



2019 级

《物联网数据存储与管理》课程

实 验 报 告

姓 名 薛海波

学 号 U201914971

班 号 物联网 1901 班

日 期 2022.04.16

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.....	1
4.2 对象存储性能分析.....	1
五、实验过程.....	2
5.1 对象存储技术实践.....	2
5.2 对象存储性能分析.....	5
六、实验总结.....	9
参考文献.....	9

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，分析性能问题，架设应用实践。

二、实验背景

本次实验使用的对象存储服务端为 minio。

MinIO 是一个基于 Apache License v2.0 开源协议的对象存储服务。它兼容亚马逊 S3 云存储服务接口，非常适合于存储大容量非结构化的数据，例如图片、视频、日志文件、备份数据和容器/虚拟机镜像等，而一个对象文件可以是任意大小，从几 kb 到最大 5T 不等。

MinIO 本身包含服务端和客户端。

本次实验使用的评测软件为 s3bench。

s3bench 提供了针对兼容 S3 的端点运行非常基本的吞吐量基准测试的功能。它执行一系列的 put 操作，然后执行一系列的 get 操作，并显示相应的统计信息。

三、实验环境

实验使用的软硬件环境如表 1 所示。

表 1 实验环境

硬件环境	CPU	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
	内存	16.0 GB
软件环境	操作系统	Windows 10 家庭中文版
	其它软件	minio RELEASE.2022-03-24T00-43-44Z mc RELEASE.2022-03-17T20-25-06Z s3bench

四、实验内容

本次实验主要内容如下：

1. 搭建系统环境，包括代码版本控制系统 Git，各种语言运行环境。
2. 对对象存储系统进行实践，采用 minio/mc 配置服务器端和客户端，并进行简单的创建或删除 bucket、上传或删除文件等操作。
3. 对对象存储系统进行测试，采用 s3bench 进行负载测试。

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，并熟悉各项操作。
2. 配置 minio 客户端 mc，在命令行下输入命令实现创建、删除 bucket 等操作，并通过访问网址查看新建及删除的结果。

4.2 对象存储性能分析

1. 读写性能对比。
2. 修改脚本范例中 numClients 参数的值，从而修改同一时间产生的请求数，

即并发数，分析并发数对存储性能的影响。

3. 修改脚本范例中 `numSamples` 参数的值，从而修改 `workers` 的数量，分析 `workers` 数量对存储性能的影响。

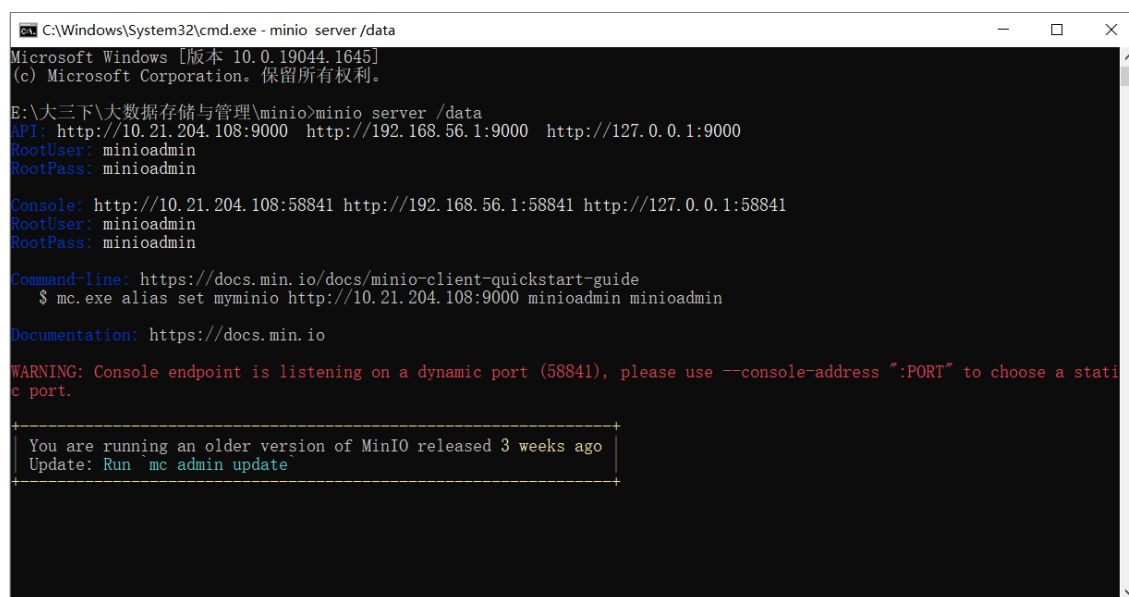
4. 修改脚本范例中 `objectSize` 参数的值，从而修改文件的大小，分析块文件大小对存储性能的影响。

五、实验过程

5.1 对象存储技术实践

下载 Windows 系统对应版本的 minio server 和 minio client。

按照官网上的例子在命令行下运行 `minio.exe`，运行结果如图 1 所示。



```
C:\Windows\System32\cmd.exe - minio server /data
Microsoft Windows [版本 10.0.19044.1645]
(c) Microsoft Corporation. 保留所有权利。

E:\大三下\大数据存储与管理\minio>minio server /data
API: http://10.21.204.108:9000 http://192.168.56.1:9000 http://127.0.0.1:9000
RootUser: minioadmin
RootPass: minioadmin

Console: http://10.21.204.108:58841 http://192.168.56.1:58841 http://127.0.0.1:58841
RootUser: minioadmin
RootPass: minioadmin

Command-line: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc.exe alias set myminio http://10.21.204.108:9000 minioadmin minioadmin

Documentation: https://docs.min.io

WARNING: Console endpoint is listening on a dynamic port (58841), please use --console-address ":PORT" to choose a static port.

You are running an older version of MinIO released 3 weeks ago
Update: Run `mc admin update`
```

图 1 启动 minio server

根据打印的信息可以看到，调用 minio 服务端的 API 的 url 为本机网络 IP 的 9000 端口，minio 服务端控制台的 url 为本机网络 IP 的 158841 端口。同时，还可以看出，两个监听端口对应的 `RootUser` 和 `RootPass` 均为默认的 `minioadmin`。

另外，红字警告表示控制台的监听端口为动态端口，而非静态端口。

然后，根据打印信息，用浏览器访问 <http://127.0.0.1:58841>，打开 minio 控制台，进入时用打印信息中默认的 `RootUser` 和 `RootPass` 登录，访问结果如图 2 所示。

另外，经测试发现，如果借助浏览器访问 <http://127.0.0.1:9000> 等 API 的 url，浏览器仍会重定向至控制台所监听的端口。

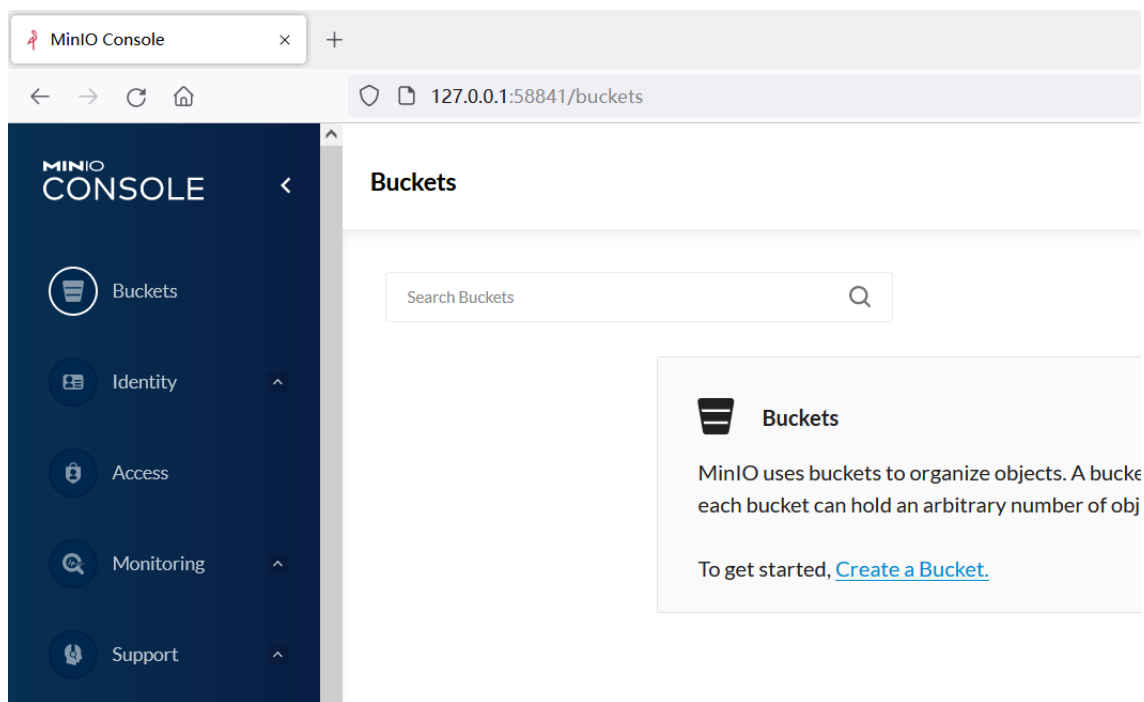


图 2 minio 控制台

点击页面右侧的 Create Bucket 可以创建新的 bucket，创建完成后如图 3 所示。该 bucket 将用于接下来的性能测试。

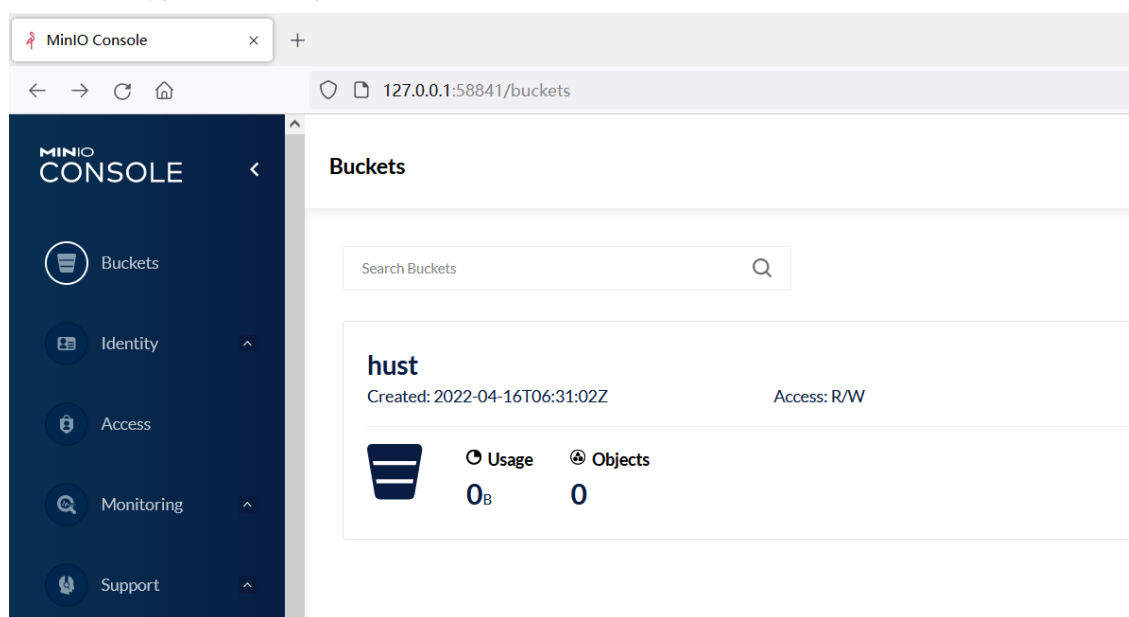


图 3 新建 bucket

接下来，按照官网上的例子在命令行下运行 `mc.exe`，添加一个名为 `iot` 的存储服务。访问存储服务中的 bucket 和对象，可以看到之前创建的 bucket。通过指令，从客户端再创建一个名为 `hello` 的新 bucket，再次访问存储服务中的 bucket 和对象，可以看到新加入的 bucket。结果如图 4 所示。

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19044.1645]
(c) Microsoft Corporation。保留所有权利。

E:\大三下\大数据存储与管理\minio>mc config host add iot http://127.0.0.1:9000 minioadmin minioadmin --api s3v4
Added iot successfully.

E:\大三下\大数据存储与管理\minio>mc ls iot
[2022-04-16 14:31:02 CST] 0B hust/

E:\大三下\大数据存储与管理\minio>mc mb iot/hello
Bucket created successfully iot/hello.

E:\大三下\大数据存储与管理\minio>mc ls iot
[2022-04-16 16:47:13 CST] 0B hello/
[2022-04-16 14:31:02 CST] 0B hust/

E:\大三下\大数据存储与管理\minio>
```

图 4 运行 mc

回到 minio 的网页控制台，刷新后可以看到刚才新创建的 bucket，显示结果如图 5 所示。

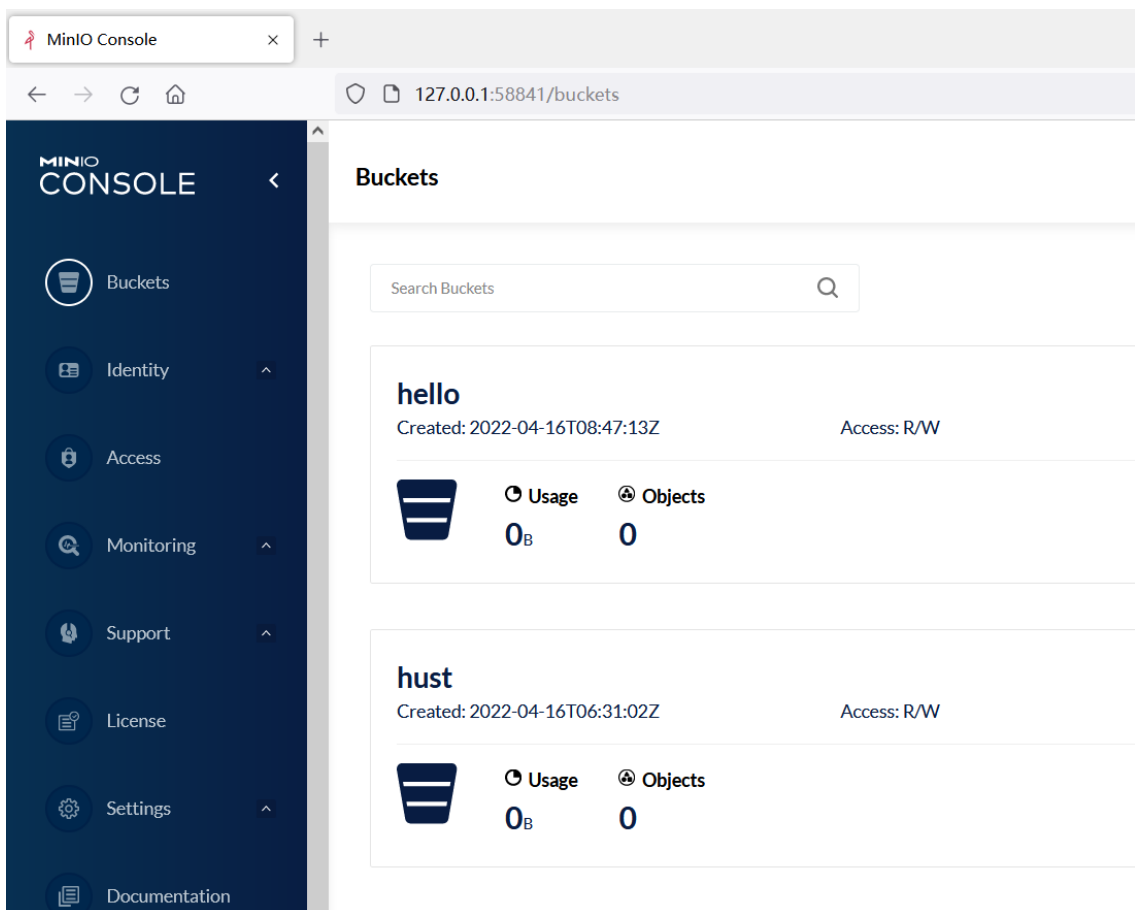


图 5 网页显示结果

回到命令行，通过指令删除刚才创建的新 bucket，再次访问存储服务中的 bucket 和对象，可以发现 bucket 已经没有了。结果如图 6 所示。另外刷新网页控制台后可以同样发现只剩下一个 bucket。

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19044.1645]
(c) Microsoft Corporation. 保留所有权利。

E:\大三下\大数据存储与管理\minio>mc config host add iot http://127.0.0.1:9000 minioadmin minioadmin --api s3v4
Added iot successfully.

E:\大三下\大数据存储与管理\minio>mc ls iot
[2022-04-16 14:31:02 CST]    0B hust/

E:\大三下\大数据存储与管理\minio>mc mb iot/hello
Bucket created successfully iot/hello.

E:\大三下\大数据存储与管理\minio>mc ls iot
[2022-04-16 16:47:13 CST]    0B hello/
[2022-04-16 14:31:02 CST]    0B hust/

E:\大三下\大数据存储与管理\minio>mc rb iot/hello
Removed iot/hello successfully.

E:\大三下\大数据存储与管理\minio>mc ls iot
[2022-04-16 14:31:02 CST]    0B hust/

E:\大三下\大数据存储与管理\minio>
```

图 6 删除 bucket

5.2 对象存储性能分析

下载 Windows 系统对应版本 s3bench.exe 及对应的命令脚本的范例脚本 s3bench.amd。

修改脚本范例，分别将 accessKey 和 accessSecret 的值改为 minioadmin，将 bucket 的值改为 hust，修改后如图 7 所示。

```
s3bench.exe ^
-accessKey=minioadmin ^
-accessSecret=minioadmin ^
-bucket=hust ^
-endpoint=http://127.0.0.1:9000 ^
-numClients=8 ^
-numSamples=256 ^
-objectNamePrefix=loadgen ^
-objectSize=1024
pause
```

图 7 本地化的脚本

将 s3bench.exe 和 s3bench.amd 放在同一目录下。运行 s3bench.amd 脚本，运行结果如图 8 所示。

```

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:          hust
objectNamePrefix: loadgen
objectSize:       0.0010 MB
numClients:       8
numSamples:       256
verbose:          %!d(bool=false)

Results Summary for Write Operation(s)
Total Transferred: 0.250 MB
Total Throughput:  0.17 MB/s
Total Duration:    1.458 s
Number of Errors:  0

-----
Write times Max:    0.099 s
Write times 99th %ile: 0.088 s
Write times 90th %ile: 0.071 s
Write times 75th %ile: 0.058 s
Write times 50th %ile: 0.045 s
Write times 25th %ile: 0.032 s
Write times Min:    0.014 s

Results Summary for Read Operation(s)
Total Transferred: 0.250 MB
Total Throughput:  2.14 MB/s
Total Duration:    0.117 s
Number of Errors:  0

-----
Read times Max:     0.008 s
Read times 99th %ile: 0.007 s
Read times 90th %ile: 0.006 s
Read times 75th %ile: 0.004 s
Read times 50th %ile: 0.003 s
Read times 25th %ile: 0.003 s
Read times Min:     0.001 s

```

图 8 运行结果

将脚本范例中 numClients 参数的值取 8、16、24，将脚本范例中 numSamples 参数的值取 256、512、768，将脚本范例中 objectSize 参数的值取 1024、2048、3072。分别再次测试，一共可以得到 3*3*3=27 组数据。在参数取这些值的情况下，服务器的吞吐率和延迟如表 2 所示。

表 2 不同条件下的吞吐率和延迟

Clients (并发数)	Samples (样本数)	objectSize (文件大小)	Write (写入)				Read (读取)			
			Throughput (吞吐率)	延迟 90%	延迟 99%	延迟 最大	Throughput (吞吐率)	延迟 90%	延迟 99%	延迟 最大
8	256	1024	0.18	0.067	0.102	0.106	2.13	0.005	0.007	0.007
16	256	1024	0.16	0.128	0.148	0.152	2.04	0.011	0.017	0.022
24	256	1024	0.12	0.236	0.307	0.32	2.06	0.017	0.026	0.032
8	512	1024	0.17	0.072	0.098	0.115	2.06	0.006	0.01	0.012
16	512	1024	0.15	0.15	0.192	0.231	2.06	0.012	0.019	0.023
24	512	1024	0.15	0.215	0.265	0.303	2.07	0.017	0.024	0.033
8	768	1024	0.16	0.078	0.1	0.105	2.33	0.005	0.007	0.01
16	768	1024	0.13	0.156	0.185	0.288	2.2	0.01	0.016	0.023
24	768	1024	0.14	0.202	0.24	0.265	2.19	0.017	0.027	0.03
8	256	2048	0.35	0.07	0.1	0.101	4.03	0.006	0.008	0.008
16	256	2048	0.3	0.139	0.176	0.18	4.08	0.012	0.018	0.022
24	256	2048	0.29	0.232	0.274	0.276	3.94	0.018	0.029	0.033
8	512	2048	0.34	0.073	0.104	0.115	4.82	0.005	0.006	0.007
16	512	2048	0.3	0.133	0.152	0.157	4.02	0.012	0.019	0.024

24	512	2048	0.29	0.199	0.237	0.262	4.09	0.018	0.026	0.039
8	768	2048	0.31	0.082	0.112	0.129	4.88	0.004	0.007	0.008
16	768	2048	0.27	0.17	0.214	0.219	4.11	0.011	0.014	0.017
24	768	2048	0.27	0.214	0.252	0.312	3.5	0.02	0.033	0.044
8	256	3072	0.52	0.075	0.099	0.102	6.42	0.006	0.009	0.009
16	256	3072	0.43	0.139	0.167	0.17	5.75	0.011	0.017	0.021
24	256	3072	0.12	0.22	0.0281	0.283	5.54	0.018	0.023	0.029
8	512	3072	0.45	0.08	0.1	0.111	6.25	0.006	0.009	0.01
16	512	3072	0.42	0.156	0.185	0.215	6.02	0.012	0.024	0.03
24	512	3072	0.38	0.239	0.289	0.322	5.44	0.018	0.024	0.026
8	768	3072	0.49	0.075	0.095	0.102	6.78	0.005	0.013	0.018
16	768	3072	0.41	0.155	0.214	0.217	6.36	0.011	0.016	0.025
24	768	3072	0.42	0.222	0.259	0.313	5.59	0.019	0.025	0.036

分析以上表单数据，可以得出服务器的读取速率远大于写入速率，符合一般认知。具体来看，随着 Clients 并发数的提升，吞吐率逐渐下降；随着 Samples 样本数的提升，吞吐率基本不变；随着 object size 文件大小的提升，吞吐率逐渐上升，且上升幅度比 Clients 并发数大。

这三个参数变量与吞吐率的折线关系如图 9、图 10 所示。这两张图中，横坐标表示 Clients 并发数，纵坐标表示实际吞吐率（单位为 mb/s），每条折线命名为：Samples 样本数*object size 文件大小。

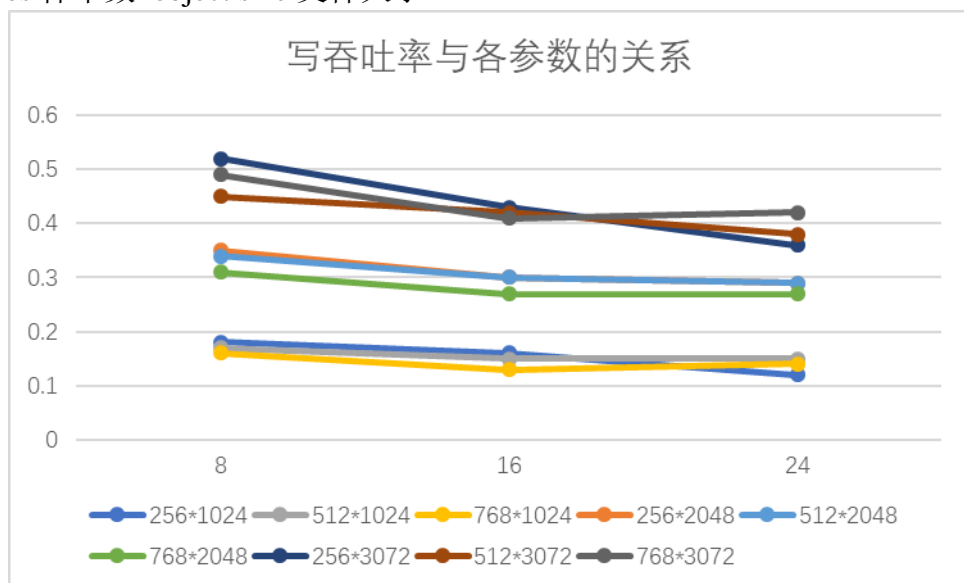


图 9 写吞吐率

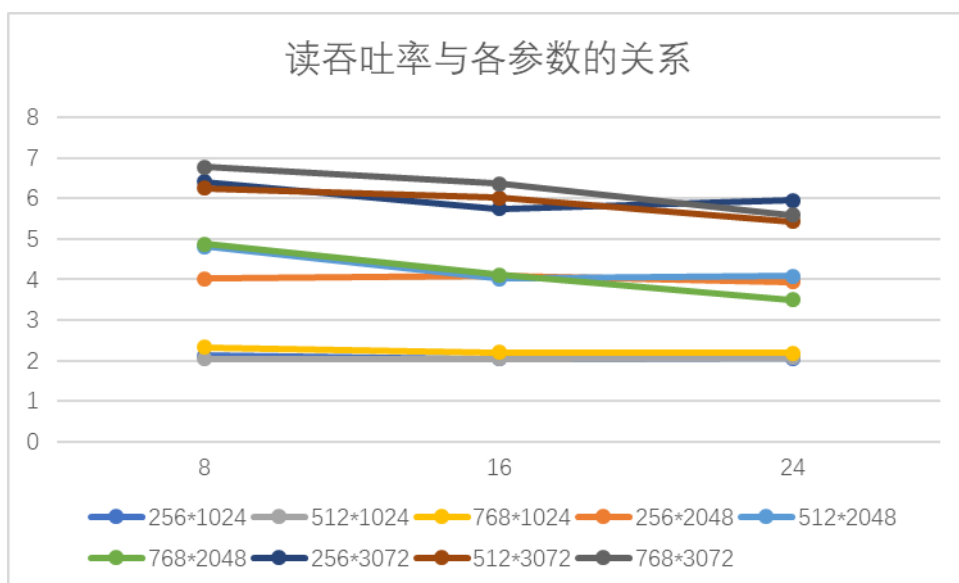


图 10 读吞吐率

除吞吐率外，以上表单数据还显示了对于 90%、99% 以及全部数据的最高延迟。可以看出，Samples 样本数和 object size 文件大小对读写延迟基本没有影响，但随着 Clients 并发数的提升，读写延迟也随之上升。由于数据的读写存在偶然情况，因此选取 90% 数据的延迟与三个参数绘制折线图。如图 11、图 12 所示，可以看出服务器的读写延迟与 Clients 并发数基本呈正比关系。

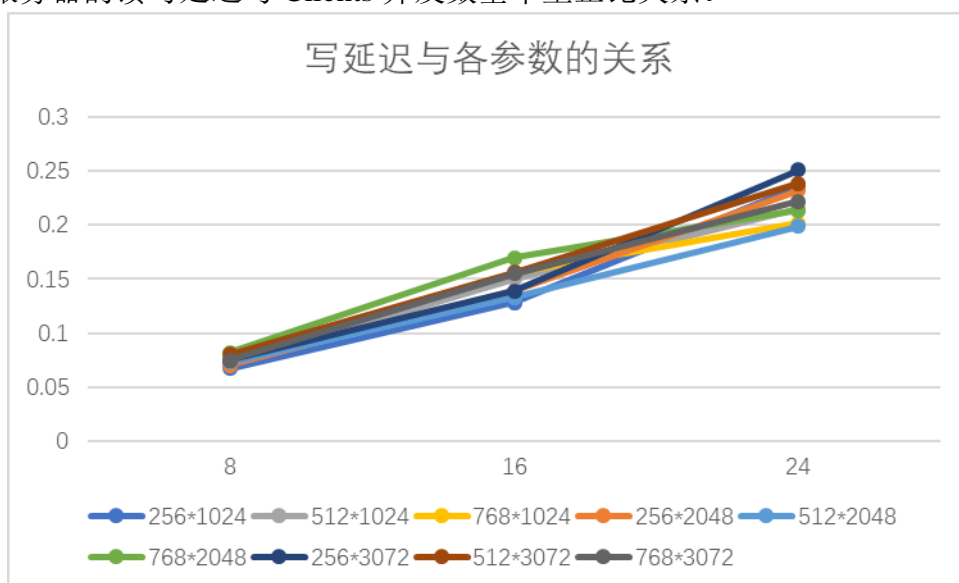


图 11 写延迟

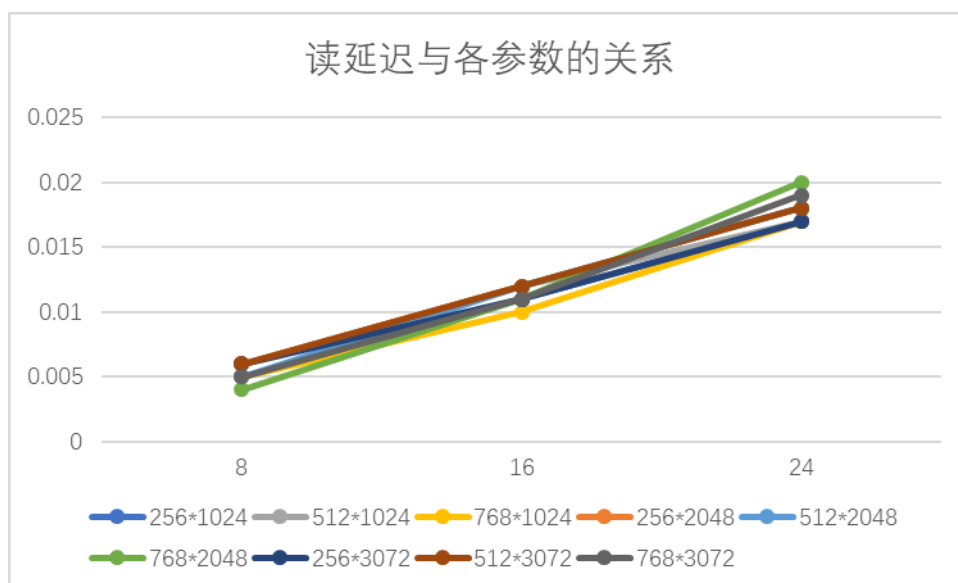


图 12 读延迟

根据以上分析可以得出结论，对于 minio，在低并发数的情况下读写大文件性能较好。

六、实验总结

本次实验让我了解了不少与对象存储技术相关的知识。一般来说，这些技术的应用平台以 Linux 为主，最初我也是尝试在 Linux 平台上实践，然而遇到了不少困难，最后不得不转向 Windows 平台。

我在本次实验中使用的服务端软件为 minio，测评软件为 s3bench，这都是十分基本的软件，主要面向初学者，使用过程中基本不存在问题。我在本次实验里也并不是没有尝试过其它软件，但都遇到了很多困难，以至于放弃。只能说我在这方面还是学艺不精。

我在本次实验中学会了很多东西，当然也有不少遗憾。抛开上面说过的哪些，在 minio 和 s3bench 部分，最主要的地方在于我测试的数据不够多，以至于绘制的折线图显得有些缺少说服力。

参考文献

- [1] 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.
- [2] ARNOLD J. OpenStack Swift[M]. O' Reilly Media, 2014.
- [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] Dean J, Barroso L A. Association for Computing Machinery, 2013. The Tail at Scale[J]. Commun. ACM, 2013, 56(2): 74 - 80.
- [5] Delimitrou C, Kozyrakis C. Association for Computing Machinery, 2018. Amdahl's Law for Tail Latency[J]. Commun. ACM, 2018, 61(8): 65 - 72.

(可以根据实际需要更新调整)