

**2016** 级

《物联网数据存储与管理》课程

**实 验 报 告**

**姓 名 胡晋源**

**学 号 U201614901**

**班 号 物联网1601班**

**日 期 2019.05.27**

**目 录**

[一、实验目的 1](#_Toc512289556)

[二、实验背景 1](#_Toc512289557)

[三、实验环境 1](#_Toc512289558)

[四、实验内容 2](#_Toc512289559)

[4.1 对象存储技术实践 3](#_Toc512289560)

[4.2 对象存储性能分析 3](#_Toc512289561)

[五、实验过程 4](#_Toc512289562)

[六、实验总结 10](#_Toc512289563)

[参考文献 10](#_Toc512289564)

# 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

对象存储，也叫做基于对象的存储，是用来描述解决和处理离散单元的方法的通用术语，这些离散单元被称作为对象。就像文件一样，对象包含数据，但是和文件不同的是，对象在一个层结构中不会再有层级结构。每个对象都在一个被称作存储池的扁平地址空间的同一级别里，一个对象不会属于另一个对象的下一级。文件和对象都有与它们所包含的数据相关的元数据，但是对象是以扩展元数据为特征的。每个对象都被分配一个唯一的标识符，允许一个服务器或者最终用户来检索对象，而不必知道数据的物理地址。这种方法对于在云计算环境中自动化和简化数据存储有帮助。

Minio Cloud Storage 是一款轻量级对的象存储服务，兼容Amazon S3。只需要简单的命令，就可以搭建服务器，能实现可以通过浏览器访问的简易网盘功能。利用Minio客户端MC可以在命令行中对云中的内容进行管理。

COSBench是衡量云对象存储服务性能的基准测试工具。它包含两个关键元素：Driver和Controller，前者产生负载，向目标对象存储系统发布操作，收集统计数据，后者协调driver，处理实时状态，接收提交的负载。

# 三、实验环境

硬件环境：

1. CPU：I7 7500U
2. 内存：8G
3. 硬盘：256SSD+500G机械

软件环境：

1. 操作系统：Ubuntu16.04
2. Java版本：1.8.0\_171
3. Python版本：python 3.5.2
4. 对象存储服务端：minio
5. 对象存储客户端：minio client
6. 对象存储测试端：COSbench、s3bench

# 实验内容

本次实验在linux环境下搭建相应平台实现对象存储，服务器端为minio，客户端为minio client，分别用cosbench和s3bench这两种不同的评测工具进行实验。

本次实验的主要内容如下：

1. 对git进行了解和熟悉，知道如何上传本地文件到github上
2. 配置python、java、go语言环境
3. 对对象存储系统进行实践，采用目前几种典型的服务，服务端的minio，s客户端的minio client，并进行简单的创建或删除bucket、上传或删除文件等操作。
4. 采用COSBench与s3bench对对象存储系统进行测试。
5. 通过修改workers、读写比radio以及数据大小对负载测试结果进行一定的性能分析。

## 4.1 对象存储技术实践

1. 配置minio server端，通过浏览器登陆127.0.0.1查看效果，进行简单的创建bucket，上传文件，删除文件和bucket操作。
2. 配置minio客户端mc，在命令行下输入命令实现创建bucket，上传文件，删除文件和bucket操作。
3. 配置COSBench，提交负载样例到COSBench，观察运行和结束后的各种状态指标。
4. 配置s3bench，直接在终端下通过修改run-s3bench.sh文件来对负载样例进行测试并将想要得到的测试结果输出到txt文件中

## 4.2 对象存储性能分析

COSbench：

1. 测试读写比radio对运行结果的影响，分别将读写比设为6:4以及8:2，观察运行结果
2. 测试块大小对运行结果的影响。

将负载样例中的每项workers数统一改为4，块大小从8k到1M逐渐翻倍增大，观察运行结果。

1. 测试workers值对运行结果的影响

将负载样例的每项大小都改为64k，将workers值从1逐渐翻倍增大到128，观察运行结果。

S3bench：

直接运行修改好的run-s3bench.sh文件，在txt中观察结果

# 五、实验过程

1. 下载并打开minio server

进入终端后，用wget命令在minio官网中下载minio

在管理员权限下输入：./minio server /data开启minio服务端如图5.1所示

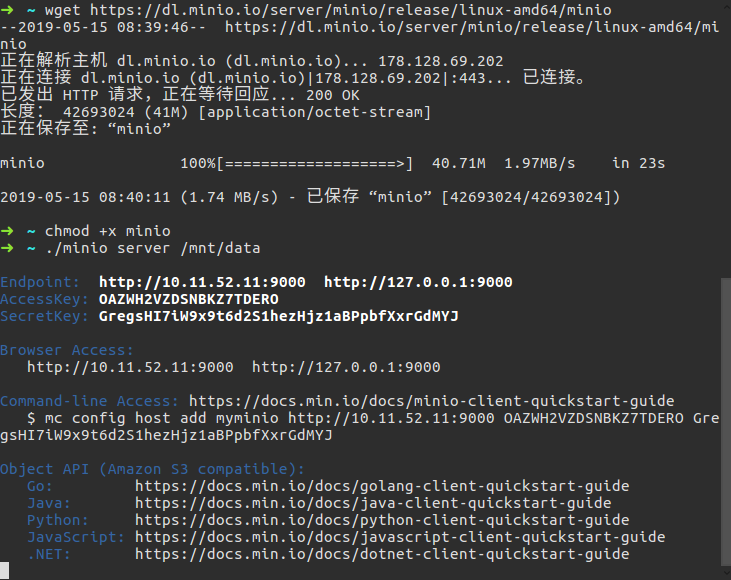


图5.1下载并打开minio server

此时在浏览器打开127.0.0.1:9000，也是给出的endpoint，在打开的minio browser，输入自己的AccessKey和SerectKey登陆。此时可以选择右下角红色标记随意添加bucket，在服务器端添加了名为test的bucket，还能上传文件，方便地实现类似网盘的功能，如图5.2所示。

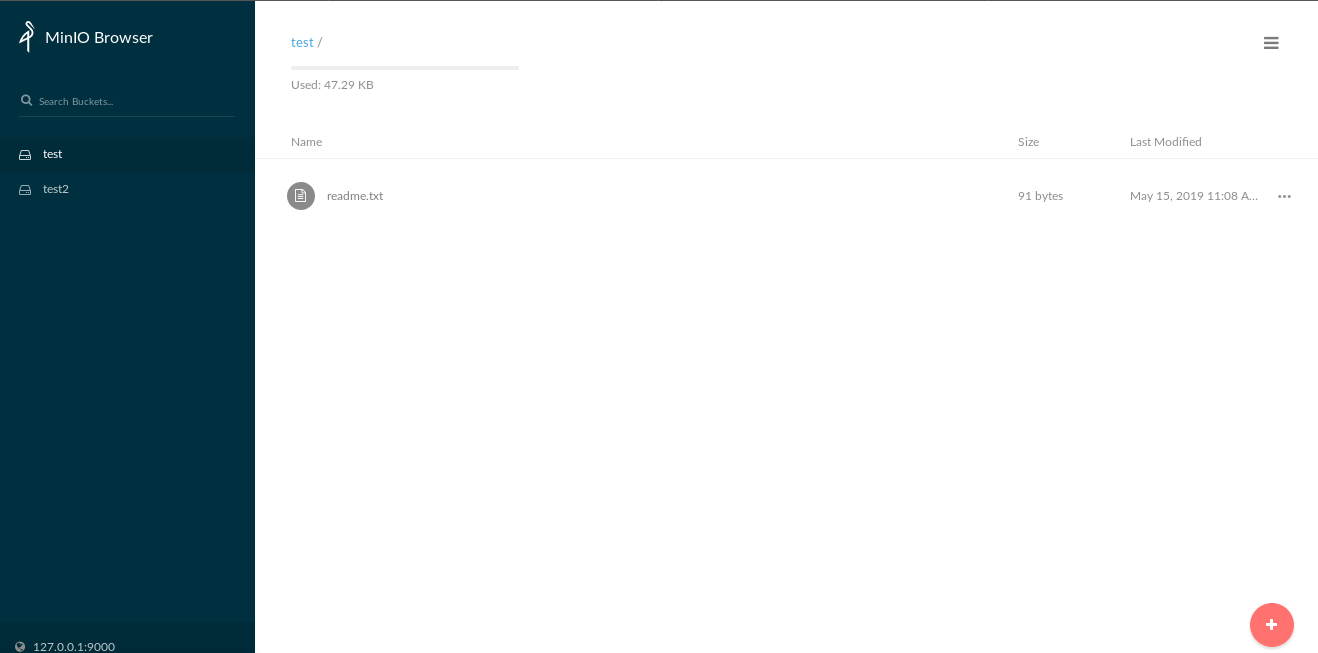


图5.2添加bucket和上传文件

1. 配置minio client

依旧是使用wget命令下载对象存储客户端minio client，然后打开客户端并添加bucket test2结果如图5.3所示。

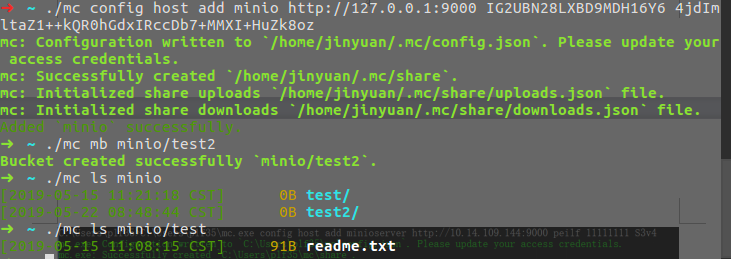
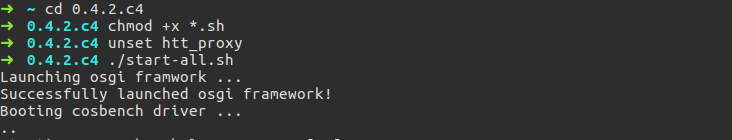


图5.3minio client操作存储系统

1. 下载并进入cosbench

同样是用wget命令下载cosbench，然后解压在终端中进入该目录用管理员权限修改权限，再输入./start-all.sh启动cosbench，如图5.4



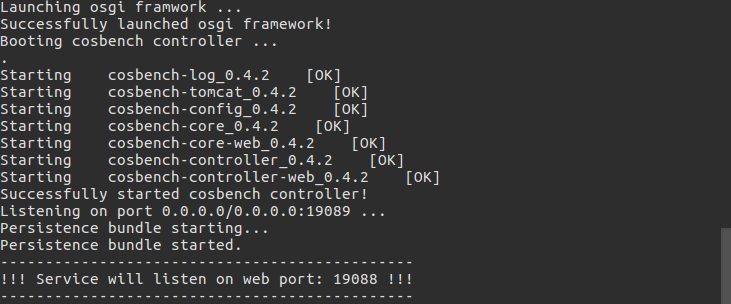


图5.4 start-all打开driver和controller

Driver的访问端口为18088，service的访问端口为19088.

双击web.bat打开浏览器中的cosbench，修改负载样例中的AccessKey和SerectKey并提交可访问cosbench界面，如图5.5

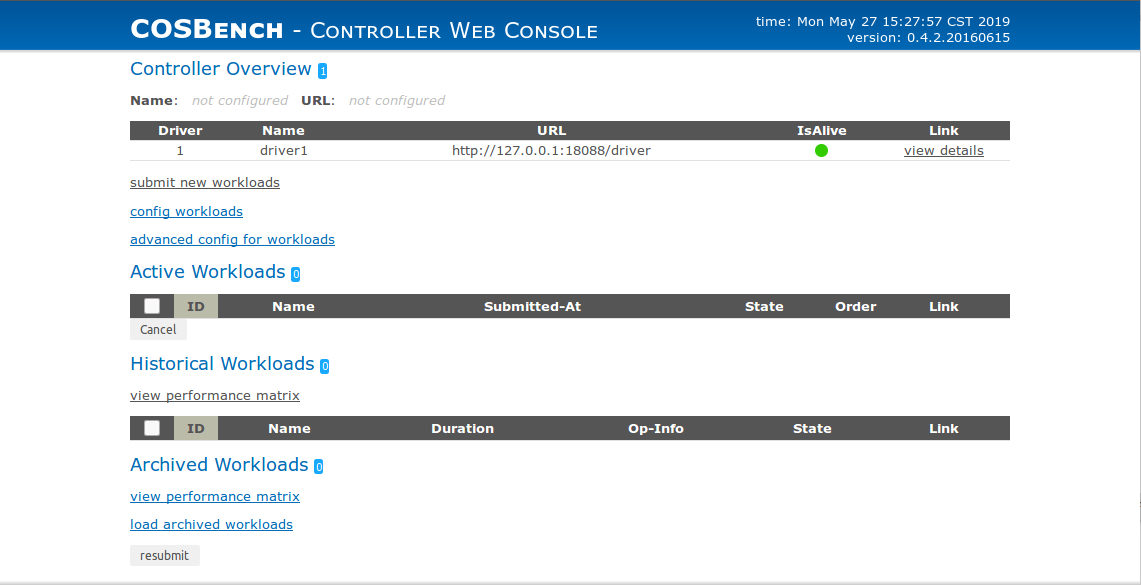


图5.5进入cosbench

1. 在cosbench中进行负载测试

点击submit new workloads添加负载进行测试，这里我们选择老师给出的原始测试用例workload-example.xml进行测试，不过在测试之前我们需要将accesskey和secretkey修改为服务端对应的内容（见图5.1），修改结果如图5.6

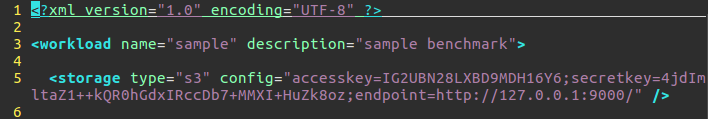
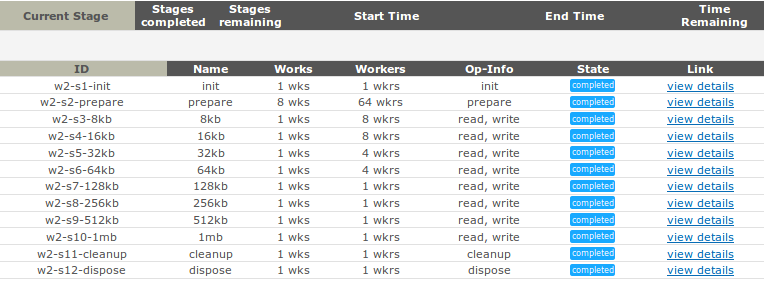


图5.6 修改测试用例秘钥

测试结果如图5.7



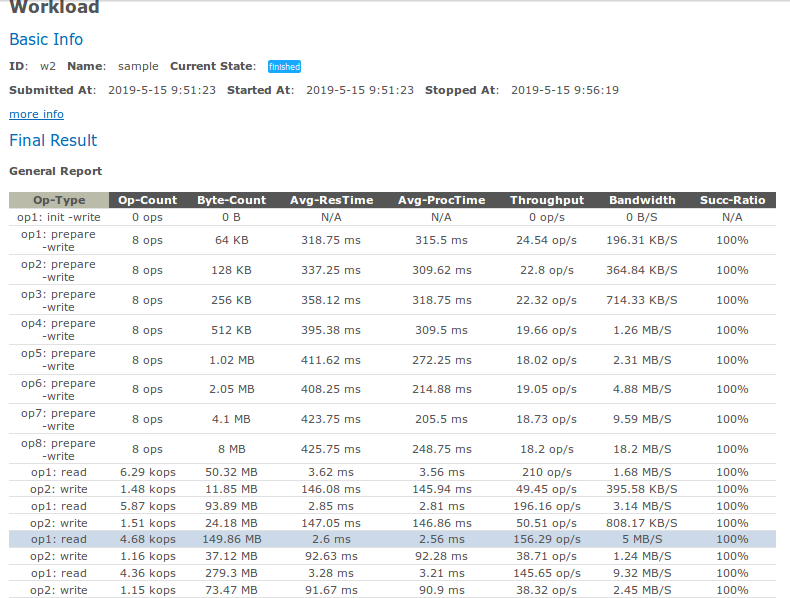


图5.7原始测试用例测试结果

在测试中，cosbench会以图的形式给出每一次测试的过程，如图5.8

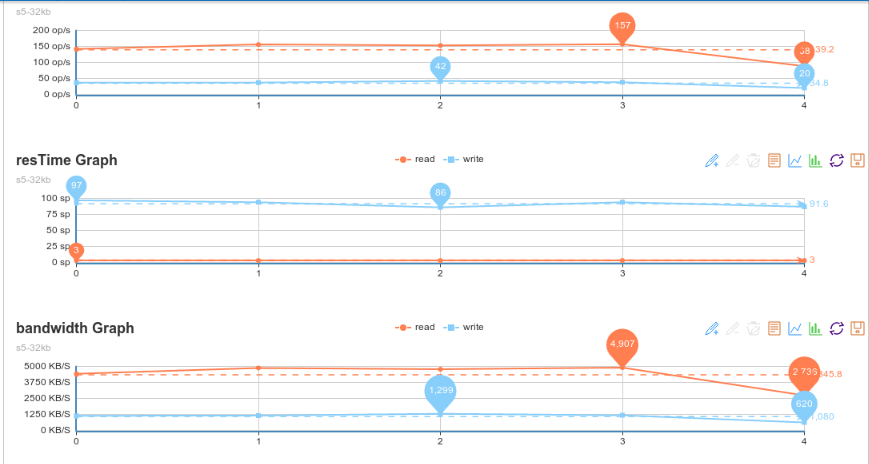


图5.8 cosbench运行过程

1. 读写性能对比。

利用原始测试用例结果图5.7先进行简单分析。写操作相对于读操作，平均处理时间要长得多，吞吐率和带宽都要小得多，但成功率较高。本次实验采用普通PC机作为服务器，而现有的磁盘和缓存的读性能都要远远强于写的性能，这符合我们的基本常识。

1. 测试块大小对性能的影响。

将每个workers值设为4进行统一，运行结果如图5.9所示。从上到下，块大小逐渐翻倍(8kb-1m）。发现读写的平均反应时间的整体趋势是随着块大小的增大而逐渐增大；带宽的趋势为，先缓慢增加，后增加得越来越快。吞吐量基本保持不变，吞吐率却始终保持100%，对此原因可能是块大小仍然不够大服务器性能较好。

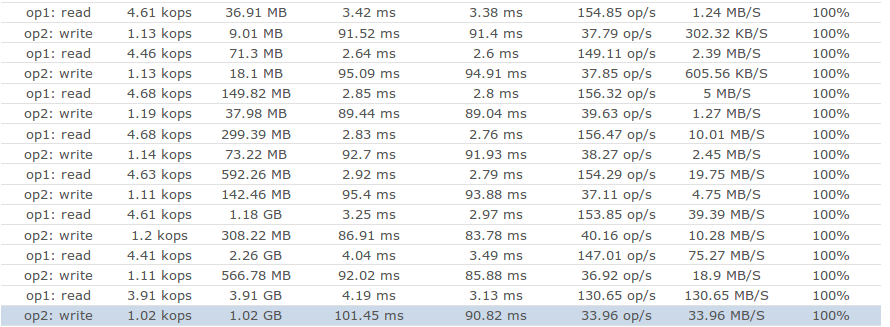


图5.9对比块大小影响

1. 测试workers值的影响。

块大小都为128k的情况下，wokers数从1翻倍到128。测试结果如图5.10所示发现吞吐量的趋势为先快速增加，然后达到峰值，之后开始减小，且减小得越来越快；读操作的带宽是先缓慢增大，后来增加得越来快；写操作的带宽先缓慢增加，后快速增加到峰值，后来出现减小的情况。

在该测试实例中，我们只将最大workers数控制在了128，,相对于服务器处理能力，这是一个不大的值，所以服务器处理能力和带宽相对充足，workers增加的情况下，读写性能快速增加，吞吐率始终保持100％。

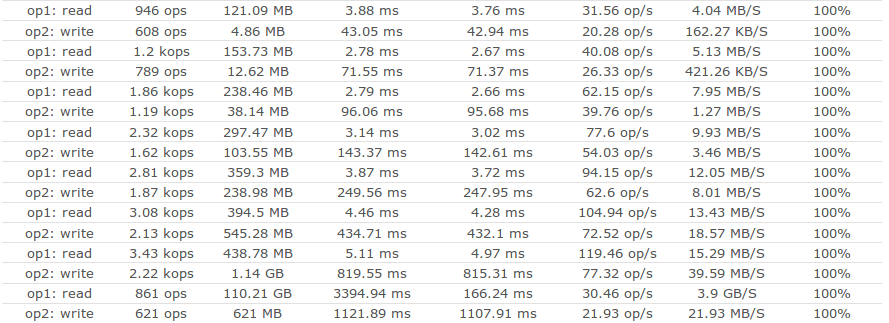


图5.10对比workers值影响

1. 用s3bench进行负载测试

# 六、实验总结

在本次实验中我自己搭建了简单的对象存储系统，并进行了测试。虽然服务器端用的是最简单清晰直观的minio，由于有可视化界面，所以使用起来非常方便，而且只有bucket和object的概念。客户端用的也是对应的最简单的minio-client但是在实验中仍然是遇到了一些问题。

比如我们在cosbench上submit一个测试文件时，要注意把accesskey和secretkey修改为服务端对应的内容，不然就无法进行测试。另外由于自己对linux的一些命令还不是特别熟悉，所以需要时常查询linux下的一些命令。另外在搭建go语言环境的时候，由于不知道为什么找不到go的安装路径了，所以废了好大的功夫才完成go语言环境的配置。最后通过对对象存储实例的评测，了解了数据块的大小、workers的数量、读写比ratio都会对系统的性能产生或大或小的影响。

通过这次实验，我了解了基本的对象存储的原理，懂得了对象存储设备只需有网就能将数据上传至云端，有效地解决了本地存储空间有限的问题，这对以后的海量数据存储提供了一种可以考虑的方案，也希望在以后的学习中进一步接触这方面的知识。另外还熟悉了git的一些操作，这对以后的学习也是大有裨益的。

# 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.