

**2016** 级

《物联网数据存储与管理》课程

**实 验 报 告**

**姓 名 肖 明**

**学 号 U201514872**

**班 号 物联网1601班**

**日 期 2019.05.28**

目录

[一、实验目的 2](#_Toc512280440)

[二、实验背景 2](#_Toc512280441)

[三、实验环境 2](#_Toc512280442)

[四、实验内容 4](#_Toc512280443)

[五、实验过程 4](#_Toc512280444)

[Minio： 4](#_Toc512280445)

[Mock-s3-master 7](#_Toc512280446)

[六、实验分析 8](#_Toc512280447)

[七、实验总结 10](#_Toc512280448)

[参考文献 10](#_Toc512280449)

# 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，进行初步测试；

3. 基于对象存储系统，架设实际应用，示范主要功能。

# 二、实验背景

现如今传统的文件系统和数据库Web后端不再足以满足要求，只好为管理非结构化数据的存储系统让路。不像文件系统中的文件，对象存储在扁平结构中。只有对象池：没有文件夹，没有目录，也没有层次体系。你只要提供对象ID，就可以请求某个对象。对象可以是本地的，也可以是远在千里之外的云服务器上，但由于它们是在扁平的地址空间，检索方式一模一样。一个重要的方面是元数据处理。对象存储提供了极高的灵活性，因为对象元数据是任意的。

元数据并不仅限于存储系统认为很重要的对象（想一想文件系统中的固定元数据）。你可以手动添加任何类型或任何数量的元数据。比如说，你可以指定与对象关联起来的应用程序的类型，指定应用程序的重要性，指定赋予对象的数据保护级别，指定是不是想把该对象复制到另一个站点或多个站点，指定何时删除该对象，不一而足。

鉴于互联网上的大部分数据是非结构化数据，加上专家们预测非结构化数据以两位数的幅度增长，因而迎面克服这个挑战很重要。非结构化数据必须以易于访问的方式来存储，兴起的对象存储技术综合了原网络存储技术的高速直接访问和数据共享等优势，同时也提供了具有高性能、高可靠性、跨平台以及安全的数据共享的存储体系结构。

# 三、实验环境

操作系统规格如图1：



图1 实验操作系统信息

Java版本如图2：



图2 java版本

Python版本为3.6如图3：

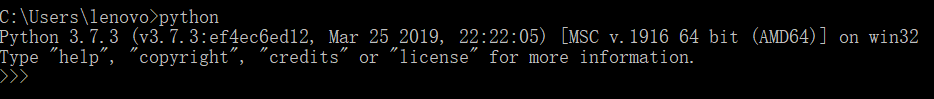


图3 Python版本

# 四、实验内容

（1）安装Java环境，下载Minio作为服务端，打开服务器端口，使用客户端对服务器进行访问，使用COSBench进行测试。

（2）安装python环境，下载Mock-S3作为服务端，使用客户端对服务器进行访问，使用COSBench进行测试。

# 五、实验过程

## Minio：

1. 安装minio客户端及服务器（Windows）。
2. 在minio目录下用命令行中打开minio客户端和服务器，在命令行中键入：minio.exe server F:\DATA。无需在F盘中新建DATA文件夹，会自动生成相应文件夹。如图4所示，我的F盘有26G空闲空间，共103G空间。用户名和密码已经修改过，修改步骤如下一步。



图4 server

1. 在第一使用上行命令时，会得到默认的用户名及密码，在网页中输入127.0.0.1:9000登陆后，便可修改密码，如图5。同时可以进行数据上传和下载。

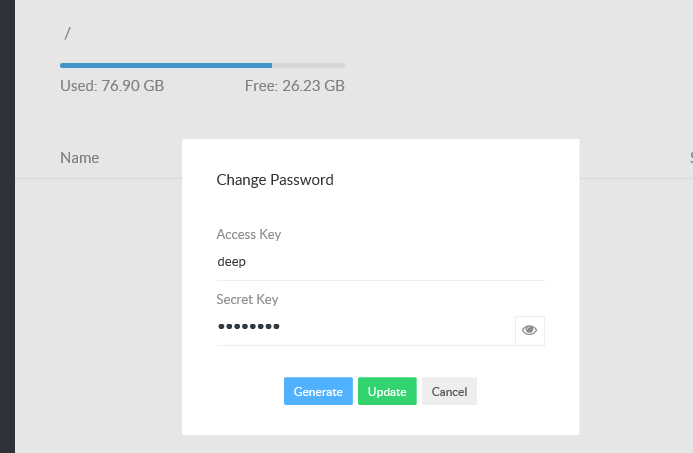


图5 修改账户及密码

1. 下载COSBench，版本0.4.2.c4，在COSBench目录下运行start-all.bat。如图6，图7。

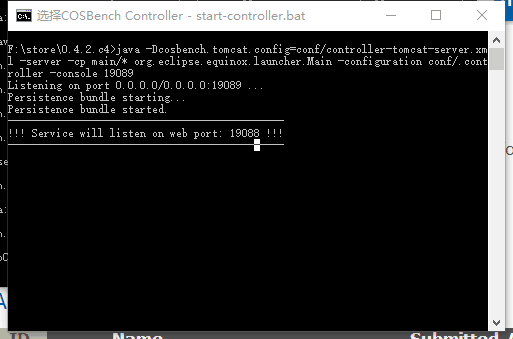


图6 start-all.bat(1)

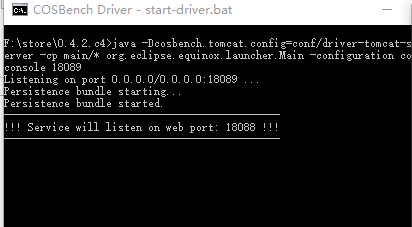


图7 start-all.bat (2)

（5）打开网址http://127.0.0.1:19088/controller/，进入cosbench的controller。如图8，点击红线标识部分进入下一界面，如图9，点击选择文件，选择obs-tutorial-master中的workload-example.xml文件。点击submit开始评测，如图10。

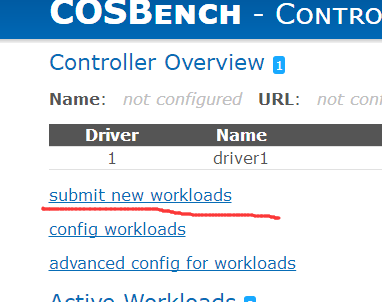


图8 进入（1）



图9 进入（2）

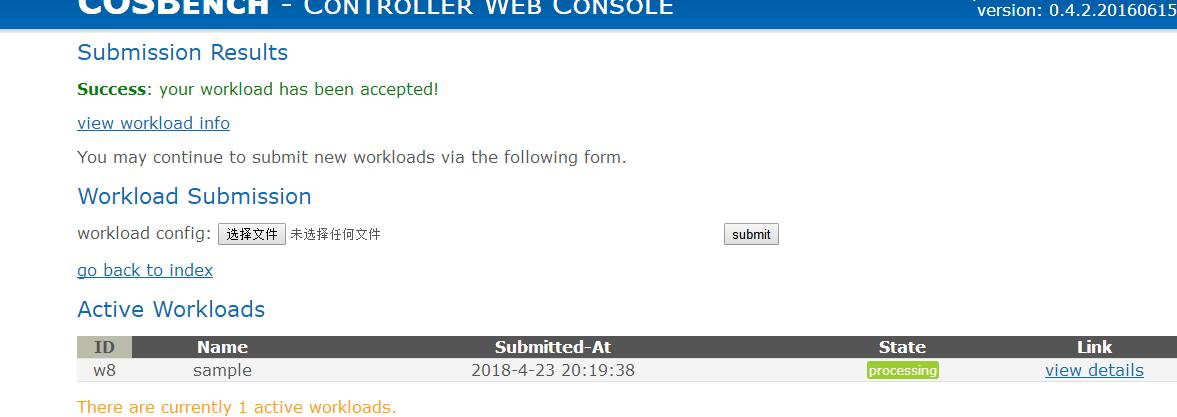


图10 开始

1. 最终点击view details观查成功情况。实验结果如图11，没有fail的情况。

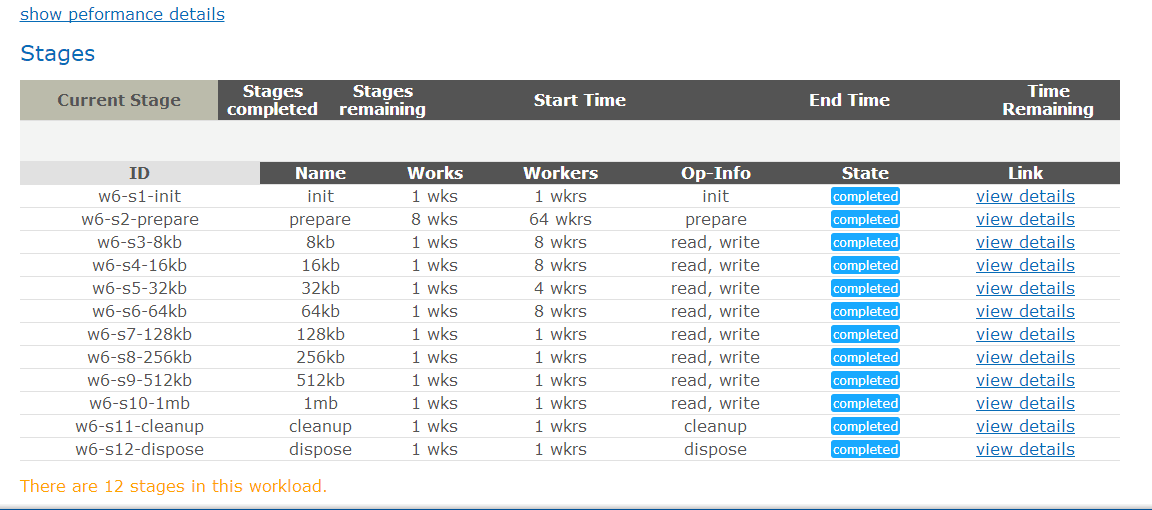


图11 minio结果

## Mock-s3-master

1. 下载mock-s3-master文件夹。
2. 在该目录下进入cmd，键入python setup.py install执行初始化,键入cd mock\_s3进入mock\_s3文件夹，键入python main.py --host 127.0.0.1 --port 9000 --root ./root。
3. 打开s3服务器，再打开start-all/bat，剩余步骤如minio步骤，实验结果如图12。全部成功，没有fail的情况。

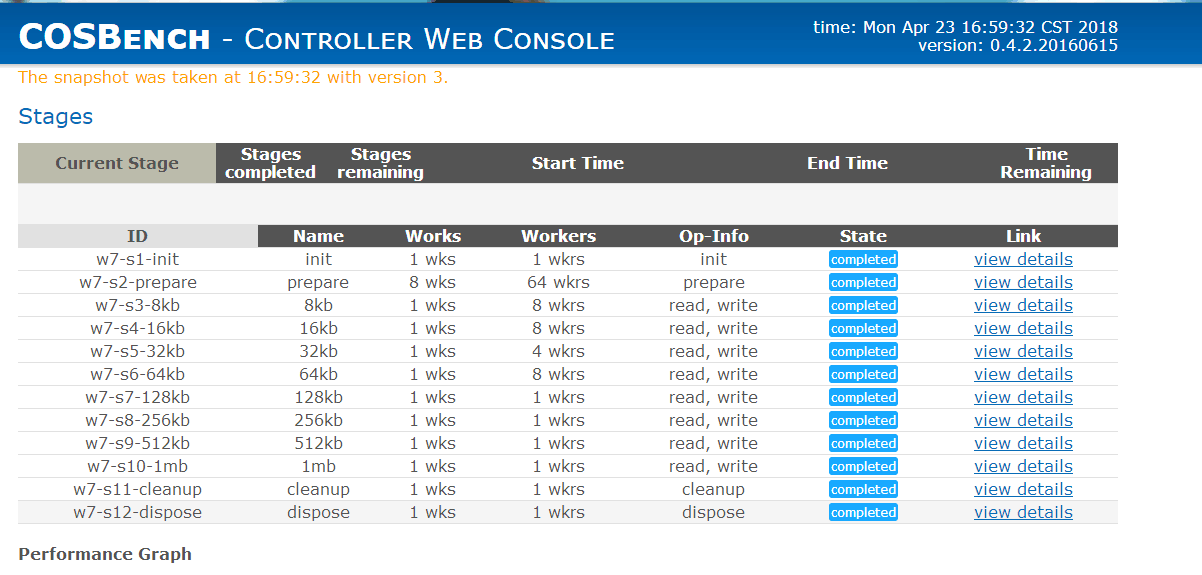


图12 mock-s3结果

# 六、实验分析

（1）在采用mimio作为服务端时，发现写的时候成功率全为100%，而读的时候有时不为100%。如图13，图14。不为100%的情况存在于size较小的情况下，随着size的增大读写成功率均为100%。

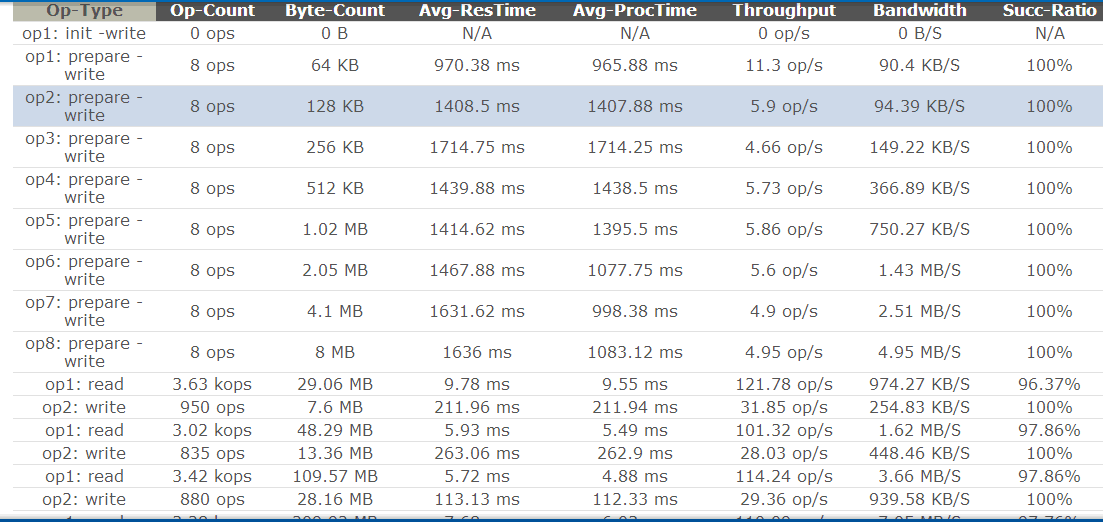


图13 minio时读写（1）

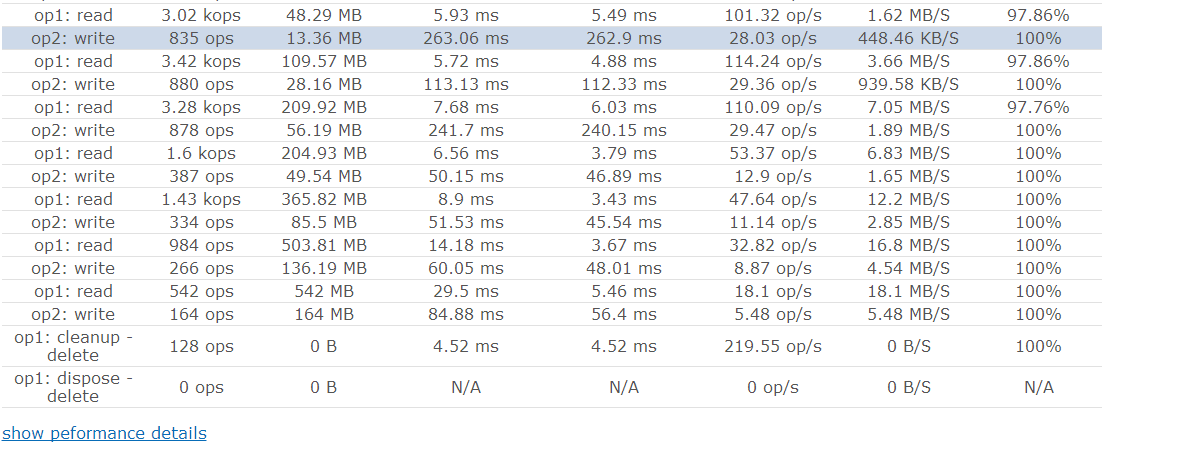


图14 minio时读写（2）

（2）mock\_s3与minio相比，读写速度明显更快，同样存在着size较小时读取正确率不为100%的情况，但不影响最后结果。如图15。

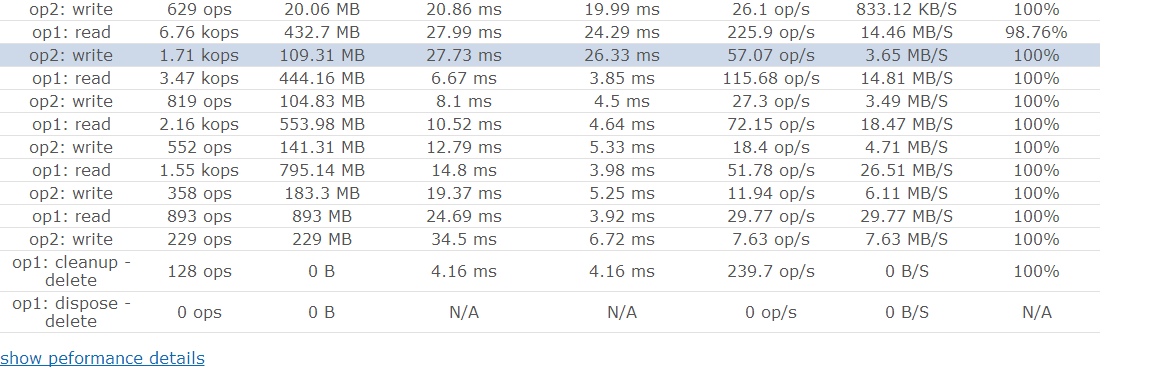


图15 mock\_s3读写

1. 实验中遇到的问题：
2. 在网页选择文件前，将workload-example.xml用记事本打开，修改其中的用户名及密码，不然跑COSBench会出错。
3. 在sock\_s3实验中，在键入python main.py --host 127.0.0.1 --port 9000 --root ./root时遇到执行错误，在命令行输入set HOME=%USERPROFILE%配置路径后解决了该问题。
4. 在minio实验时打开start-all.bat文件时报错，提示Java中bin文件夹中server中缺少jvm.dll文件，但我的那个目录下连server都没有，后来在bin文件夹中创建了server目录并将jvm.dll文件复制过去就好了。

# 七、实验总结

通过这次实验，学习了如何搭建minio和sock\_s3服务器，并学会了如何通过网页客户端进行文件上传和下载，并通过COSBench来测试其读写效率。学习了一些GitHub的知识，为之后的工作和学习打下了一些基础。在实验的过程中，我没有遇到使用Mac系统和Windows7系统的同学们遇到的问题，比如需要通过修改works的值来消除fail的情况，这说明不同的系统在执行同一网络存储请求时可能会得到不一样的结果，我理解到网络存储相关的技术人员在程序设计时也要考虑普适性的问题。通过这次实验我稍微了解了对象存储的相关知识，让我对客户端，服务端之间的交互有了更直接的体会，总的来说这次实验还是有收获的，希望通过不断地学习增长知识，对计算机技术有更深的理解。

# 参考文献

[1] ARNOLD J. OpenStack Swift[M]. O’Reilly Media, 2014.

[2] ZHENG Q, CHEN H, WANG Y等. COSBench: A Benchmark Tool for Cloud Object Storage Services[C]//2012 IEEE Fifth International Conference on Cloud Computing. 2012: 998–999.

[3] WEIL S A, BRANDT S A, MILLER E L等. Ceph: A Scalable, High-performance Distributed File System[C]//Proceedings of the 7th Symposium on Operating Systems Design and Implementation. Berkeley, CA, USA: USENIX Association, 2006: 307–320.

[4] GitHub

[5] CSDN

[6] StackOverflow