Experiment：Local test for Serverless Cloud Function

Introduction

本实验主要是使用论文《*Function Bench : A Suite of Workloads for Serverless Cloud Function Service*》 中的测试样例对docker容器的性能进行测试，同时为了探究不同的资源分配方案对测量结果的影响，主要对CPU的时间片长度这个参数作对照实验，以模拟混合部署下资源无法单独占用的实际场景。

由于该论文中提出的函数都是基于不同云计算平台下的对比测试，在本地实验的时候需要进行代码的修改，例如去除与云服务器的信息通信和数据传输等，只保留本地运行。

Environment

参数	配置
处理器个数	8（0-7）
CPU型号	Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
CPU核心数	4
单核缓存大小	8192KB
内存大小	16GB
操作系统	Ubuntu 20.04

Construction

建立两个容器，并将工作目录和所需要的python第三方包都挂载到容器对应的目录之下。

容器1时间片设置为每1s占用0.2秒cpu资源：

```
docker run -itd -v "$(pwd):/usr/local/test_scripts"
-v "/usr/local/lib/python3.8/dist-
packages:/usr/local/lib/python3.8/dist-packages" \
    --cpu-period 1000000 --cpu-quota 200000 \
    --name container1 dolphin/ubuntu

docker exec -it container1 /bin/bash
```

容器2时间片设置为每1s占用1秒cpu资源：

```
docker run -itd -v "$(pwd):/usr/local/test_scripts"
-v "/usr/local/lib/python3.8/dist-
packages:/usr/local/lib/python3.8/dist-packages" \
    --cpu-period 1000000 --cpu-quota 1000000 \
    --name container2 dolphin/ubuntu

docker exec -it container2 /bin/bash
```

除了时间片不相同之外，其他cgroup参数均保持一致。

Test

（时间单位均为秒，默认结果均测量10次，取平均值）

cpu-memory

该部分主要对主机内部的计算资源和存储资源进行测试，所提供的任务对CPU和内存的敏感性较高，类型上从大方向可以分为数学计算、工程开发和机器学习。

1. chameleon

   测试使用chameleon渲染页面的时间延迟，这里的页面内容是一个可以指定列和行的矩阵。

   ```
   python3 ./aws/cpu-
   memory/chameleon/lambda_function.py
   ```

   参数设置：row=col=400

   测试结果：

   容器1：mean：4.103，std：0.238

   容器2：mean：0.920，std：0.059

   比值：4.459

2. feature_generation

   主要是针对于文本的处理，这里以feature_extractor为例，任务内容是抽取语料库中所有单词，具体包括小写化字母、删除其他字符、建立列表结构、序列化。测量该过程的时间。

   需要的python第三方库：pandas，pytz，dateutil

   ```
   python3 ./aws/cpu-memory/feature_generation/feature_extractor/lambda_function.py
   ```

   参数设置：选择的语料为reviews10mb.csv

   测试结果：

   容器1：mean：4.650，std：0.390

   容器2：mean：0.976，std：0.020

   比值：4.793

3. float_operation

   执行n轮浮点数操作，每一轮均执行sin，cos，sqrt操作

   ```
   python3 ./aws/cpu-memory/float_operation/lambda_function.py
   ```

   参数设置：n=1000000

   测试结果：

   容器1：mean：2.594，std：0.428

   容器2：mean：0.584，std：0.130

   比值：4.441

4. image_processing

对图像进行处理，包括左右/上下翻转，旋转，滤镜等操作

在运行时会发现镜像中缺乏相应的依赖共享文件，类似如下的错误，这是由于ubuntu镜像相当于是一个简化版本的虚拟机，只保留一些基础的依赖。

```
ImportError: libjpeg.so.8: cannot open shared
object file: No such file or directory
```

按照要求进行下载即可：

```
apt install -y libjpeg8 libimagequant0 libtiff5
```

参数设置：图片为animal-dog.jpg

测试结果：

容器1：mean：10.299，std：0.746

容器2：mean：2.282，std：0.122

比值：4.513

5. linpack

求解线性方程组，随机生成指定维度n的矩阵和向量，用该矩阵和向量组成的线性方程组进行求解，测量求解过程的时间。

```
python3 ./aws/cpu-
memory/linpack/lambda_function.py
```

参数设置：n=200，为减少随机数生成导致的误差，这里测试次数设为100次

测试结果：

容器1：mean：0.113，std：0.384

容器2：mean：0.022，std：0.123

比值：5.136

6. matmul

   矩阵乘法，输入为n*n的矩阵，执行矩阵乘法操作。

   ```
   python3 ./aws/cpu-memory/matmul/lambda_function.py
   ```

   参数设置：n=300，为减少随机数生成导致的误差，这里测试次数设为100次

   测试结果：

   容器1：mean：0.265，std：0.432

   容器2：mean：0.054，std：0.201

   比值：4.907

7. model_serving

   选择其中两个用例进行测试

   - ml_lr_prediction

   使用已训练完成的模型进行测试数据的预测，首先是读取保存模型参数的pk文件，然后是读入需要预测结果的测试集数据，作相同预处理之后进行预测。

   ```
   python3 ./aws/cpu-memory/model_serving/ml_lr_prediction/lambda_function.py
   ```

   需要的python第三方库：sklearn

   参数设置：训练集为reviews10mb.csv，测试集为reviews20mb.csv

   测试结果：

容器1：mean：29.914，std：0.248

容器2：mean：6.587，std：0.205

比值：4.541

- ml_video_face_detection

使用相关的库进行video转换并且在转化过程中对每一帧进行实时的脸部识别（不局限于人类面部）

```
python3 ./aws/cpu-memory/model_serving/ml_video_face_detection/lambda_function.py
```

同样出现了共享文件缺失问题，安装如下的依赖：

```
apt install -y libgl1-mesa-glx libglib2.0-dev
```

需要的python第三方库：python-opencv

参数设置：输入是一段testVideo001.mp4，由于时间过长，这里只测试了一次

测试结果：

容器1：mean：1735.517

容器2：mean：357.011

比值：4.861

8. model_training

模型训练，本例中使用的是最基础的逻辑回归模型，训练集是reviews10mb.csv，将每条评论都转为tf-idf向量，作为输入特征，训练目的是用户的评分（1～5）。

```
python3 ./aws/cpu-memory/model_training/lambda_function.py
```

需要的python第三方库：sklearn，pandas

参数设置：训练集reviews10mb.csv

测试结果：

容器1：mean：31.373，std：1.466

容器1：mean：6.397，std：0.591

比值：4.904

9. pyaes

   对随机生成的字符串进行加密解密算法测试，采用的是pyaes下的加密算法

   ```
   python3 ./aws/cpu-memory/pyaes/lambda_function.py
   ```

   需要的python第三方库：pyaes

   参数设置：字符串长度为1024，循环次数为16次

   测试结果：

   容器1：mean：0.896，std：0.243

   容器2：mean：0.177，std：0.072

   比值：5.062

10. video_processing

    对video进行处理，主要是逐步抽取video中的帧，并进行灰化，将结果保存为一个新的avi文件。

    ```
    python3 ./aws/cpu-
    memory/video_processing/lambda_function.py
    ```

    需要的python第三方库：opencv-python

参数设置：输入文件testVideo001.mp4，由于时间过长，这里只测试了一次

测试结果：

容器1：mean：235.977

容器2：mean：43.626

比值：5.409

disk

该部分主要对磁盘的读写性能进行测试

1. sequential_disk_io

   顺序读写磁盘块，首先创建新的文件，往文件中随机写入指定大小的字符，并强制写到磁盘上（防止之停留在内存中），然后在顺序读取所新建的文件。测量该过程的延迟。

   ```
   python3
   ./aws/disk/sequential_disk_io/lambda_function.py
   ```

   参数设置：读写文件大小为8MB，缓冲区大小为512字节

   测试结果：

指标	容器1	容器2	比值
写带宽	6.093 (std = 1.587)	23.001 (std = 1.659)	
写延迟	1.415 (std = 0.396)	0.350 (std = 0.030)	4.042
读带宽	154.202 (std = 88.498)	154.249 (std = 27.239)	
读延迟	0.187 (std = 0.283)	0.053 (std = 0.006)	3.528

2. random_disk_io

   随机读写磁盘块，首先创建新的文件，重复往文件中随机位置写入的字符，直到文件大小为指定大小，并强制写到磁盘上（防止之停留在内存中），然后在随机读取所新建的文件。测量该过程的延迟。

   ```
   python3
   ./aws/disk/random_disk_io/lambda_function.py
   ```

   参数设置：读写文件大小为8MB，缓冲区大小为512字节

   测试结果：

指标	容器1	容器2	比值
写带宽	33.420 (std = 13.980)	34.157 (std = 5.295)	
写延迟	0.372 (std = 0.334)	0.240 (std = 0.042)	1.55
读带宽	35.201 (std = 50.131)	89.020 (std = 17.661)	
读延迟	0.641 (std = 0.293)	0.092 (std = 0.015)	6.95

3. gzip_compression

压缩文件延迟测试，打开一个文件，调用gzip的库将内容数据进行压缩，生成一个tar.gz类型的压缩包，测量该过程的延迟。

```
python3
./aws/disk/gzip_compression/lambda_function.py
```

参数设置：读写文件规模为8MB，缓冲区大小为512字节

测试结果：

容器1：mean：1.841，std：0.220

容器2：mean：0.444，std：0.054

比值：4.146

network

该部分测试的是在发送网络请求和接受网络响应的延迟，由于没有部署云服务器，因此在本测试中只对访问公网资源进行测试。

1. json_dumps_loads

   测量从指定网站上下载内容并进行序列化所需的时间

   ```
   python3
   ./aws/network/json_dumps_loads/lambda_function.py
   ```

   参数设置：link="http://www.vizgr.org/historical-events/search.php?format=json&begin_date=-3000000&end_date=20151231&lang=en"

   由于访问的是外网，网络延迟非常明显

   测试结果：

指标	容器1	容器2	比值
网络延迟	79.996	35.024	2.284
序列化延迟	0.070	0.080	0.875

Conclusion

1. cpu-memory类型的实验

   容器1和容器2表现出相对规律性的性能关系，从延迟的角度，容器1的延迟大概是容器2的4～6倍，这与时间片的分配策略呈现高度相关性（容器1为每1秒占用0.2秒cpu，容器2为每1秒占用1秒cpu），可以验证出在保证其他条件不变的情况下，执行效率和时间片长度呈现线性化关系。

需要指出的是，对于运行时随机生成操作数的任务（例如malmul和linpack），其函数的计算复杂程度和随机生成的数有关，因此多次执行时的延迟也出现大幅度的波动（从方差可以看出）。不过从结果上看，容器1的延迟仍然显著高于容器2。

2. disk类型的实验

   从实验结果可以看出，相比较于cpu-memory类型的实验，disk类型实验的结果现象并不明显，尽管从定性的角度分析容器2的性能仍比容器1要高，但已经很难发现线性关系，推测这是由于该类实验中与磁盘读写相关，而两个容器在此保持一致，抵消了部分由于cpu导致的性能差异。

3. network类型的实验

   network类型实验的结果现象同样不明显，且由于外网访问延迟过长，两者性能都不高。