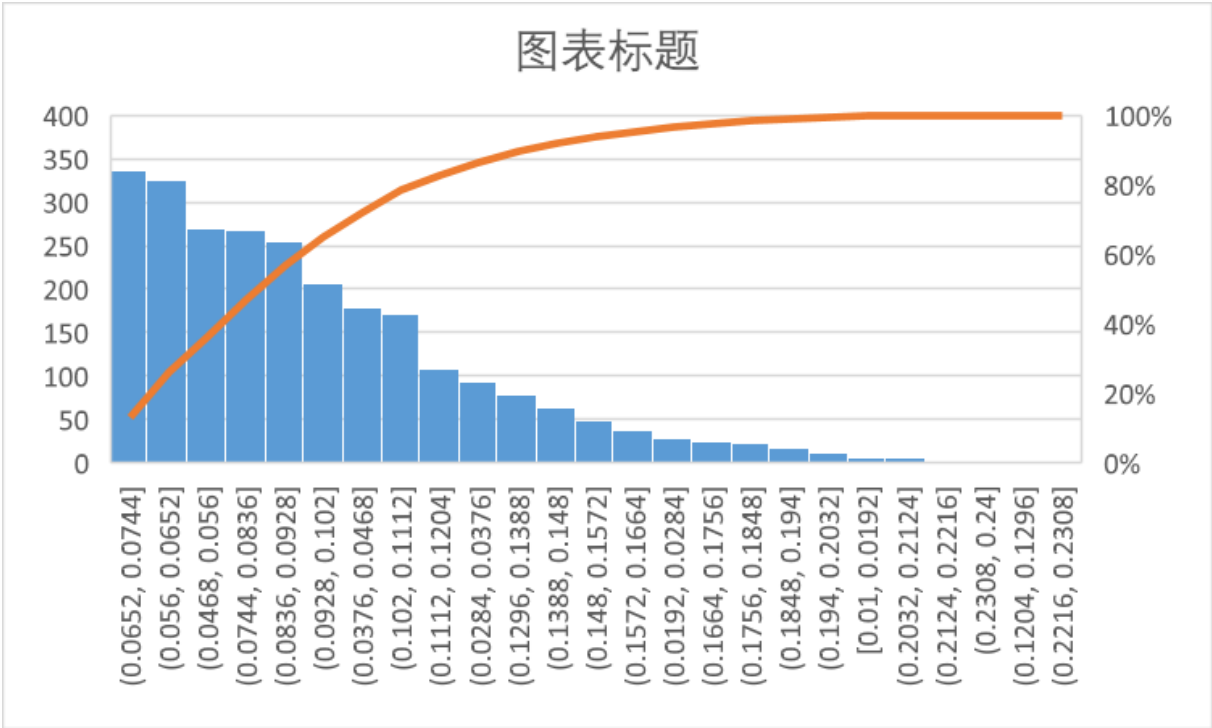
Results Summary for Read Operation(s)
Total Transferred: 25.000 MB
Total Throughput:  18.24
```

Total Duration:	1.370 s
Number of Errors:	0

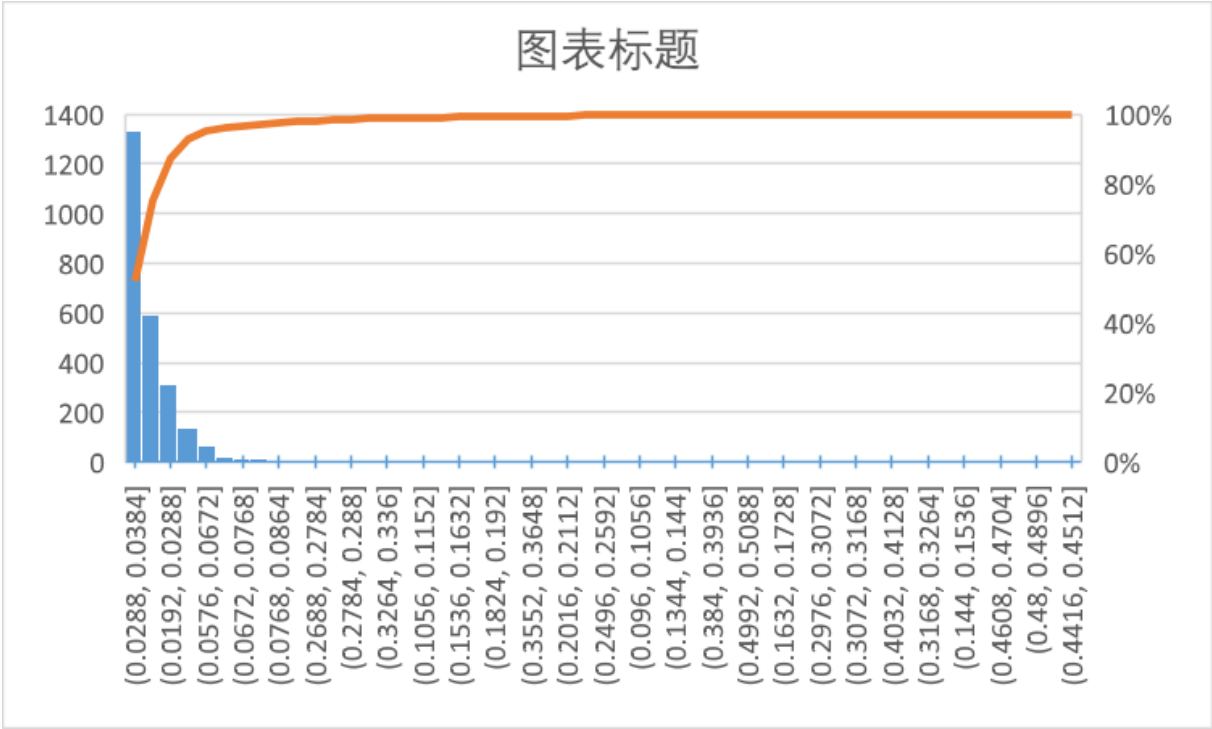
Read times Max:	0.502 s
Read times 99th %ile:	0.274 s
Read times 90th %ile:	0.047 s
Read times 75th %ile:	0.037 s
Read times 50th %ile:	0.032 s
Read times 25th %ile:	0.028 s
Read times Min:	0.003 s

我将 `s3bench` 的 `verbose` 配置为 `true`，就能得到详细的每一个请求的延迟和吞吐率数据。整理每一个请求的信息，绘制成为累计曲线图，可以看到读操作据有明显的长尾效应，绝大多数读取请求都能够在 67 毫秒左右完成。写入操作的长尾效应不明显。

write 累计曲线



read 累计曲线



实验三：尾延迟挑战

我尝试使用 Java 开发一个测试程序，使其具备对冲请求的功能，并检查对冲请求对请求耗时的作用。

实现方法：

通过设置请求的超时时间为期望的时间，一旦第一个请求超时，通过捕获超时异常的代码段发起第二个请求，以第一个请求的发起时间为起点，一第二个请求的结束时间为终点，记录整个请求过程的耗时，从而实现对冲请求。

Java 代码如下：

```
package com.example;

import io.minio.MinioClient;
import io.minio.PutObjectOptions;

import java.io.ByteArrayInputStream;
import java.util.ArrayList;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

/**
 * @rem -accessKey admin
 * @rem -accessSecret admin9000
 * @rem -bucket=loadgen Bucket for holding all test objects.
 * @rem -endpoint=http://100.89.120.123:9000 Endpoint URL of object storage service being
 * tested.
 * @rem -numClients=8 Simulate 8 clients running concurrently.
 * @rem -numSamples=256 Test with 256 objects.
 * @rem -objectNamePrefix=loadgen Name prefix of test objects.
 * @rem -objectSize=1024 Size of test objects.
 * @rem -verbose Print latency for every request.
 */
public class Client extends Thread {
    private final String bucket = "lab1";
    private final String endpointURL = "http://100.89.120.123:9000";
    private final String user = "admin";
    private final String pass = "admin9000";
    private final String objectNamePrefix = "test";
    private static final int numClients = 3;
    private final int numSamples = 50;
    private final int objectSize = 10240;
    private boolean shouldCleanup = false;
```

```

private boolean finished = false;
private static byte[] data = new byte[10240];
private final MinioClient client;
private long timeLimit = 35;

public boolean isFinished() {
    return finished;
}

public Client() throws Exception {
    this.client = new MinioClient(endpointURL, "admin", "admin9000");
}

public static void main(String[] args) {
    ArrayList<Client> clients = new ArrayList<>(numClients);
    ExecutorService pool = Executors.newCachedThreadPool();
    for (int i = 0; i < numClients; i++) {
        try {
            clients.add(new Client());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    for (Client client : clients) {
//        client.start();
        pool.submit(client);
    }
    while (true) {
        if (clients.stream()
            .anyMatch(
                c -> {
                    return !c.finished;
                }
            )) {
            Thread.yield();
        } else {
            for (Client client : clients) {
                client.shouldCleanup = true;
            }
            break;
        }
    }
}

private void tryPut(int i) {

```

```

        try {
            client.putObject(
                bucket,
                this.toString() + objectNamePrefix + i,
                new ByteArrayInputStream(data),
                new PutObjectOptions(data.length,
PutObjectOptions.MIN_MULTIPART_SIZE));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void clean() {
        for (int i = 0; i < numSamples; i++) {
            try {
                client.removeObject(bucket, this.toString() + objectNamePrefix + i);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    @Override
    public void run() {
        for (int i = 0; i < numSamples; i++) {
            long startTime = System.currentTimeMillis();
            Request request = new Request(startTime, i);
            request.start();
            while (true) {
                Thread.yield();
                synchronized (request.lock) {
                    try {
                        if (request.done) {
                            break;
                        } else {
                            if (System.currentTimeMillis() - request.startTime > timeLimit)
{
                                request.setCanceled(true);
                                request = new Request(startTime, i);
                                request.start();
                                request.join();
                                break;
                            } else {
                                Thread.yield();

```

```

        }
    }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
}
}
finished = true;
while (true) {
    Thread.yield();
    if (shouldCleanup) {
        clean();
        break;
    }
}
}
}

```

```

private class Request extends Thread {
    private boolean canceled = false;
    private final long startTime;
    private boolean done = false;
    private final int number;
    public final Object lock = new Object();

    public boolean isDone() {
        return done;
    }

    private Request(long startTime, int number) {
        this.startTime = startTime;
        this.number = number;
    }

    public void setCanceled(boolean canceled) {
        this.canceled = canceled;
    }

    @Override
    public void run() {
        tryPut(number);
        synchronized (lock) {
            if (!canceled) {
                System.out.println(System.currentTimeMillis() - startTime);
            }
        }
    }
}

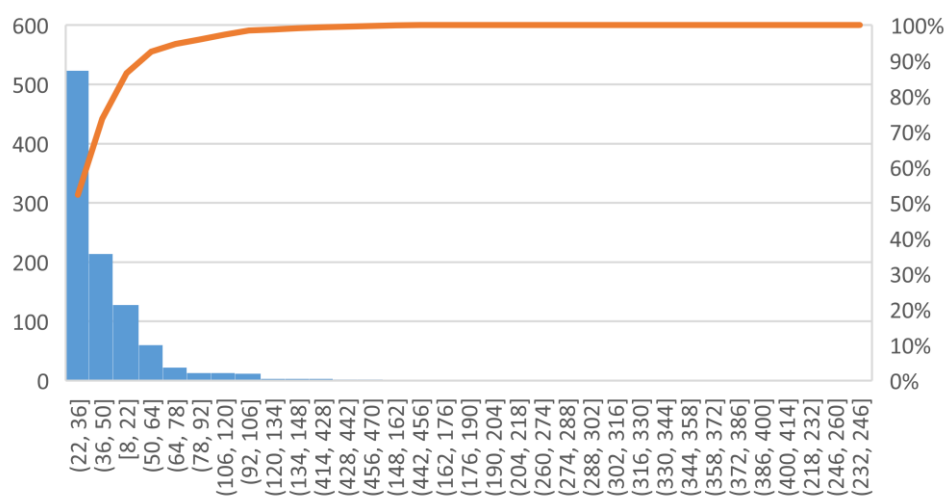
```

maven 用于引入依赖的 pom.xml 如下:

实验结果

下面两幅图对比相同 **java** 代码，在使用对冲请求和不使用对冲请求的时候读取请求的尾延迟性能。注意两幅图的横坐标不同，虽然有对冲读取的曲线上升更缓慢，但是在相同百分比的范围内横坐标更小，也就是延迟更低。

无对冲读取



有对冲读取

