



2017 级

《物联网数据存储与管理》课程  
**实 验 报 告**

姓 名 段佳俊

学 号 U201714655

班 号 物联网 1701 班

日 期 2020.05.25

# 目 录

一、实验目的 .....	1
二、实验背景 .....	1
三、实验环境 .....	1
四、实验内容 .....	1
4.1 对象存储技术实践 .....	2
4.2 对象存储性能分析 .....	2
五、实验过程 .....	2
六、实验总结 .....	8
参考文献 .....	8

## 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

对象存储是用来描述解决和处理离散单元的方法的通用术语。对象在一个层结构中不会再有层级结构，是以扩展元数据为特征的。

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。

## 三、实验环境

操作系统	Windows 10 64 位
处理器	Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
内存	8GB
Java 版本	1.8.0_251
Server	Minio
Client	Minio Client
Test	COSBench

## 四、实验内容

本实验在 windows 环境下完成相应的环境配置，基础的实验环境安装完毕后部署对象存储的服务端、客户端和测试工具，然后启动 COSBench 测试程序，提交所给的测试文件，观察负载测试的结果，分析吞吐率、延迟等技术指标。

## 4.1 对象存储技术实践

- 1.在 windows 下配置 java 环境。
- 2.在官网上下载安装对象存储服务 minio 和客户端 minio client
3. 安装后使用浏览器访问 <http://127.0.0.1:9000>，如果可以访问说明安装成功，否则重新安装
- 4.下载测试程序 COSBench，配置完成后提交负载测试样例，观察运行结果。

## 4.2 对象存储性能分析

- 1.读写性能对比
- 2.分析块的大小对运行结果的影响  
将负载样例中 8kb~1mb work 的 workers 数量改为一致，分析块的大小对存储性能的影响。
- 3.分析 workers 数量对运行结果的影响  
将负载样例中块的大小改为一致，修改 workers 的数量，分析 workers 数量对存储性能的影响。

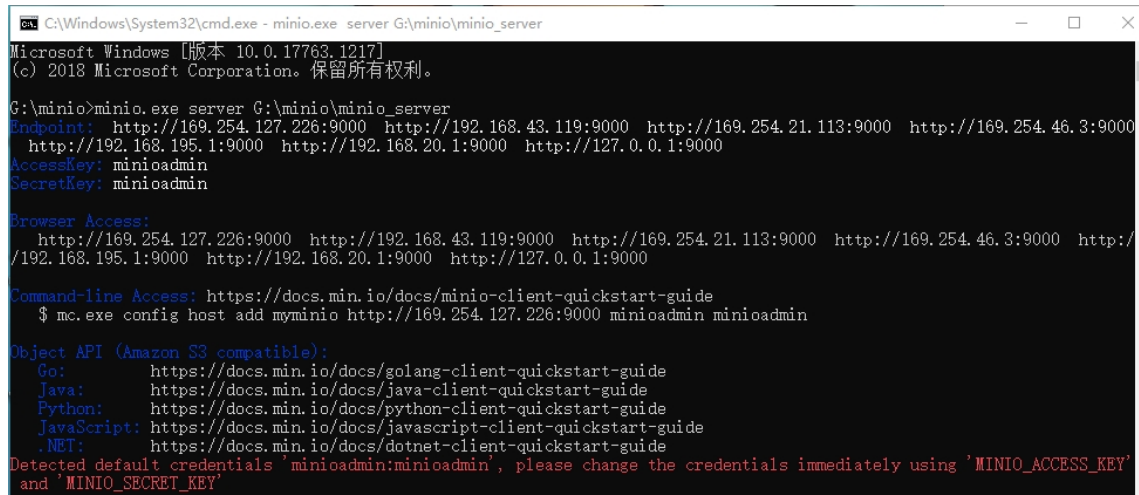
## 五、实验过程

- 1.下载配置 minio server 和 minio client

进入 minio 官网，下载 minio.exe 文件到在 G 盘新建的文件夹 minio，打开命令行，使用 cd 命令进入安装目录，键入：

```
G:\minio>minio.exe server G:\minio\minio_server
```

打开服务器，获取相应的 endpoint、accessKey 和 secretKey，如图所示。



```
C:\Windows\System32\cmd.exe - minio.exe server G:\minio\minio_server
Microsoft Windows [版本 10.0.17763.1217]
(c) 2018 Microsoft Corporation. 保留所有权利。

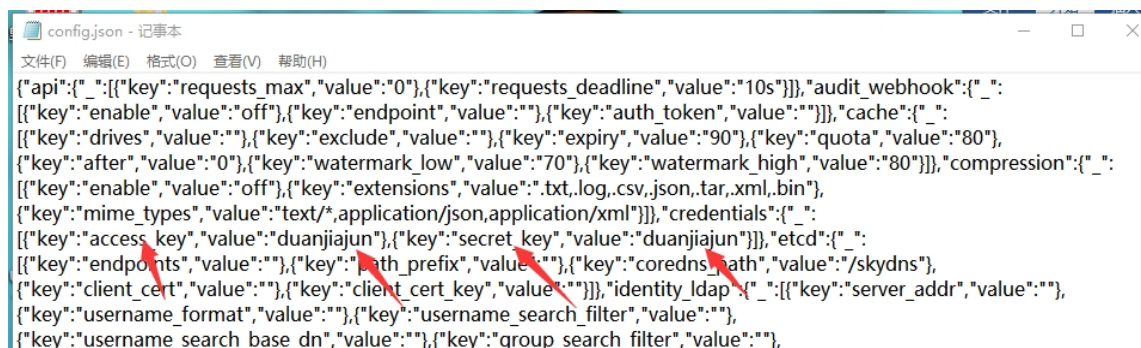
G:\minio>minio.exe server G:\minio\minio_server
Endpoint: http://169.254.127.226:9000 http://192.168.43.119:9000 http://169.254.21.113:9000 http://169.254.46.3:9000
          http://192.168.195.1:9000 http://192.168.20.1:9000 http://127.0.0.1:9000
AccessKey: minioadmin
SecretKey: minioadmin

Browser Access:
http://169.254.127.226:9000 http://192.168.43.119:9000 http://169.254.21.113:9000 http://169.254.46.3:9000 http://192.168.195.1:9000 http://192.168.20.1:9000 http://127.0.0.1:9000

Command-line Access: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc.exe config host add myminio http://169.254.127.226:9000 minioadmin minioadmin

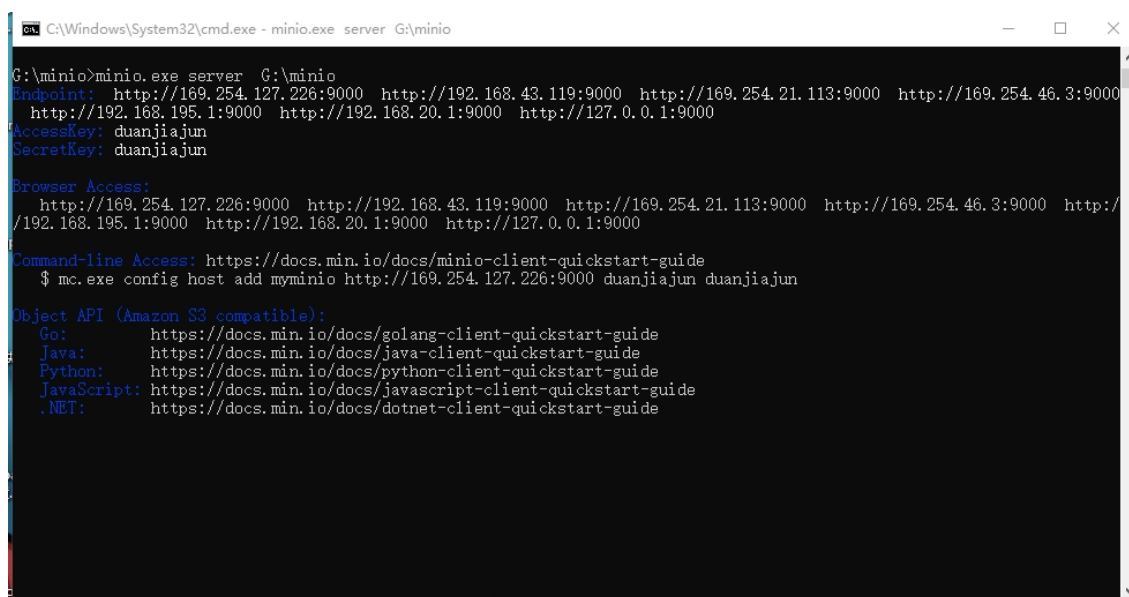
Object API (Amazon S3 compatible):
Go:      https://docs.min.io/docs/golang-client-quickstart-guide
Java:    https://docs.min.io/docs/java-client-quickstart-guide
Python:  https://docs.min.io/docs/python-client-quickstart-guide
JavaScript: https://docs.min.io/docs/javascript-client-quickstart-guide
.NET:    https://docs.min.io/docs/dotnet-client-quickstart-guide
Detected default credentials 'minioadmin:minioadmin', please change the credentials immediately using 'MINIO_ACCESS_KEY' and 'MINIO_SECRET_KEY'
```

其中 accessKey 和 secretKey 显示为初始值，均为 minioadmin，打开 minio.exe 所在文件夹，按照路径 minio\_server\minio.sys\config 找到文件 config.json，在文件中修改 accessKey 和 secretKey 的值为自定义的值，如图所示。



```
{
  "api": {
    "requests_max": 0,
    "requests_deadline": "10s",
    "audit_webhook": {
      "enable": "off",
      "endpoint": "",
      "auth_token": ""
    },
    "cache": {
      "drives": "",
      "exclude": "",
      "expiry": "90",
      "quota": "80",
      "after": "0",
      "watermark_low": "70",
      "watermark_high": "80"
    },
    "compression": {
      "enable": "off",
      "extensions": ".txt,.log,.csv,.json,.tar,.xml,.bin",
      "mime_types": "text/*,application/json,application/xml"
    },
    "credentials": {
      "access_key": "duanjiajun",
      "secret_key": "duanjiajun"
    },
    "etcd": {
      "endpoints": "",
      "path_prefix": "",
      "coredns_path": "/skydns",
      "client_cert": "",
      "client_cert_key": ""
    },
    "identity_ldap": {
      "username_format": "",
      "username_search_filter": "",
      "username_search_base_dn": "",
      "group_search_filter": ""
    }
  }
}
```

修改完成后，保存文件并退出，关闭先前的终端，接着重新打开输入上述指令，结果如图所示，可以发现终端中的 AccessKey 和 SecretKey 已经更新为用户自定义的值。



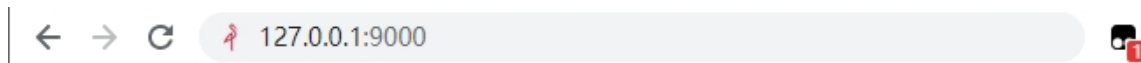
```
G:\minio>minio.exe server G:\minio
Endpoint: http://169.254.127.226:9000 http://192.168.43.119:9000 http://169.254.21.113:9000 http://169.254.46.3:9000
http://192.168.195.1:9000 http://192.168.20.1:9000 http://127.0.0.1:9000
AccessKey: duanjiajun
SecretKey: duanjiajun

Browser Access:
http://169.254.127.226:9000 http://192.168.43.119:9000 http://169.254.21.113:9000 http://169.254.46.3:9000 http://192.168.195.1:9000 http://192.168.20.1:9000 http://127.0.0.1:9000

Command-line Access: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc.exe config host add myminio http://169.254.127.226:9000 duanjiajun duanjiajun

Object API (Amazon S3 compatible):
Go: https://docs.min.io/docs/golang-client-quickstart-guide
Java: https://docs.min.io/docs/java-client-quickstart-guide
Python: https://docs.min.io/docs/python-client-quickstart-guide
JavaScript: https://docs.min.io/docs/javascript-client-quickstart-guide
.NET: https://docs.min.io/docs/dotnet-client-quickstart-guide
```

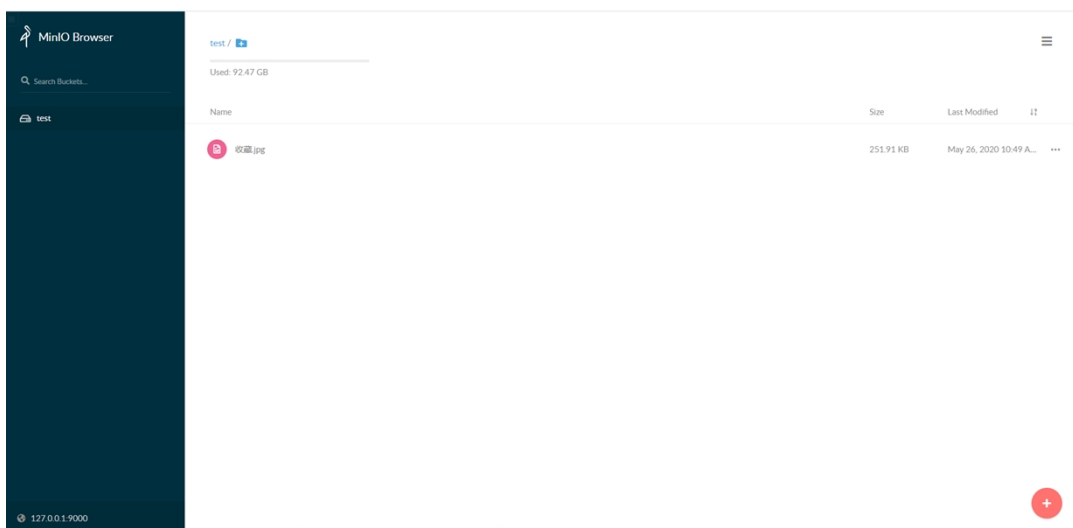
打开浏览器，在地址栏中键入：



进入 MinIO Browser 登陆界面，在界面输入终端中获取的 AccessKey 和 SecretKey 的值，如图所示：



点击登录按钮，进入主界面，如图所示：

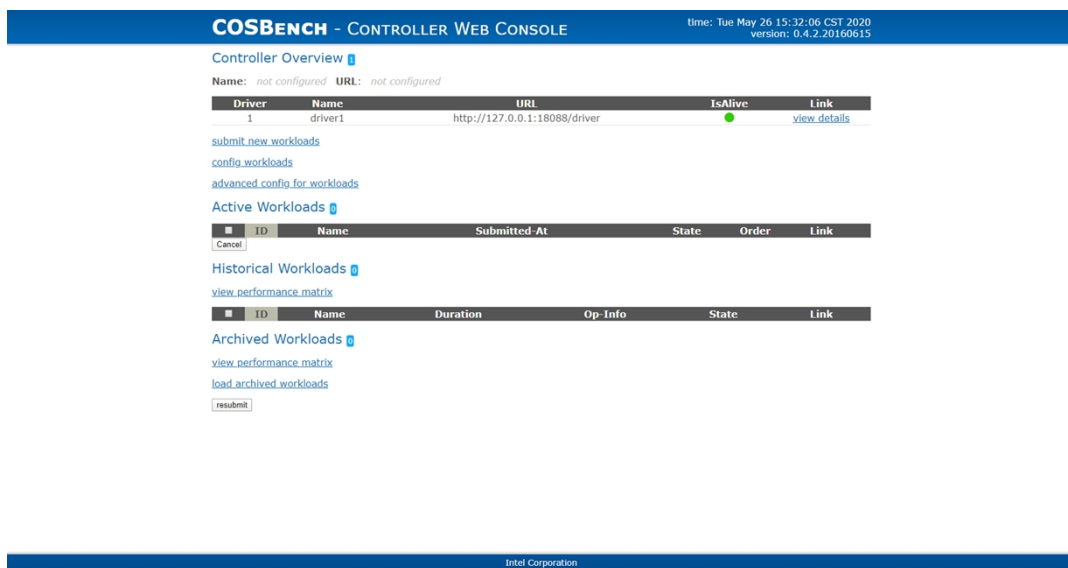


3.在 github 上下载 COSBench 压缩包，解压，双击运行 start-all.bat，结果如图所示。可以看到该批处理文件打开了 driver 和 controller 两个窗口，分别在端口 18089 和 19089 监听。

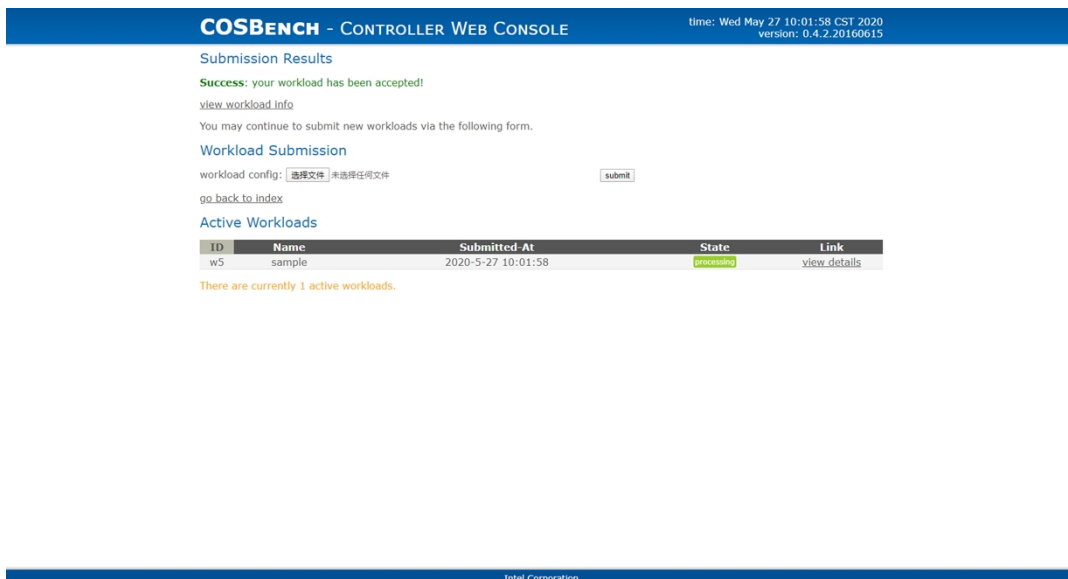
```
COSBench Controller - start-controller.bat
C:\Users\段佳俊\Desktop\0.4.2.c4\0.4.2.c4>java -Dcosbench.tomcat.config=conf/controller-tomcat-server.xml -server -cp main/* org.eclipse.equinox.launcher.Main -configuration conf/.controller -console 19089
Listening on port 0.0.0.0/0.0.0.0:19089 ...
Persistence bundle starting...
Persistence bundle started.
!!! Service will listen on web port: 19088 !!!

选择COSBench Driver - start-driver.bat
C:\Users\段佳俊\Desktop\0.4.2.c4\0.4.2.c4>java -Dcosbench.tomcat.config=conf/driver-tomcat-server.xml -server -cp main/* org.eclipse.equinox.launcher.Main -configuration conf/.driver -console 18089
Listening on port 0.0.0.0/0.0.0.0:18089 ...
Persistence bundle starting...
Persistence bundle started.
!!! Service will listen on web port: 18088 !!!
```

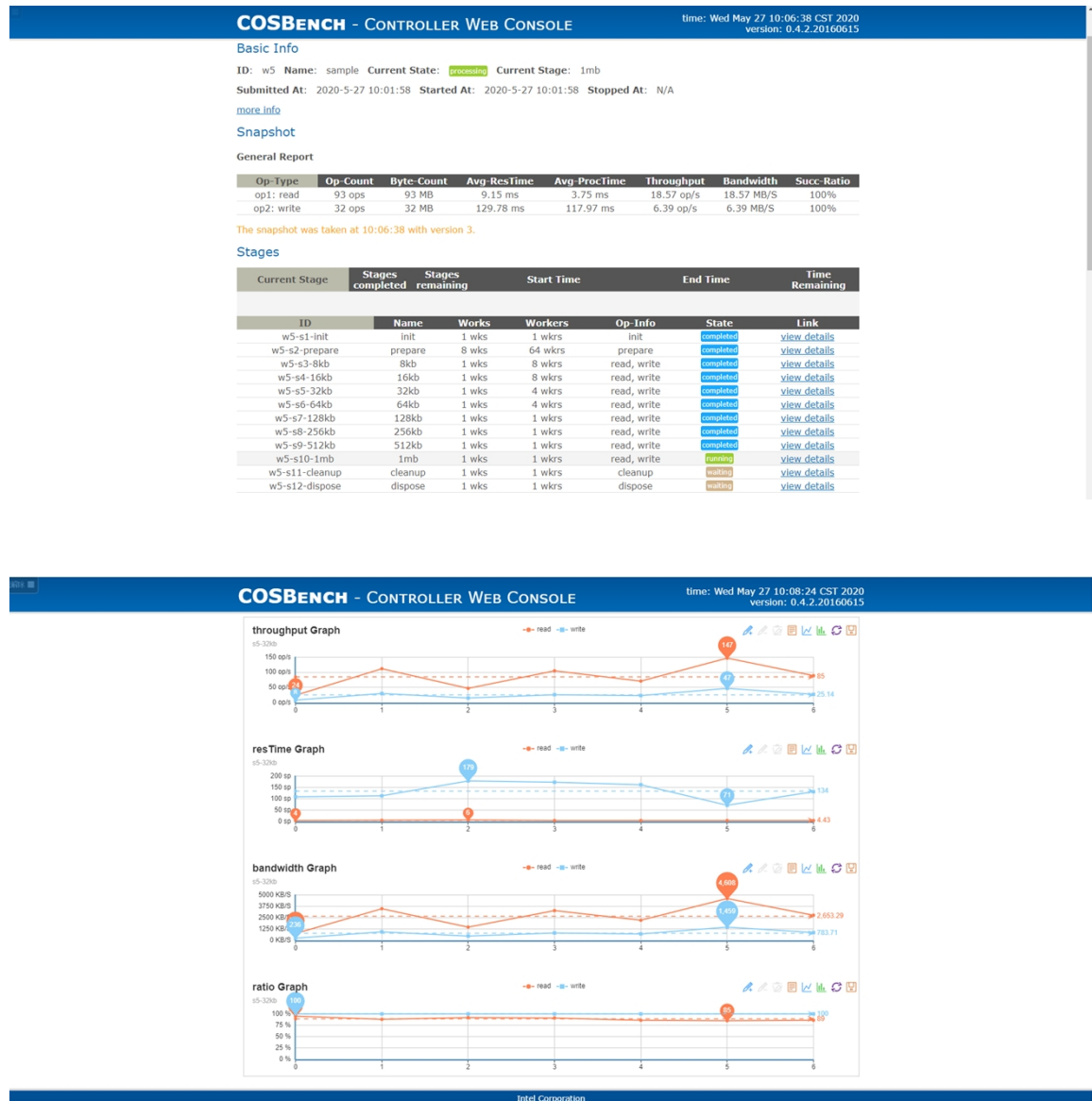
双击运行 web.bat，自动打开浏览器，弹出网页，如图所示：



5 修改负载样例的 AccessKey、SerectKey 和 endpoint，点击 submit new workloads，选择需要提交的文件，点击 submit 进行提交，提交完成后会在 Active Workloads 下显示相关信息，如图所示。



6. 点击 view details 可以查看刚才提交的负载的详细处理结果,运行结果如图所示。





## 7.读写性能比较

在 final result 下的 General Report, 中，可以看到 Op-count、Byte-count、Avg-ResTime、Avg-ProcTime、Throughput、Bandwidth 等信息，如图所示。

COSBENCH - CONTROLLER WEB CONSOLE							time: Wed May 27 10:08:24 CST 2020 version: 0.4.2.20160615
Final Result							
General Report							
Op-Type	Op-Count	Byte-Count	Avg-ResTime	Avg-ProcTime	Throughput	Bandwidth	Succ-Ratio
op1: init - write	0 ops	0 B	N/A	N/A	0 op/s	0 B/s	N/A
op1: prepare - write	8 ops	64 KB	1238.5 ms	1238.5 ms	6.48 op/s	51.85 KB/S	100%
op2: prepare - write	8 ops	128 KB	1087.38 ms	1087.12 ms	7.54 op/s	120.57 KB/S	100%
op3: prepare - write	8 ops	256 KB	1262.75 ms	1262 ms	6.34 op/s	202.91 KB/S	100%
op4: prepare - write	8 ops	512 KB	1238.25 ms	1237.12 ms	6.49 op/s	415.52 KB/S	100%
op5: prepare - write	8 ops	1.02 MB	1198.5 ms	1194.12 ms	6.81 op/s	872.21 KB/S	100%
op6: prepare - write	8 ops	2.05 MB	1186.88 ms	1177 ms	6.9 op/s	1.77 MB/S	100%
op7: prepare - write	8 ops	4.1 MB	541.62 ms	529.75 ms	21.4 op/s	10.96 MB/S	100%
op8: prepare - write	8 ops	8 MB	644.75 ms	469 ms	18.4 op/s	18.4 MB/S	100%
op1: read	4.83 kops	38.61 MB	6.79 ms	6.68 ms	161.4 op/s	1.29 MB/S	95.96%
op2: write	1.29 kops	10.35 MB	158.36 ms	158.33 ms	43.28 op/s	346.22 KB/S	100%
op1: read	2.79 kops	44.62 MB	4.95 ms	4.84 ms	95.49 op/s	1.53 MB/S	95.09%
op2: write	703 ops	11.25 MB	311.55 ms	311.53 ms	24.06 op/s	384.89 KB/S	100%
op1: read	2.89 kops	92.42 MB	4.34 ms	4.1 ms	98.46 op/s	3.15 MB/S	87.62%
op2: write	852 ops	27.26 MB	120.71 ms	120.67 ms	29.05 op/s	929.51 KB/S	100%
op1: read	2.98 kops	190.85 MB	4.88 ms	4.4 ms	99.43 op/s	6.36 MB/S	84.72%
op2: write	788 ops	50.43 MB	130.6 ms	130.29 ms	26.27 op/s	1.68 MB/S	100%
op1: read	1.54 kops	197.63 MB	4.96 ms	4.42 ms	51.51 op/s	6.59 MB/S	100%
op2: write	374 ops	47.87 MB	59.52 ms	58.52 ms	12.48 op/s	1.6 MB/S	100%
op1: read	1.44 kops	369.92 MB	6.27 ms	4.99 ms	48.26 op/s	12.36 MB/S	100%
op2: write	387 ops	99.07 MB	53.8 ms	51.45 ms	12.93 op/s	3.31 MB/S	100%
op1: read	1.42 kops	729.6 MB	7.24 ms	4.79 ms	47.63 op/s	24.39 MB/S	100%
op2: write	366 ops	187.39 MB	53.43 ms	48.25 ms	12.23 op/s	6.26 MB/S	100%
op1: read	677 ops	677 MB	9.76 ms	4.51 ms	22.57 op/s	22.57 MB/S	100%
op2: write	187 ops	187 MB	124.91 ms	113.58 ms	6.23 op/s	6.23 MB/S	100%
op1: cleanup - delete	128 ops	0 B	6.19 ms	6.19 ms	160.8 op/s	0 B/s	100%
op1: dispose - delete	0 ops	0 B	N/A	N/A	0 op/s	0 B/s	N/A
<a href="#">show performance details</a>							
Stages							
Intel Corporation							

可以看到，在采用 minio 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%。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%。读操作的 op 数和字节数比写操作更多，但平均处理时间却比写操作平均处理时间要短的多，吞吐量和带宽也是写操作的数倍，说明读写性能远远优于写性能。

## 8.实验中遇到的问题

在 Cosbench 网页端提交工作负载文件，运行负载后，没有产生预期数据，经过网上搜集资料，发现是没有修改负载文件中的用户名、密码和 endpoint 经过修改后，负载文件可以正确运行。

在进行 minio 实验时打开 start-all.bat 文件时报错，COSBench\_driver 控制台在运行时出现异常，通过检查命令行输出的信息，发现是该端口被其他的程序占用，使得程序运行出错，接着使用任务管理器，关闭占用的程序，重启 start-all.bat 文件后，顺利运行。

## 六、实验总结

通过这次实验，学习了如何搭建 minio 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。

在使用 COSBench 对所给负载样例进行测试时，最开始没有修改 AccessKey、SecretKey 和 endpoint，导致运行时出现 terminated，最终运行结果一直 fail，在查阅相关资料后对文件进行了修改，之后就比较顺利地完成了后面的内容。除此以外，还通过 COSBench 分析比较了不同块的大小和 workers 数对性能的影响。

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，对象存储系统只需要联网即可将数据上传至云端，有效解决了物联网设备本地存储空间较小的问题，是一个应对物联网海量数据存储挑战的良好的解决方案。它极大地扩大了存储系统的规模和功能，从根本上改变了存储蓝图，处理和解决了不间断可扩展性、弹性下降、限制数据持久性、无限技术更新和成本失控等棘手的存储问题。在工业界有着广泛的应用和出色的表现。

## 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.