

**2017** 级

《物联网数据存储与管理》课程

**实 验 报 告**

**姓 名 陈 旺**

**学 号 U201714757**

**班 号 物联网1701班**

**日 期 2020.05.31**

目录

[一 实验目的 3](#_Toc41854292)

[二 实验背景 3](#_Toc41854293)

[三 实验环境 3](#_Toc41854294)

[四 实验内容 3](#_Toc41854295)

[1 语言环境搭建 3](#_Toc41854296)

[2 对象存储系统搭建 4](#_Toc41854297)

[3 对象存储系统性能测试 4](#_Toc41854298)

[五 实验过程 4](#_Toc41854299)

[1 语言环境搭建 4](#_Toc41854300)

[2 服务端mock-s3安装 5](#_Toc41854301)

[3 客户端的两种方式——aws-shell和boto3API编写的python程序 6](#_Toc41854302)

[4 系统性能测试的两种方式——测试工具s3-benchmark和自行编写的测试程序（附带图形界面） 9](#_Toc41854303)

[六 结果分析 10](#_Toc41854304)

[七 实验总结 14](#_Toc41854305)

## 一 实验目的

1. 熟悉对象存储技术，代表性系统及其特性；

2. 实践对象存储系统，部署实验环境；

3. 在构建好的对象存储系统上，进行初步测试和分析；

4. 了解基于对象存储构建的实际应用的基本原理。

## 二 实验背景

本实验是对应于“物联网数据存储与管理”课程的实验课，通过本实验，加深对课程的理解和应用。

本次实验为对象存储入门实验，其中主要的部分有：基础环境搭建；对象存储服务器端准备；对象存储客户端准备；对象存储测评工具的使用（基于API对象测评编程）。

基础环境在下一节详细介绍。

关于对象存储使用的服务端、客户端和测评工具在下一节具体说明。

实验难度的选择：

服务端：mock-s3

客户端：boto3 API python程序与aws-shell（都做了）

系统测试：s3-benchmark（简要做）与python boto3系统性能测试程序（详细做）

## 三 实验环境

1 网络环境：本实验的完全使用个人笔记本电脑，没有接入互联网（PC联网，运行环境不需要互联网）。

2 语言环境：python3.7.6（anaconda3，jupyter）、go1.14.3 linux/amd64。

3 实验工具：服务端-mock-s3，客户端s3bench（同时也利用boto3的API编写了客户端基本传输处理命令的python代码，测试可用）、测试工具-s3-benchmark（同时也编写了编写了测试程序，且可以通过命令行+参数运行、并图形展示）。

## 四 实验内容

### 语言环境搭建

所需的各种语言环境为python（anaconda）、java、go

此外，如何使用API进行测试，需要额外获取aws-sdk-jaava或boto3。

### 对象存储系统搭建

本实验选择基于python的对象存储系统，服务端搭建mock-s3，客户端做出了两种模式——aws-shell和利用boto3API编写的python程序。

### 对象存储系统性能测试

在测试方式上，也有两种选择——利用s3-benchmark测试工具和利用boto3API编写的python测试程序。存储系统的测试，主要注重这几个方面——对象大小对传输速率的影响、客户端数量对传输速率的影响、客户端数量对%点延迟的影响。

## 五 实验过程

### 1 语言环境搭建

说明：由于整个实验都选择依赖于python和golang，所以可以不需要安装java语言环境。

python自带3.6，安装3.7并链接到3.7版本。如下图1所示。

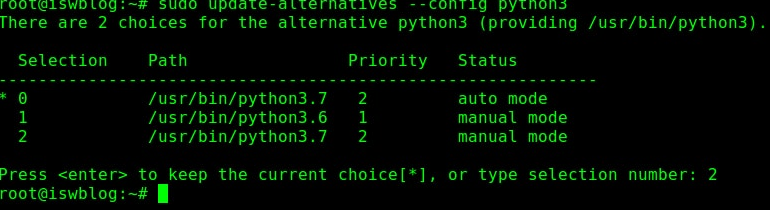


图1 python3.6更新到3.7

go语言安装，首先在golang官网下载64位go1.14.3 Linux安装包，解压到指定目录下，然后配置环境变量。之后，要现在golang.org/x library，整个go的环境才基本搭建完整。如下图2所示。

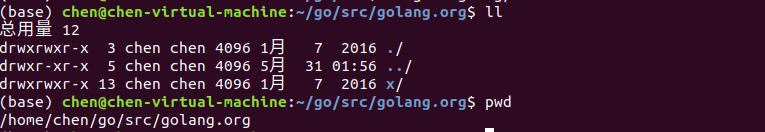
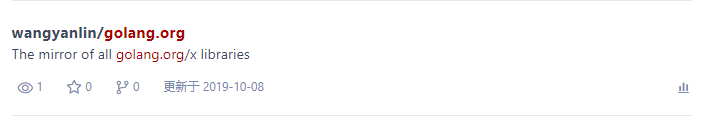
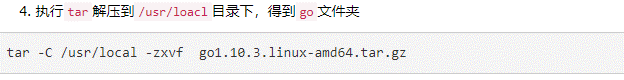
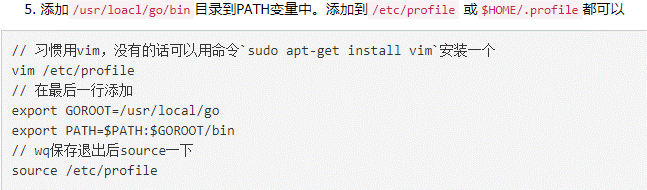


图2 golang环境配置

### 2 服务端mock-s3安装

mock-s3基于python环境，获取资源包后，进入目录，然后可选择两种运行方式——先安装后运行和直接运行（.py文件，带参数），具体如下图3所示。

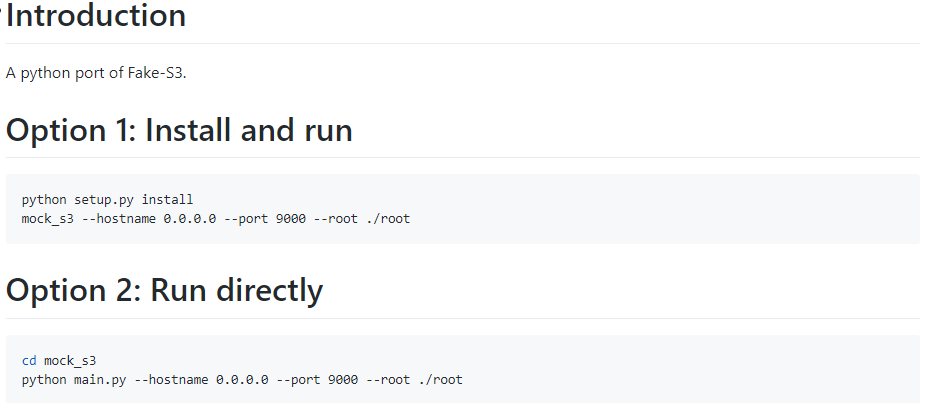
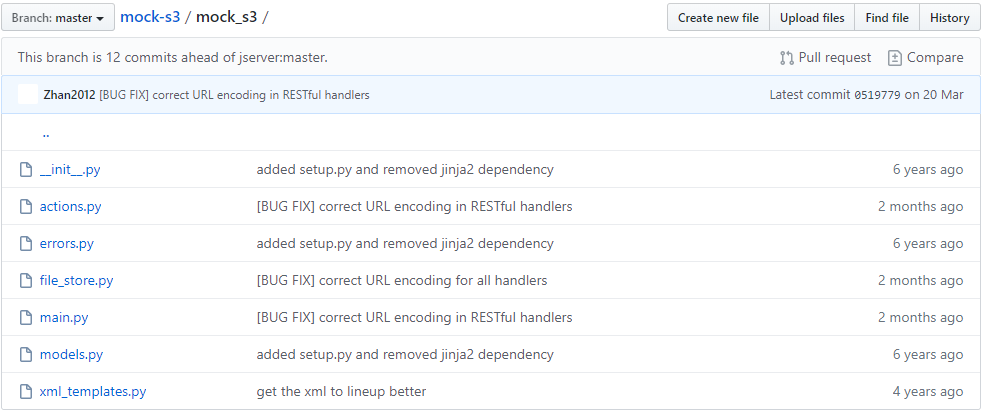
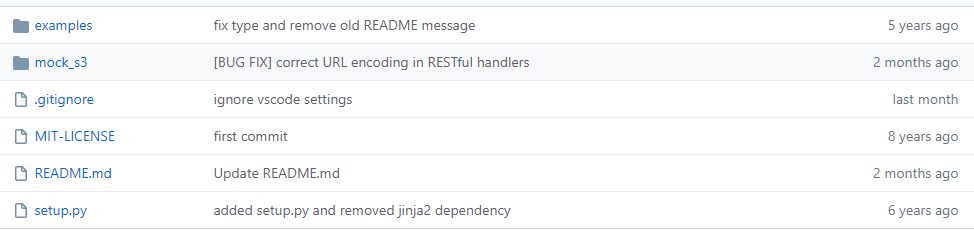


图3 mock-s3的安装和运行

### 3 客户端的两种方式——aws-shell和boto3API编写的python程序

aws-shell安装如下图4所示。



图4 aws-shell安装

由于是国内，此pip命令安装源最好更换为国内镜像，比如用清华的源。如图5所示。



图5 使用清华大学的安装源更换pip默认安装源

aws-shell 操作s3的命令行方式如下图6所示

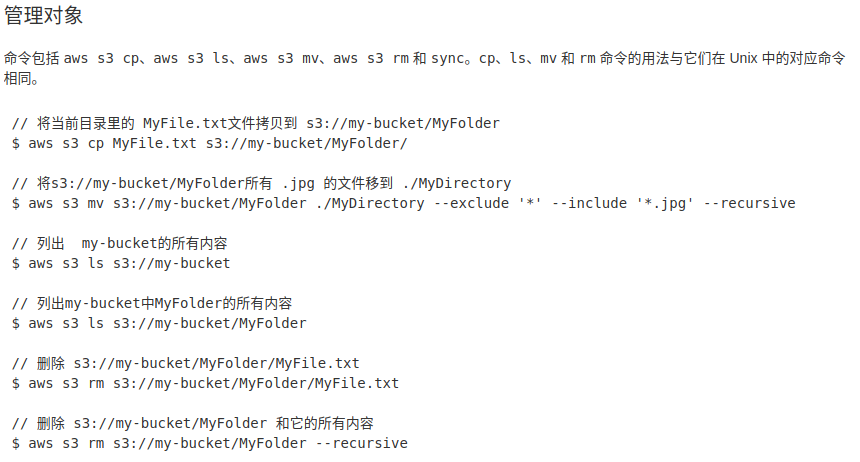


图6 aws-shell操作s3的几个简单示例

当使用本机作为服务端的时候，往往需要指定终端，使用语句 –endpoint-url <http://127.0.0.1:port> 。

利用boto3API编写python程序，首先，使用命令 pip3 install -i [https://pypi.tuna.tsinghua.edu.cn/simple boto3 安装boto3](https://pypi.tuna.tsinghua.edu.cn/simple%20boto3%20安装boto3)用于python import。然后利用boto3的API操作s3。如下图7所示。仅展示上传文件操作的示例。注：由于使用的是本地服务，所以获取s3 resource的时候，应该加上参数endpoint=”<http://127.0.0.1:9000>” 。



图7 boto3 官方api文档 文件上传

图8为编写的文件上传函数，使用工具为jupyter。



图8 利用boto3 编写的上传文件函数

### 系统性能测试的两种方式——测试工具s3-benchmark和自行编写的测试程序（附带图形界面）

测试工具选用s3-benchmark，基于golang环境。

s3-benchmark的使用如下图9所示。图中矩形红框标注的分别是参数语法和输出数据。

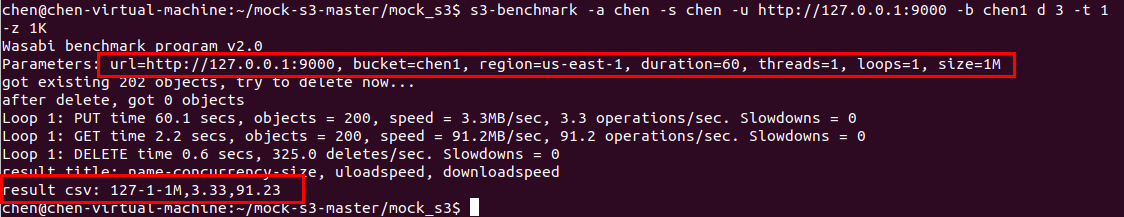


图9测试工具 测试简图

对于测试工具分析性能，我们可以使用脚本，将输出定向到文件，然后再对文件数据进行分析，这中方式老师在课堂已经教给大家了。

下面将着重介绍使用boto3 API编写的测试程序。

整个测试程序包含一个基文件（froot.py）和一个测试用文件（obs\_test\_x\_y.py），其中，obs\_test\_x\_y.py调用froot.py。

froot.py运行的命令行及参数是

python3 froot.py bucket clients filenum filesize

解释：

bucket：s3中的bucket名称

clients：客户端个数

filenum：测试存储对象的个数

filesize：存储对象大小

测试程序使用命令行运行，运行规则如下：

python3 obs\_test\_x\_y.py bucket clients filenum filesize\_range x\_type y\_type

解释：

bucket：s3中 bucket的名称

clients：客户端个数

filenum：the number of object\_file to be tested

x\_type：绘图中，x轴表示的类型，x\_type=0，则x轴表示filesize；x\_type=2，则x轴表示客户端数量

y\_type：绘图中，y轴表示的类型，y\_type=1，则y轴表示传输速率speed，y\_type=3，则y轴表示%点的延迟

filesize\_range（整数）：当x\_type=0时，filesize\_range（整数）的单位是8M，否则 filesize的单位是比特B。

说明：当x\_type=0 时，从1到filesize\_range的循环中，都将调用froot.py，并且froot.py传进的参数中，filesize是变量；当x\_type=2时，从1到clients的循环中，都将调用froot.py，并且froot.py传入的参数中，clients时变量。

每次froot.py运行的结果都输出到.csv文件中，等obs\_test\_x\_y.py的循环结束后，再对.csv文件处理，绘出图形。

代码请见报告同一文件夹内的.py文件。

## 六 结果分析

1 服务端mock-s3运行截图如图10所示。

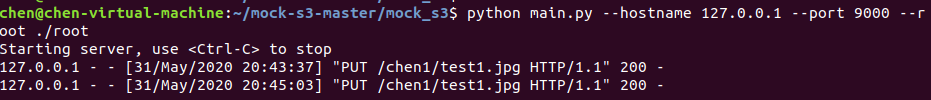


图10 mock-s3服务端运行截图

由图可知，mock-s3服务端正常运行。

2 客户端aws-shell运行截图如图11所示

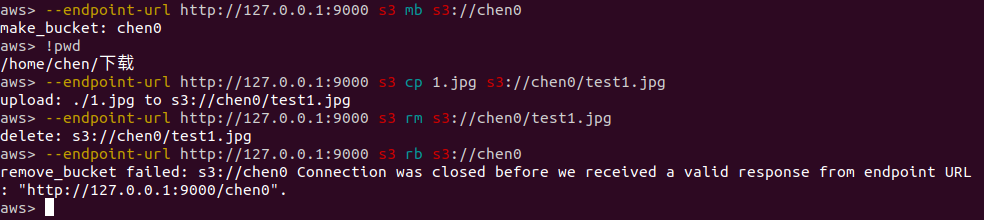


图11 aws-shell运行截图

一共测试了四条PUT/GET，分别为新建bucket、存放对象、删除对象、删除bucket，根据输出可知，aws-shell客户端工作正常。

3 客户端python程序（命令行方式执行，只给出了存放对象的python程序运行），运行截图见图12.

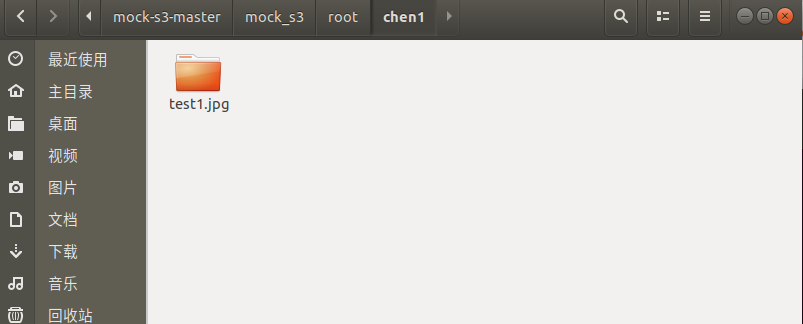
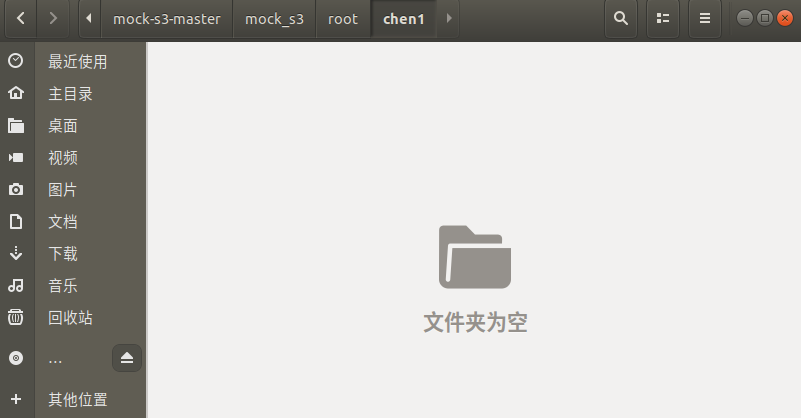


图12 tt.py客户端存放对象程序运行截图

tt.py的运行参数有bucket、filepath、key。使用者只需给出这三个参数，执行命令行即可存放指定对象。根据图示，该程序正常运行。

4 测试工具s3-benchmark对存储系统性能分析（简要展示），运行如图13所示。

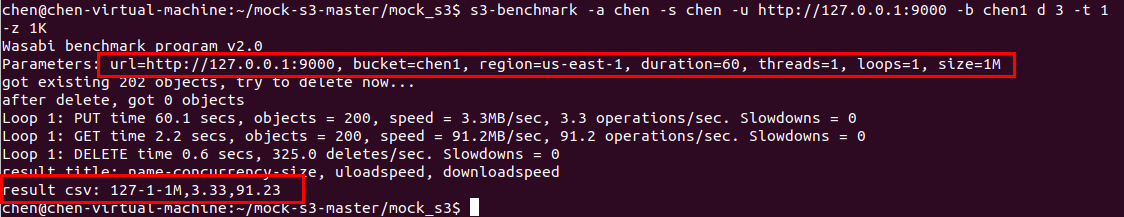


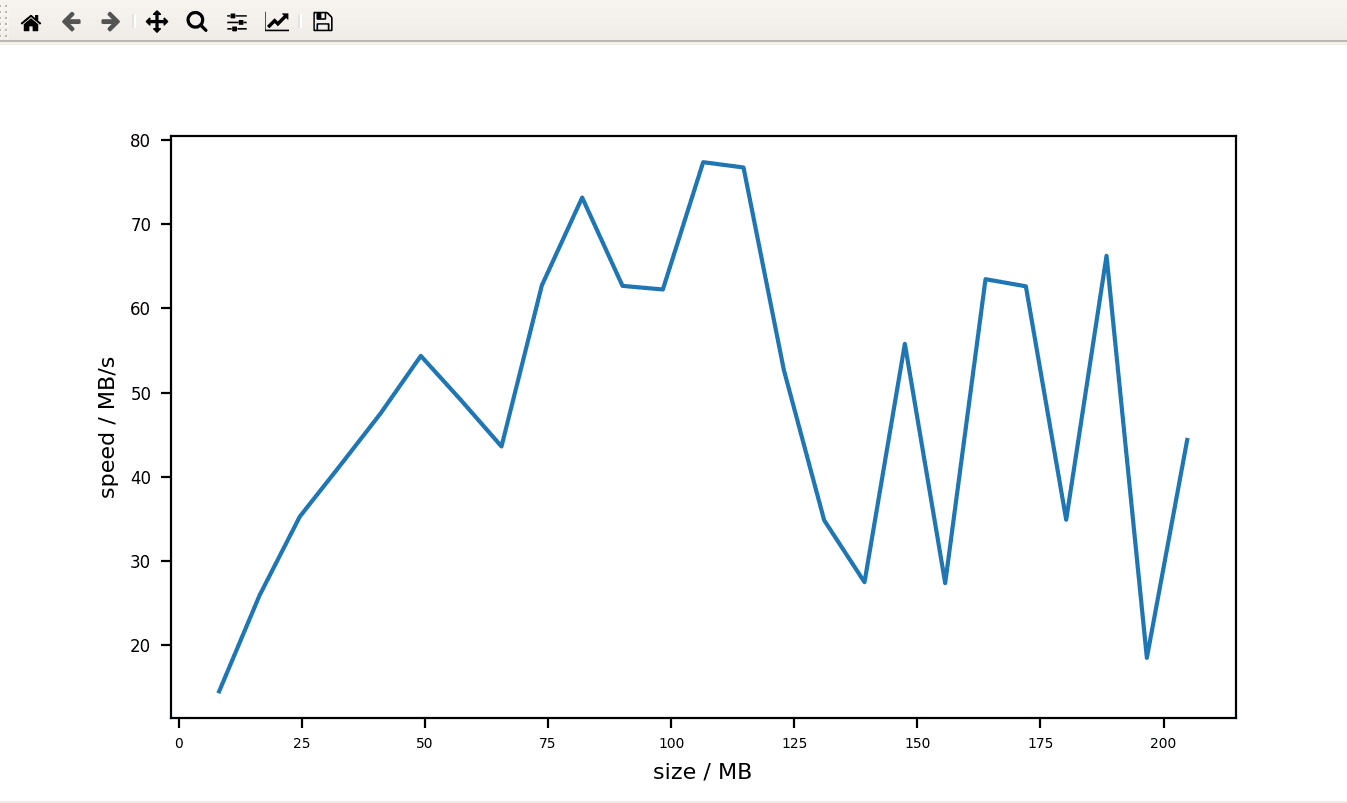
图 13 s3-benchmark测试工具运行截图

运行参数解读图上已给出，需要着重做的是将不同参数下运行的结果重定向输出到文件，然后对文件的数据进行分析。

5 系统性能测试 基于boto3 API 的python程序

该测试程序的主要原理在上一节已经进行说明，下面来看运行结果并分析。

1. 对象大小对传输速率的影响，见图14。对象大小以8MB的倍数递增到200MB。共25个点对。客户端数量=2。



根据改图可以看出，当文件大小在110MB左右，传输速率达到顶峰，当然由于实验条件有限（VMWare-Ubuntu性能限制等原因），200MB以后的数据我们无法考虑。在110MB以后，传输速率呈波动式下降。分析波动的原因——文件过大，且python程序不成熟。

1. 客户端数量对传输速率的影响，见图15。该实验做了多次，取了其中三次图片输出。

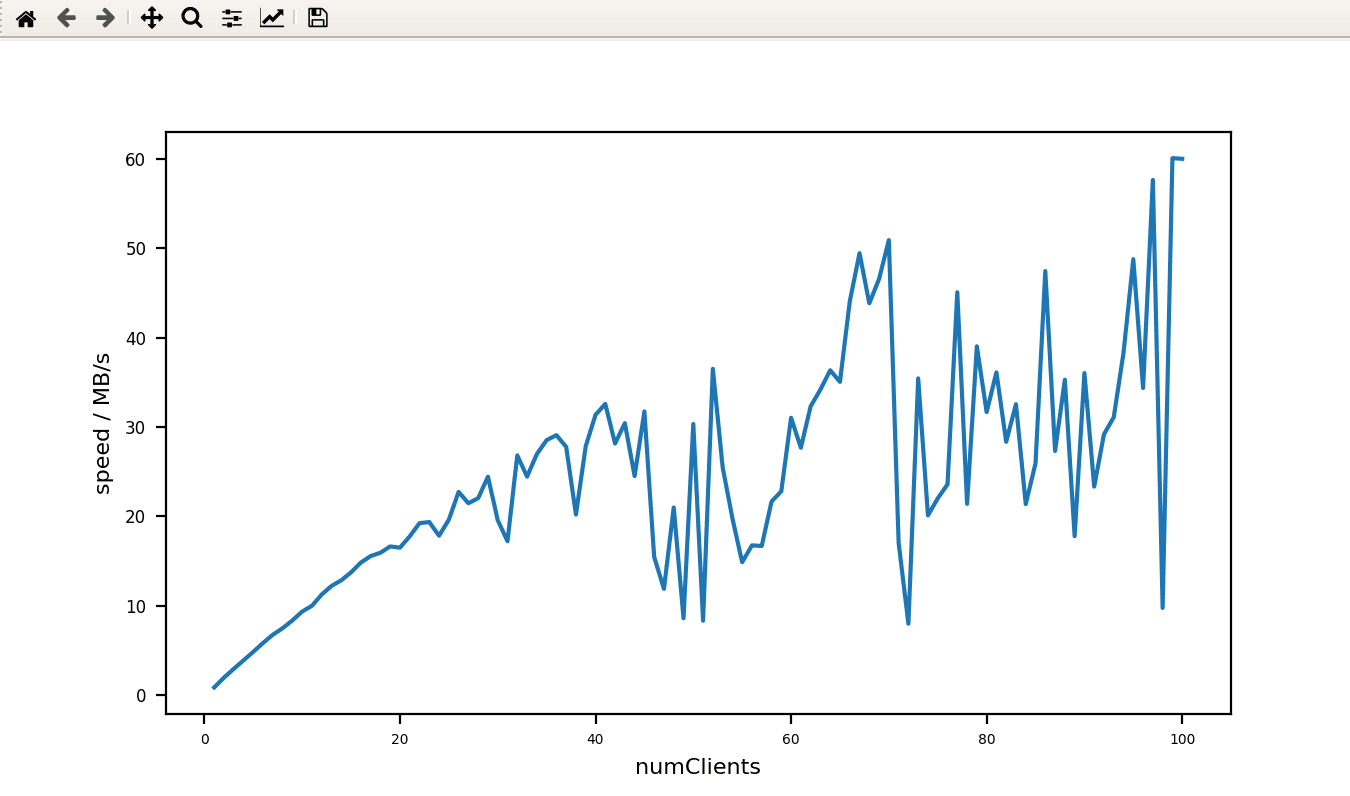
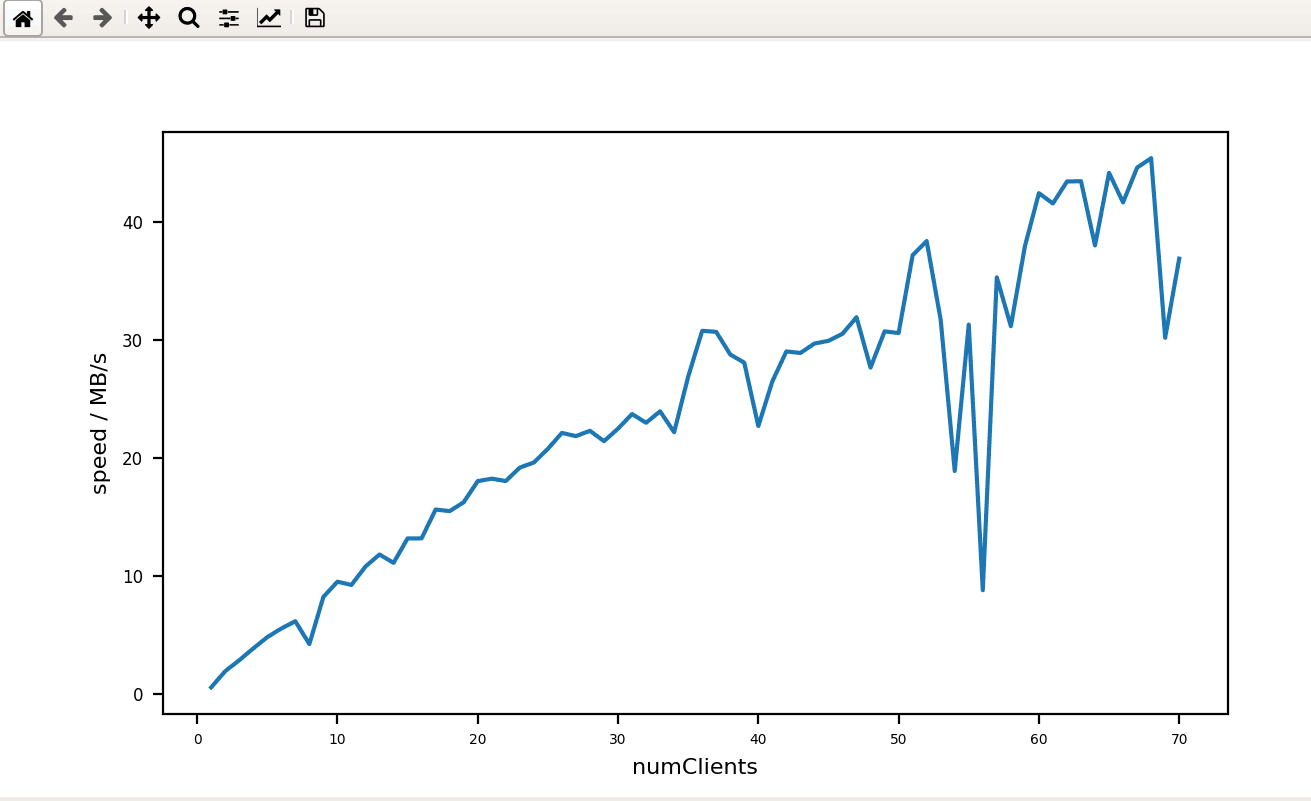
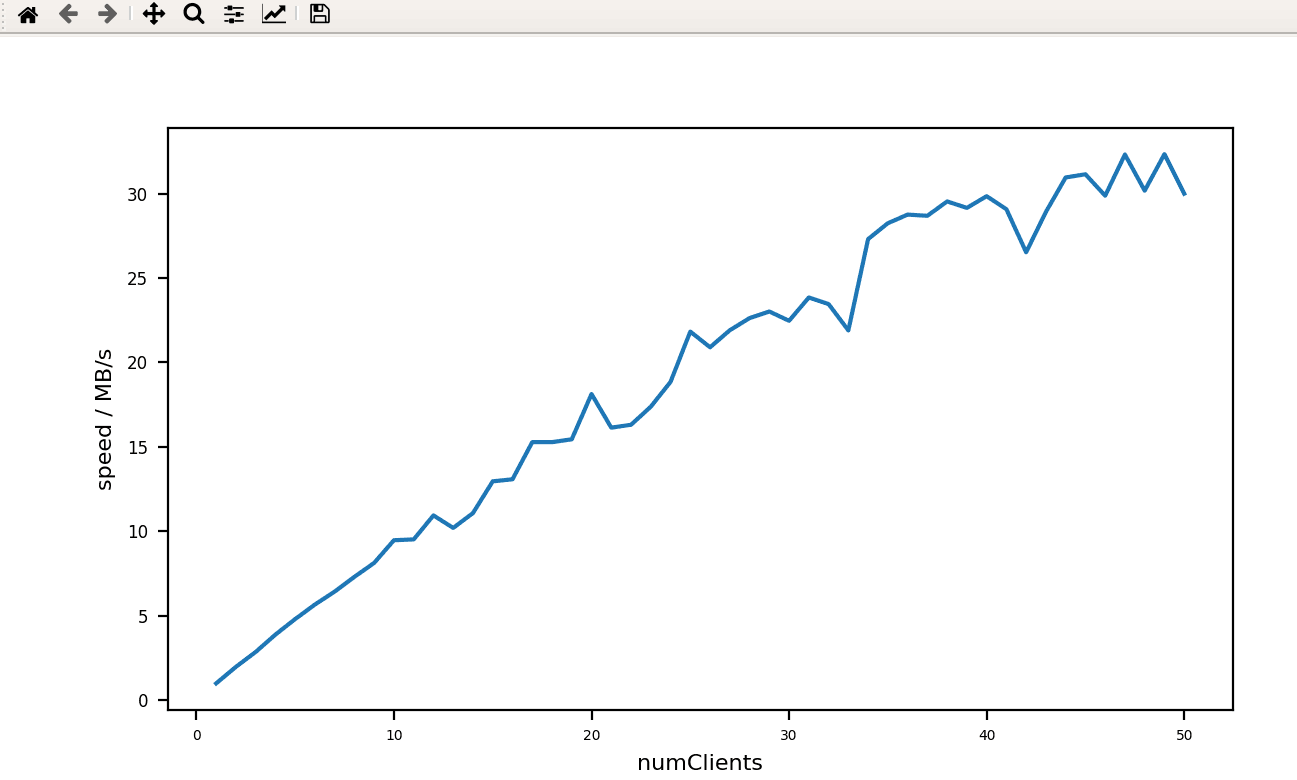


图15 客户端数量对传输速率的影响

这三次图片输出的客户端数量上限分别为50、70、100。可以看出，客户端数=50时，传输速率增长开始变慢；客户端数=70时，传输速率增长变慢，且波动性增大；客户端数=100时，已经呈现水平值上线波动的特性。可知，当客户端数=100时，传输速率的增长放慢甚至不再增长。从理论分析，客户端数再继续增大时，传输速率会减小。

1. 客户端数量对99th%ile延迟的影响，见图16。

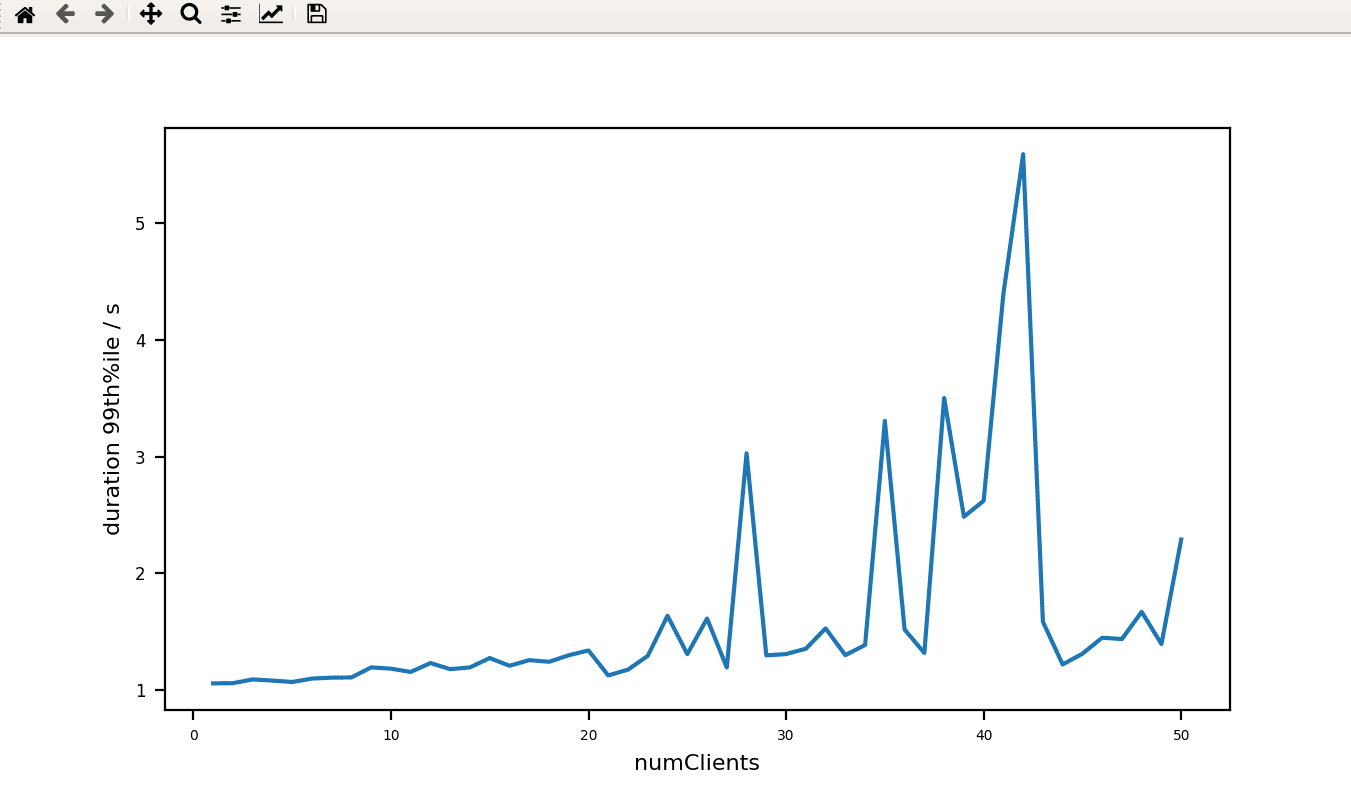


图16 客户端数量对99th%ile Duration 的影响

从本图可以看出，客户端越多，基本上呈现延迟越大的规律，且延迟向上波动性增长。从理论分析也不难得出这个结论。客户数量越多，服务端需要同时处理的请求越多，越难调度，越容易产生更大延迟。

## 七 实验总结

通过本实验，让我系统的了解了对象存储在服务端、客户端和应用端的具体存在形式和作用，同时对对象存储的系统构建有了大致的掌握，拥有了一定的应用部署能力。

本实验最有趣的地方是，自己通过boto3的API写python测试程序，然后将自己得出的各种量跟专业的测试工具进行对比，找不足找bug的过程。这一过程让我对传输速率、延迟、百分点延迟和并发等在对象存储中特点有了更深的认识，自己编写的python程序，十分的不成熟，需要多次分析专业的测试工具寻找规律寻找突破点，让人在旧头疼中发现新头疼，不停dubug，不停更新认知。