数据中心实验

环境搭建

环境说明:

- 操作系统: Linux Manjaro KDE

使用Anaconda新建虚拟环境并安装相应依赖

```
# 新建datacenter环境
conda create -n datacenter python=3.9
conda activate datacenter
# 用于编写性能测试代码的python相关库，包括Amazon的用于s3 API的boto3
pip install boto3 throttle numpy pandas matplotlib tqdm
# 安装docker和golang
sudo pacman -S docker go
# 启用使用pacman包管理器安装的docker服务
sudo systemctl start docker
# 设置开机启动docker服务
sudo systemctl enable docker
# 使用go安装s3bench,用于性能观测尝试
go install github.com/igneous-systems/s3bench@latest
```

Object Storage Server

启动Minio

初学时使用较为完善的Minio

```
wget   http://dl.minio.org.cn/server/minio/release/linux-amd64/minio

chmod +x minio
```

编写minio启动脚本 `run_minio.sh` ，注意Web Console端口是 `9898` 但是实际的API端口是 `9000`

```
#!/usr/bin/zsh

export MINIO_ROOT_USER=admin
export MINIO_ROOT_PASSWORD=chenliwei

# Export metrics(dashboard)
export MINIO_PROMETHEUS_AUTH_TYPE="public"

# Use -C flags to store configuration file in local directory ./
./minio -C ./ server ./minio_server --console-address ":9898"
```

启动Minio

> ./run_minio.sh

启动S3Mock

使用docker运行Adobe的S3Mock，注意它还没有实现aws_access_key和region配置，见Issue#305: Support configure AccessKey & Region

```
# 9090是S3Mock的http端口, 9191是https端口, 环境参数initialBuckets中给出初始桶'loadgen'用于s3b
docker run -p 9090:9090 -p 9191:9191 --env initialBuckets=loadgen -t adobe/s3mock
```

性能观测

利用s3bench初步观测性能

编写s3bench脚本 `run_s3bench.sh` ，观测S3Mock的对象存储服务器的性能(如果要观测Minio的只需把endpoint的端口 `9090` 改为Minio的API端口 `9000`)

```
# Locate s3bench

s3bench=~/go/bin/s3bench

if [ -n "$GOPATH" ]; then
    s3bench=$GOPATH/bin/s3bench
fi

# -accessKey      Access Key
# -accessSecret   Secret Key
# -endpoint       对象存储服务器的API接口
# -bucket         在对象存储服务器中创建的bucket的名称，用于测试负载

$s3bench \
  -accessKey=admin \
  -accessSecret=chenliwei \
  -endpoint=http://127.0.0.1:9090 \
  -bucket=loadgen \
  -objectNamePrefix=loadgen \
  -numClients=8 \
  -numSamples=256 \
  -objectSize=$(( 1024*32 ))
```

```
~/workspace/datacenter ./run_s3bench.sh
Test parameters
endpoint(s):      [http://127.0.0.1:9090]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.0312 MB
numClients:       8
numSamples:       256
verbose:       %!d(bool=false)


Generating in-memory sample data... Done (279.481µs)

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://127.0.0.1:9090]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:       0.0312 MB
numClients:       8
numSamples:       256
verbose:       %!d(bool=false)


Results Summary for Write Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  30.19 MB/s
Total Duration:    0.265 s
Number of Errors:  0
------------------------------------
Write times Max:       0.049 s
Write times 99th %ile: 0.033 s
Write times 90th %ile: 0.015 s
Write times 75th %ile: 0.009 s
Write times 50th %ile: 0.007 s
Write times 25th %ile: 0.005 s
Write times Min:       0.002 s


Results Summary for Read Operation(s)
Total Transferred: 8.000 MB
Total Throughput:  47.08 MB/s
Total Duration:    0.170 s
Number of Errors:  0
------------------------------------
Read times Max:       0.023 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: 0.004 s
Read times 25th %ile: 0.003 s
Read times Min:       0.002 s


Cleaning up 256 objects...
```

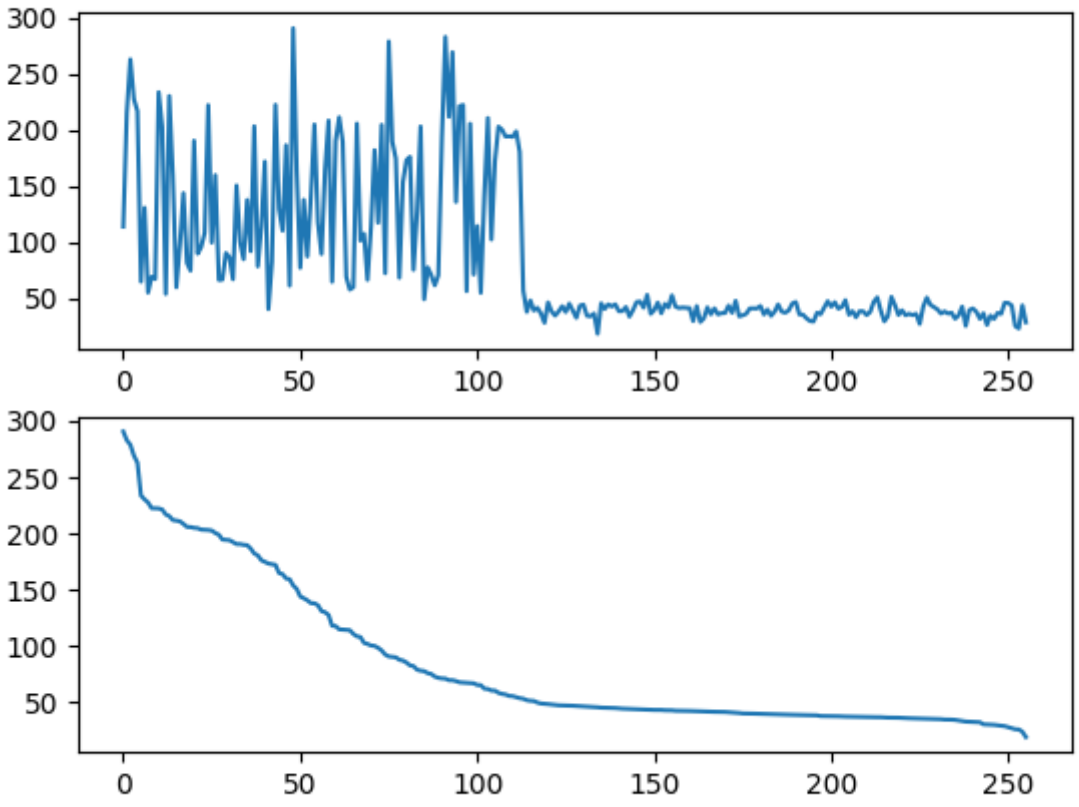
编写代码测试性能并汇总分析

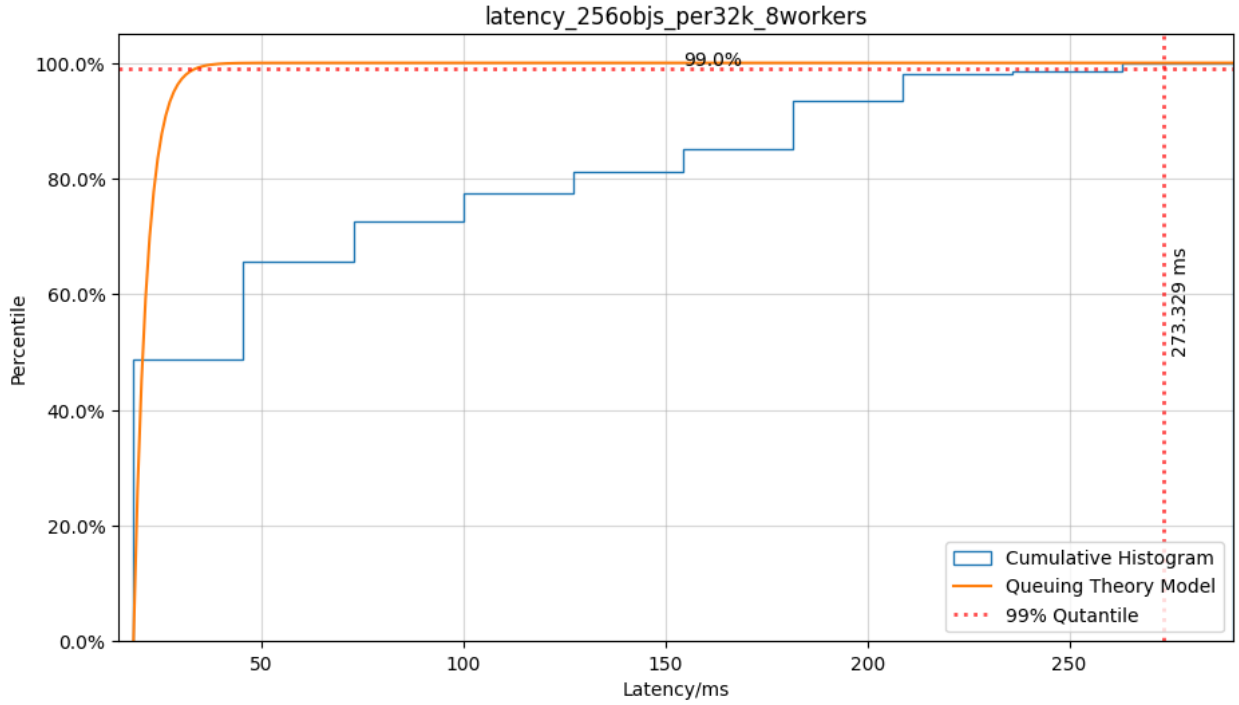
测试目标为S3Mock对象存储服务器，使用amazon提供的boto3 python API编写测试代码，具体代码见另附的实验代码文件。收集延迟信息并使用matplotlib依据排队论模型$F(t) = 1 - e^{-\alpha t}$，$\alpha = 0.3$拟合延迟数据绘制曲线，并添加99%分位点对应的实际延迟辅助线。

单次测试

先尝试对象个数 `256` ，对象尺寸 `32KB` ，并发数 `8`

```
from performance_test import PerformanceTester

tester = PerformanceTester()
tester.latency_collect(object_num=256, object_size=32, workers=8)
```



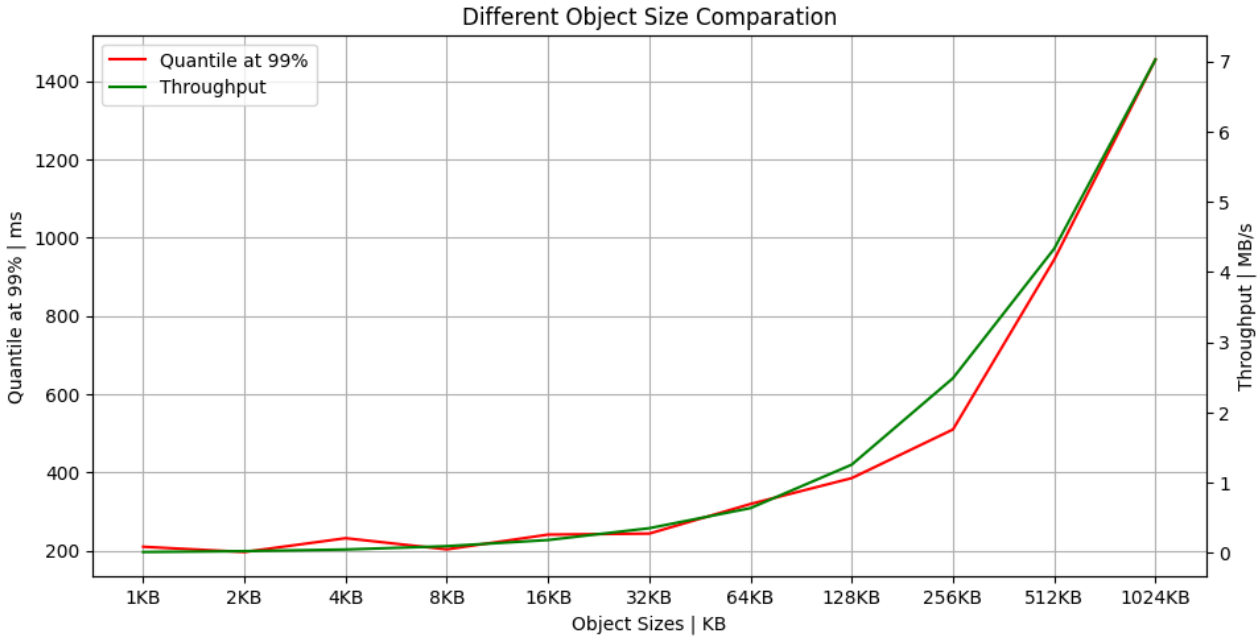


对照测试

改变对象尺寸，观测性能差异，从1KB每次乘2直到1024KB，对象个数固定为256， 并发数固定为8

```
tester.latency_compare(init_size=1, step=2, rounds=10, object_num=256, workers=8)
```

结果如下图所示，可以看到随着对象尺寸增大，吞吐率和尾延迟(延迟的99%分位数)都在增大



尾延迟挑战

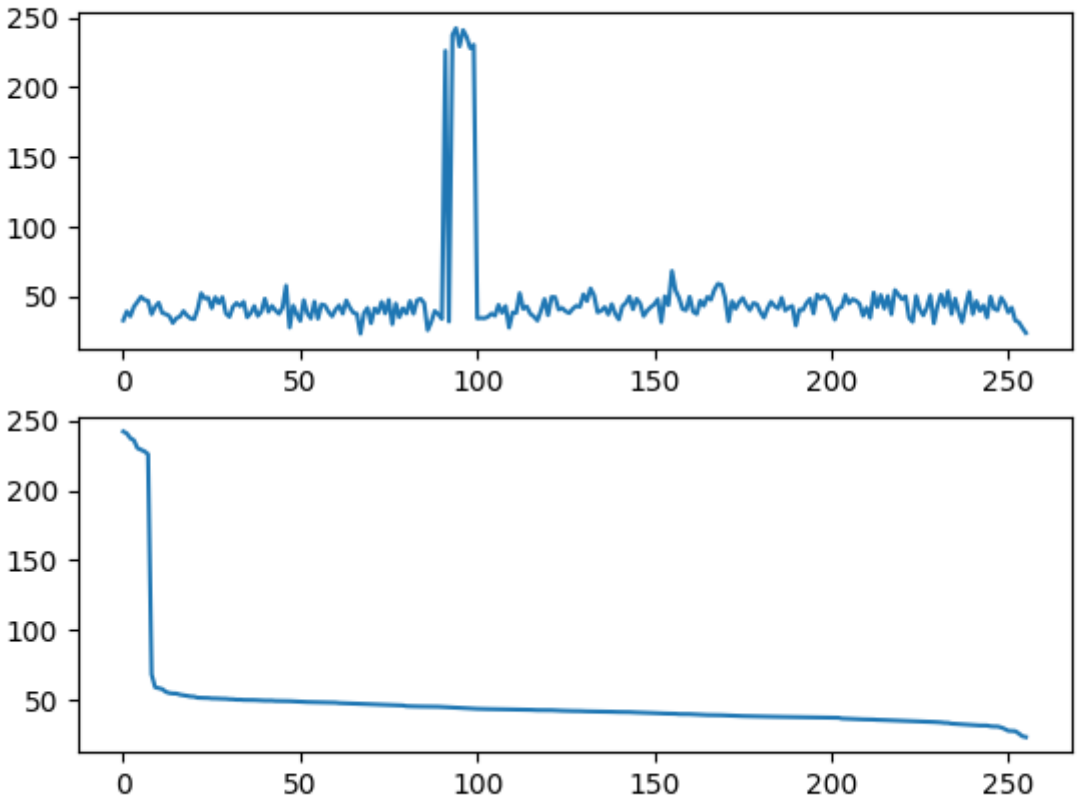
利用python `asyncio.wait_for()` 函数，在每个线程里异步执行向对象存储服务器写对象的请求，如果超时仍未返回(执行完毕)，则会自动取消并继续发送原请求，直至其延迟小于超时时间才会返回。核心代码如下所示:

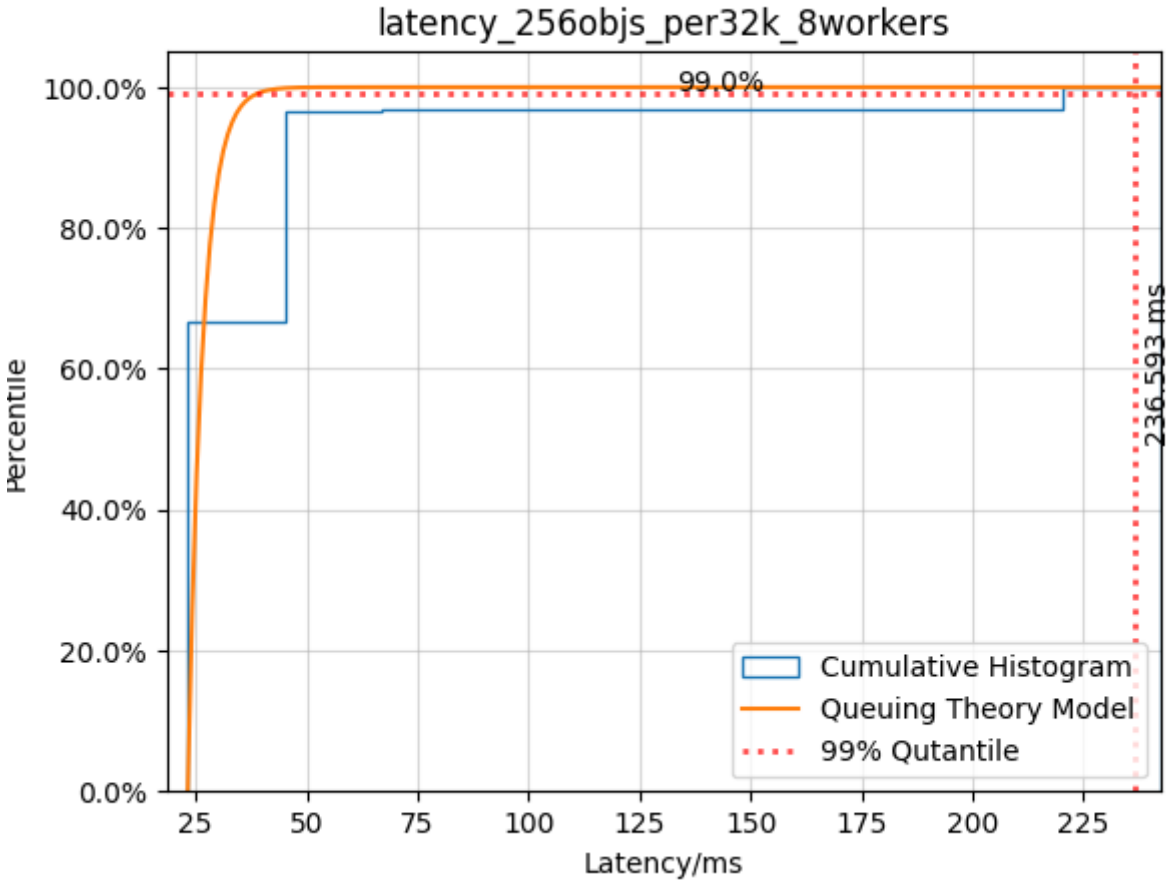
```
async def hedged_request(self, s3_res, i, suffix=''):
    '''
    模拟对冲请求，使用asyncio

    发送的请求如果在限定延迟后还未到达，立即发送第二个请求
    接收到任意一个请求的返回值后，取消其他请求

    Return
    ------
    latency (float) : 小于给定阈值的延迟，单位ms
    '''
    obj_name = "testObj%08d%s" % (i, suffix, )
    latency = 0

    async def write_object():
        start = time.time()
        s3_res.Object(self.test_bucket_name, obj_name).upload_file(self.test_file)
        end = time.time()
        return (end - start) * 1000

    # 如果超时则再发一次请求，直到某次请求的延迟小于timeout即可
    while latency == 0:
        try:
            # 超时时间设置为200ms，超时后自动取消
            latency = await asyncio.wait_for(write_object(), timeout=0.2)
        except asyncio.TimeoutError as timeout:
            print("[INFO] Time out!Submitting a new hedged request...")
            continue

    return latency
```

将函数对象传递给参数 `request_func=tester.hedged_request` ，与上一节一样，仍然是 `256` 个大小为 `32KB` 的对象，使用 `8` 个线程执行写入

```
# 模拟对冲请求
hedged_latency_file, _, _ = tester.latency_collect(object_num=256,
                                                   object_size=32,
                                                   workers=8,
                                                   request_func=tester.hedged_request)
tester.latency_plot(hedged_latency_file)
```

将运行结果绘制成图，如下图所示





吞吐率比较见下图控制台日志所示，与不采用对冲请求的写入测试相比，明显吞吐率大大提高

```
[Init] s3 server at http://127.0.0.1:9090
[0] Bucket Name: loadgen
****************************************
[INFO] Begin a new round of performance test
[Test Params]
对象个数: 256
对象尺寸: 32
并发数: 8
[Using Request Function] arrival_rate_max
Accessing S3 at http://127.0.0.1:9090: 100%|████████████████████████████████| 256/256 [00:02<00:00, 89.59it/s]
[Result Analysis]
Total Transferred : 8.000 MB
Total Duration : 21.996 s
Throughput : 0.36 MB/s
****************************************
[INFO] Begin a new round of performance test
[Test Params]
对象个数: 256
对象尺寸: 32
并发数: 8
[Using Request Function] hedged_request
Accessing S3 at http://127.0.0.1:9090: 100%|████████████████████████████████| 256/256 [00:02<00:00, 93.63it/s]
[Result Analysis]
Total Transferred : 8.000 MB
Total Duration : 11.071 s
Throughput : 0.72 MB/s
(datacenter) [penistrong@ServerPenistrong datacenter]$
```