



2017 级

《物联网数据存储与管理》课程

实 验 报 告

姓 名 王潇逸

学 号 U201714584

班 号 物联网 1701 班

日 期 2020.06.06

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.....	2
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.....	10
参考文献.....	10

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

物联网的应用随着移动互联网的兴起而蓬勃发展，快速增长的物联网数据带来了存储挑战：

1. 庞大：以几何级数不断增长的数据量，对存储和处理带来挑战。
2. 复杂：数据内容多样性带来的数据异构，使得数据结构越来越复杂。

因此，需要一个高可用、可扩展的海量数据存储系统来满足物联网存储需求。面向对象存储技术提供了一种解决方案。对象存储同兼具 SAN 高速直接访问磁盘特点及 NAS 的分布式共享特点。将数据通路（数据读或写）和控制通路（元数据）分离，并且基于对象存储设备构建存储系统，每个对象存储设备具有一定的智能，能够自动管理其上的数据分布。

三、实验环境

由于本次实验均在虚拟机中运行，故此处直接用属性截图展示实验具体环境，如图 1 所示。



图 1 实验所使用的虚拟机环境配置

由于本次实验中所采用的一些模块需要使用 GO 环境，本实验所使用的 GO 环境版本如图 2 所示。

```
sak16046401@sak16046401-VirtualBox:~$ go version
go version go1.6.2 linux/amd64
```

图 2 实验所使用的 GO 环境版本

四、实验内容

根据给出的实验教程，自主选择合适的对象存储服务端、对象存储客户端、对象存储评测工具进行实验。

4.1 对象存储技术实践

1. 搭建对象存储服务端，选择使用 Minio。
2. 搭建对象存储客户端，选择使用 osm，作为 Go 的一个模块进行安装。
3. 测试对象存储基本功能是否正常。

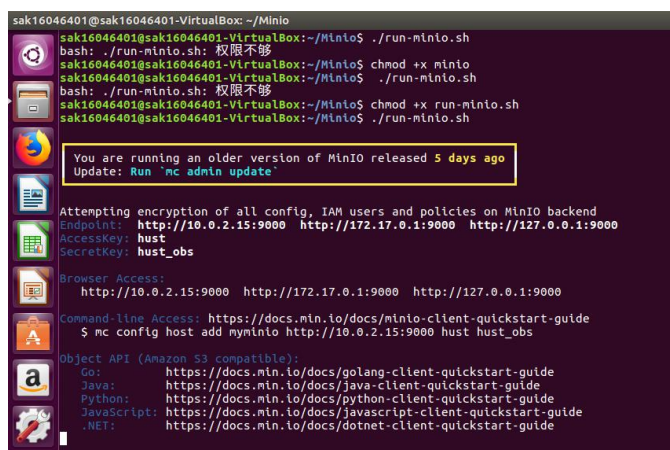
4.2 对象存储性能分析

1. 搭建对象评测工具环境，选择 S3 Bench，作为 Go 的一个模块进行安装。
2. 调整对象存储评测参数，包括客户端数量、对象数量、对象大小，观察数据存储性能。
3. 编写 shell 脚本实现数据批量测试，并重定向 bash 终端将结果输出到文本文件，以便进一步分析处理。
4. 整理和分析不同参数下存储性能数据。

五、实验过程

5.1 搭建对象存储服务端

1. Minio 服务端是拆箱即用的，只需下载二进制可执行文件到本地后修改执行权限即可。执行参考资料中的运行脚本，以指定的访问密钥和安全密钥运行服务端，如图 3 所示。



```
sak16046401@sak16046401-VirtualBox: ~/Minio
sak16046401@sak16046401-VirtualBox:~/Minio$ ./run-minio.sh
bash: ./run-minio.sh: 权限不够
sak16046401@sak16046401-VirtualBox:~/Minio$ chmod +x minio
sak16046401@sak16046401-VirtualBox:~/Minio$ ./run-minio.sh
bash: ./run-minio.sh: 权限不够
sak16046401@sak16046401-VirtualBox:~/Minio$ chmod +x run-minio.sh
sak16046401@sak16046401-VirtualBox:~/Minio$ ./run-minio.sh

You are running an older version of MinIO released 5 days ago
Update: Run "mc admin update"

Attempting encryption of all config, IAM users and policies on MinIO backend
Endpoints: http://10.0.2.15:9000 http://172.17.0.1:9000 http://127.0.0.1:9000
AccessKey: hust
SecretKey: hust_obs

Browser Access:
http://10.0.2.15:9000 http://172.17.0.1:9000 http://127.0.0.1:9000

Command-line Access: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc config host add myminio http://10.0.2.15:9000 hust hust_obs

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
```

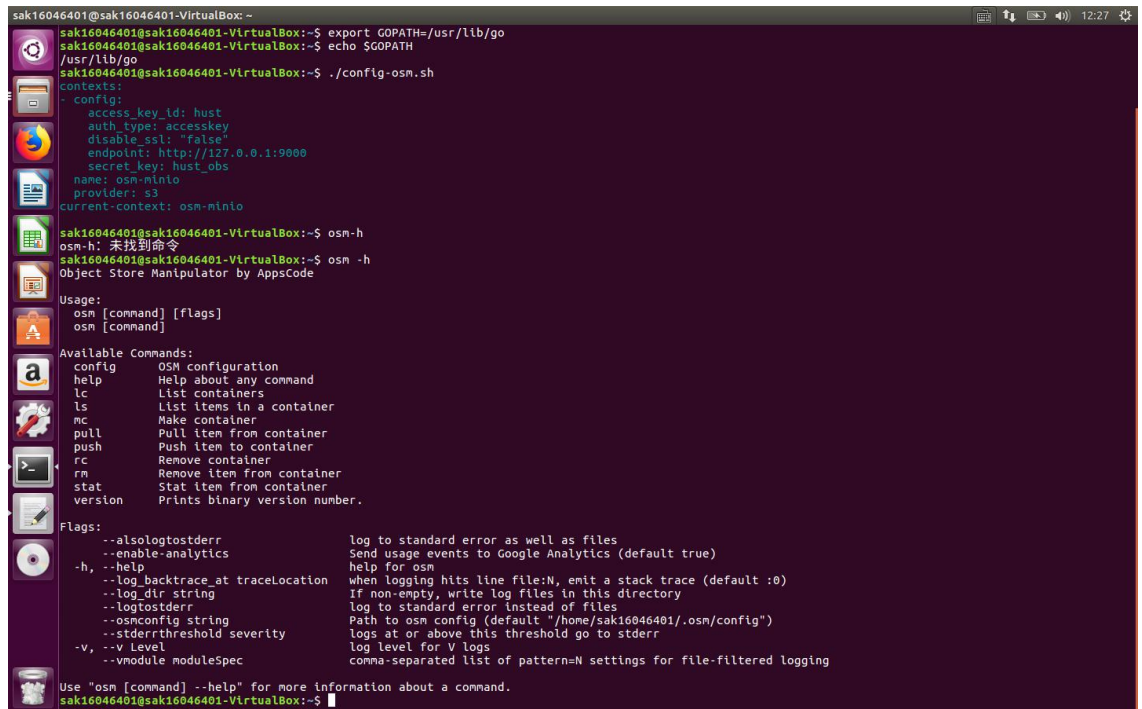
图 3 本地运行 minio 服务端

5.2 搭建对象存储客户端

1. 将 osm 作为一个 Go 模块加载，在此之前需要先安装 GO 环境。然后从 github 获取到代码包，并将其放到 GOPATH 目录下的 src/github.com 文件夹中，之后使用

go install 命令进行模块安装。

2. 安装完成后在 GOPATH 目录的 bin 目录下会出现 osm 的可执行文件。
3. 运行 config-osm.sh 脚本配置 osm 参数。
4. 输入 osm -h 测试是否安装配置成功，如图 4 所示。



```
sak16046401@sak16046401-VirtualBox:~$ export GOPATH=/usr/lib/go
sak16046401@sak16046401-VirtualBox:~$ echo $GOPATH
/usr/lib/go
sak16046401@sak16046401-VirtualBox:~$ ./config-osm.sh
contexts:
- config:
  access_key_id: hust
  auth_type: accesskey
  disable_ssl: "false"
  endpoint: http://127.0.0.1:9000
  secret_key: hust_obs
  name: osm-minio
  provider: s3
  current-context: osm-minio

sak16046401@sak16046401-VirtualBox:~$ osm-h
osm-h: 未找到命令
sak16046401@sak16046401-VirtualBox:~$ osm -h
Object Store Manipulator by AppCode

Usage:
  osm [command] [flags]
  osm [command]

Available Commands:
  config  OSM configuration
  help    Help about any command
  lc      List containers
  ls      List items in a container
  mc      Make container
  pull    Pull item from container
  push    Push item to container
  rc      Remove container
  rm      Remove item from container
  stat    Stat item from container
  version Prints binary version number.

Flags:
  --alsologtostderr          log to standard error as well as files
  --enable-analytics         Send usage events to Google Analytics (default true)
  -h, --help                 help for osm
  --log_backtrace_at traceLocation when logging hits line file:N, emit a stack trace (default :0)
  --log_dir string           If non-empty, write log files in this directory
  --logtostderr              log to standard error instead of files
  --osmconfig string         Path to osm config (default "/home/sak16046401/.osm/config")
  --stderrthreshold severity logs at or above this threshold go to stderr
  -v, --v Level              log level for V logs
  --vmodule moduleSpec       comma-separated list of pattern=N settings for file-filtered logging

Use "osm [command] --help" for more information about a command.
sak16046401@sak16046401-VirtualBox:~$
```

图 4 osm 安装和配置

5.3 测试对象存储功能

1. 运行 minio 服务端，在浏览器打开图形管理界面，访问 127.0.0.1:9000，输入密钥，即可查看服务端情况，如图 5 所示。

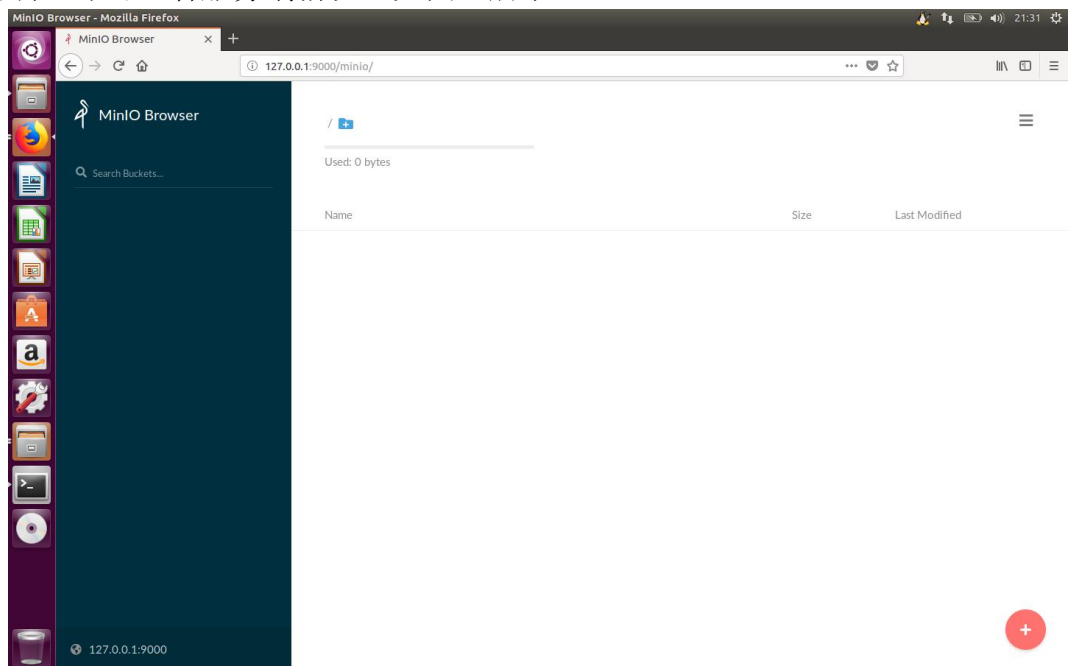


图 5 minio 服务端图形界面

2. 在终端窗口内利用 osm 上传文件，如图 6 所示。

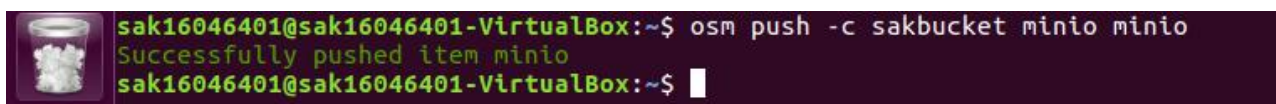


图 6 本地终端利用 osm 上传文件

3. 本地上传成功后，刷新 minio 服务端，查看文件上传情况，如图 7 所示。可以看到建立的仓库中有了对应文件，即上传成功。

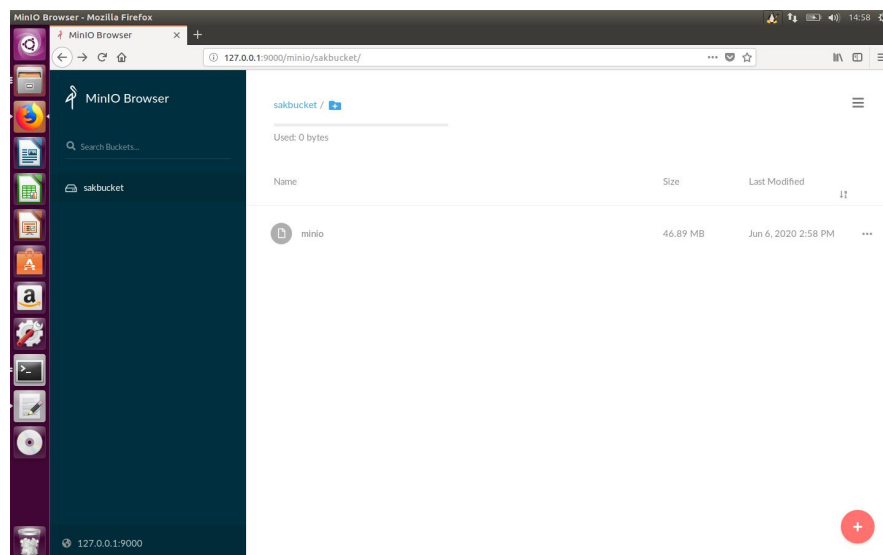


图 7 minio 服务端的文件上传情况

5.4 对象存储性能分析

1. S3 Bench 作为 GO 模块加载，先下载 aws-sdk-go 和 go-jmespath 依赖包进行安装，再安装 s3bench。安装完成后在 GOPATH 的 bin 目录下会出现 s3bench 可执行文件，如图 8 所示。

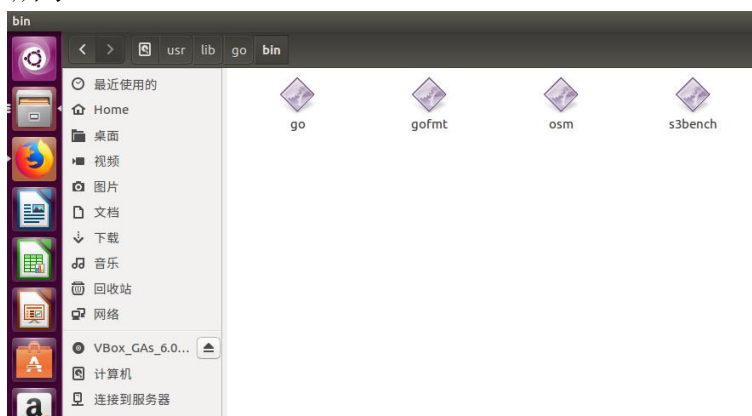


图 8 GOPATH 的 bin 目录下的可执行文件

2. 由于 S3 Bench 没有批量测试和输出数据为文件的功能，故通过编写 shell 脚本进行批量测试，并将 bash 的输出重定向到文本文件，具体测试步骤如下：

①对象存储大小从 1KB 增加到 1MB，并发客户端数量为 1，对象数量为 100，观察对象大小对存储性能的影响。

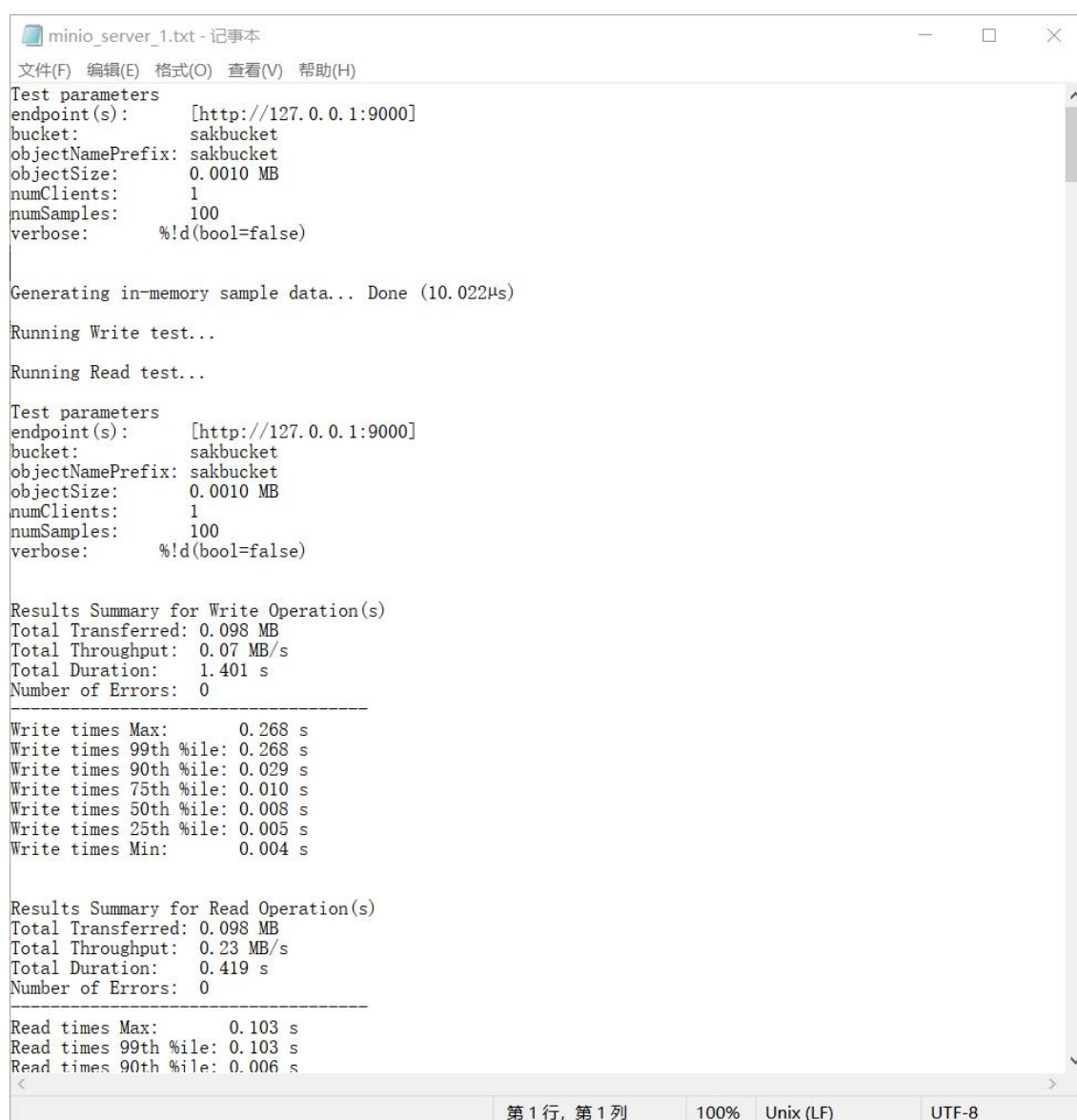
②并发客户端从 1 增加到 100，对象数量为 100，对象大小为 1KB，观察并发客户端数量对存储性能的影响。

③并发客户端从 1 增加到 100，对象数量为 100，对象大小为 100KB，与测试 2 的内容进行比较。

④并发客户端数量为 1，对象数量从 5 增加到 1280，对象大小为 1KB，观察对象数量对存储性能的影响。

由于所使用脚本文件多大同小异，仅通过修改相关参数的方式改变功能，故将 4 个脚本文件(s3bench1,s3bench2,s3bench3 和 s3bench4)均存放在附件中，此处不再赘述。

3.执行测试脚本，输出重定向文件。此处以步骤 1 的输出文件内容为例，如图 9 所示。



```
minio_server_1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           sakbucket
objectNamePrefix: sakbucket
objectSize:       0.0010 MB
numClients:       1
numSamples:       100
verbose:          %!d(bool=false)

Generating in-memory sample data... Done (10.022µs)

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           sakbucket
objectNamePrefix: sakbucket
objectSize:       0.0010 MB
numClients:       1
numSamples:       100
verbose:          %!d(bool=false)

Results Summary for Write Operation(s)
Total Transferred: 0.098 MB
Total Throughput:  0.07 MB/s
Total Duration:    1.401 s
Number of Errors:  0

-----
Write times Max:      0.268 s
Write times 99th %ile: 0.268 s
Write times 90th %ile: 0.029 s
Write times 75th %ile: 0.010 s
Write times 50th %ile: 0.008 s
Write times 25th %ile: 0.005 s
Write times Min:      0.004 s

Results Summary for Read Operation(s)
Total Transferred: 0.098 MB
Total Throughput:  0.23 MB/s
Total Duration:    0.419 s
Number of Errors:  0

-----
Read times Max:       0.103 s
Read times 99th %ile: 0.103 s
Read times 90th %ile: 0.006 s

< 第 1 行, 第 1 列 100% Unix (LF) UTF-8
```

图 9 重定向文件内容

4. 整理数据。将输出的结果文件中的数据整理到 Excel 表格中以便进一步处

理，如图 10 所示。

NO.	Clients-Samples-Size(MB)	W Throughput(MB/S)	W Duration(S)	W 99th(S)	W 90th(S)	R Throughput(MB/S)	R Duration(S)	R 99th(S)	R 90th(S)
1-1	1-100-0.0010	0.07	1.401	0.268	0.029	0.23	0.419	0.103	0.006
1-2	1-100-0.0020	0.21	0.917	0.078	0.011	1.20	0.163	0.019	0.002
1-3	1-100-0.0039	0.49	0.798	0.035	0.01	4.04	0.097	0.004	0.001
1-4	1-100-0.0098	1.25	0.779	0.018	0.01	7.73	0.126	0.009	0.002
1-5	1-100-0.0195	2.49	0.784	0.019	0.01	20.87	0.094	0.003	0.001
1-6	1-100-0.0391	4.83	0.808	0.044	0.01	30.9	0.126	0.008	0.002
1-7	1-100-0.0977	12.22	0.799	0.03	0.01	88.69	0.110	0.005	0.002
1-8	1-100-0.1953	12.48	1.565	0.734	0.01	141.09	0.138	0.006	0.002
1-9	1-100-0.3906	24.24	1.612	0.668	0.013	279.73	0.14	0.004	0.002
1-10	1-100-1	30.86	3.241	0.493	0.038	46.38	2.156	0.284	0.032
2-1	1-100-0.0010	0.11	0.894	0.082	0.1	0.47	0.207	0.062	0.003
2-2	2-100-0.0010	0.13	0.746	0.053	0.018	0.84	0.117	0.046	0.003
2-3	4-100-0.0010	0.29	0.336	0.072	0.026	1.00	0.098	0.030	0.006
2-4	8-100-0.0010	0.42	0.231	0.036	0.025	0.95	0.103	0.072	0.012
2-5	16-100-0.0010	0.49	0.198	0.057	0.054	0.64	0.153	0.150	0.134
2-6	32-100-0.0010	0.53	0.185	0.124	0.079	0.78	0.126	0.121	0.116
2-7	64-100-0.0010	0.60	0.163	0.142	0.122	0.58	0.168	0.131	0.124
2-8	70-100-0.0010	0.75	0.131	0.106	0.094	0.78	0.125	0.117	0.107
2-9	80-100-0.0010	0.39	0.247	0.239	0.232	0.81	0.120	0.115	0.103
2-10	90-100-0.0010	0.61	0.160	0.154	0.144	0.78	0.125	0.118	0.109
2-11	100-100-0.0010	0.52	0.189	0.179	0.175	0.75	0.129	0.122	0.115
3-1	1-100-0.0977	12.61	0.775	0.016	0.010	83.31	0.117	0.005	0.002
3-2	2-100-0.0977	21.50	0.454	0.026	0.012	92.63	0.105	0.031	0.002
3-3	4-100-0.0977	26.44	0.369	0.040	0.025	92.73	0.105	0.044	0.08
3-4	8-100-0.0977	31.94	0.306	0.050	0.035	68.60	0.142	0.056	0.033
3-5	16-100-0.0977	33.15	0.295	0.101	0.066	83.13	0.117	0.093	0.025
3-6	32-100-0.0977	37.82	0.258	0.145	0.132	86.25	0.113	0.107	0.1
3-7	64-100-0.0977	33.17	0.294	0.232	0.203	74.89	0.130	0.115	0.104
3-8	70-100-0.0977	36.42	0.268	0.235	0.195	72.34	0.135	0.125	0.096
3-9	80-100-0.0977	35.03	0.279	0.265	0.228	71.97	0.136	0.129	0.12
3-10	90-100-0.0977	35.80	0.273	0.256	0.245	70.54	0.138	0.127	0.116
3-11	100-100-0.0977	33.90	0.288	0.273	0.260	68.53	0.142	0.134	0.126
4-1	1-5-0.0010	0.03	0.145	0.116	0.116	1.15	0.004	0.001	0.001
4-2	1-10-0.0010	0.15	0.055	0.009	0.009	0.50	0.02	0.006	0.006
4-3	1-20-0.0010	0.13	0.147	0.011	0.010	0.76	0.026	0.004	0.002
4-4	1-40-0.0010	0.13	0.298	0.010	0.009	0.94	0.042	0.004	0.002
4-5	1-80-0.0010	0.12	0.640	0.037	0.010	1.00	0.078	0.004	0.002
4-6	1-160-0.0010	0.13	1.245	0.016	0.015	1.05	0.149	0.003	0.001
4-7	1-320-0.0010	0.12	2.538	0.023	0.010	1	0.312	0.003	0.001
4-8	1-640-0.0010	0.12	5.109	0.025	0.010	1.04	0.601	0.003	0.001
4-9	1-1280-0.0010	0.12	10.507	0.029	0.010	0.97	1.290	0.004	0.001

图 10 整理后的数据

5. 分析结果

对本次实验的结果，从性能和传输速率两方面进行分析。

性能方面：

依据图 10 分别作出对象大小、并发客户端数量、对象数量对性能影响的折线图，分别如图 11、12、13、14 所示。

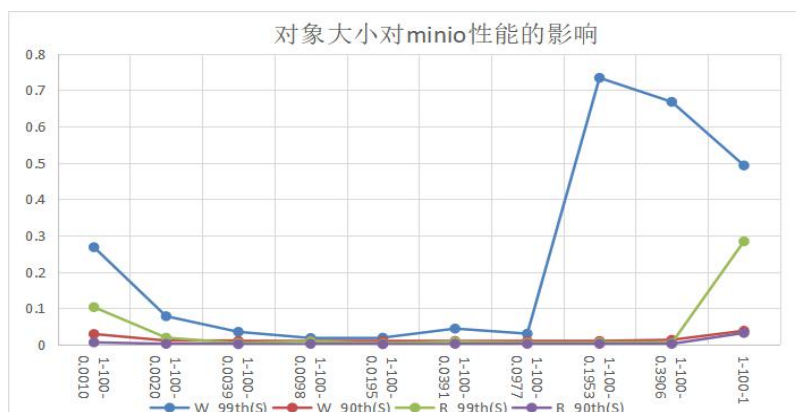


图 11 对象大小对性能的影响

从图 11 可知，读取延迟无较大的变化，而写入延迟随对象大小增大呈现先减小

再增大的变化。

根据上述分析，可知对象大小越大时，吞吐率越大，且延迟在对象较小和较大时比较高。

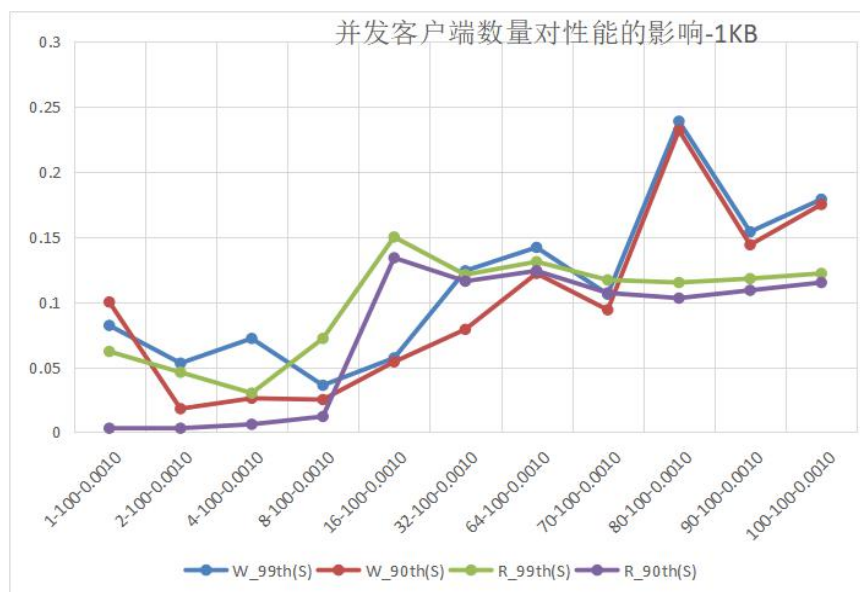


图 12 并发客户端数量对性能的影响-1KB

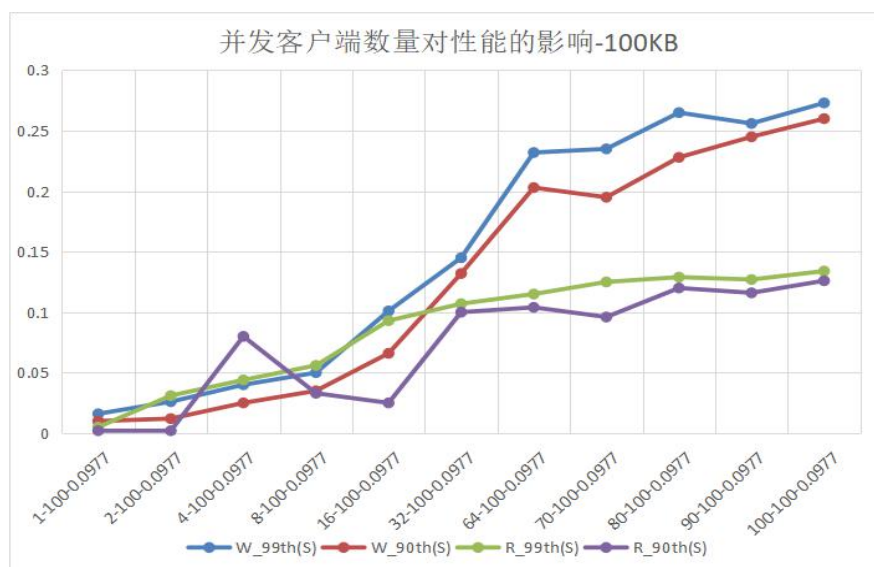


图 13 并发客户端数量对性能的影响-100KB

从图 12 和图 13 可看出，在两种不同条件下，随着并发客户端的增加，吞吐率也在增加。且相比 1KB 大小的测试对象，100KB 大小的测试对象有着更大的吞吐率。

但与此同时，随着并发客户端的增加，延迟也呈现增加的趋势。总结下来，minio 在并发客户端增加的情况下，虽然吞吐率在提高，但延迟也提高了，因此其总体性能是在下降的。

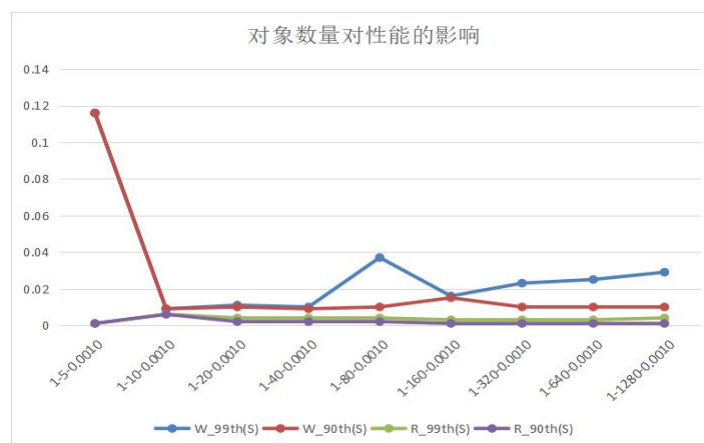


图 14 对象数量对性能的影响

从图 14 中可以看出，当并发客户端数量固定为 1 时，虽然对象数量从 5 到 1280 成倍的增加，但服务端的吞吐率和延迟变化并不大。这是因为因为此时只有一个客户端，所以其数据是串行到达，对象数量只是影响处理时间，对吞吐率和延迟无几乎没有影响。

传输速率方面：

依据图 10 分别作出对象大小、并发客户端数量、对象数量对读/写速率影响的折线图，分别如图 15、16、17 所示。

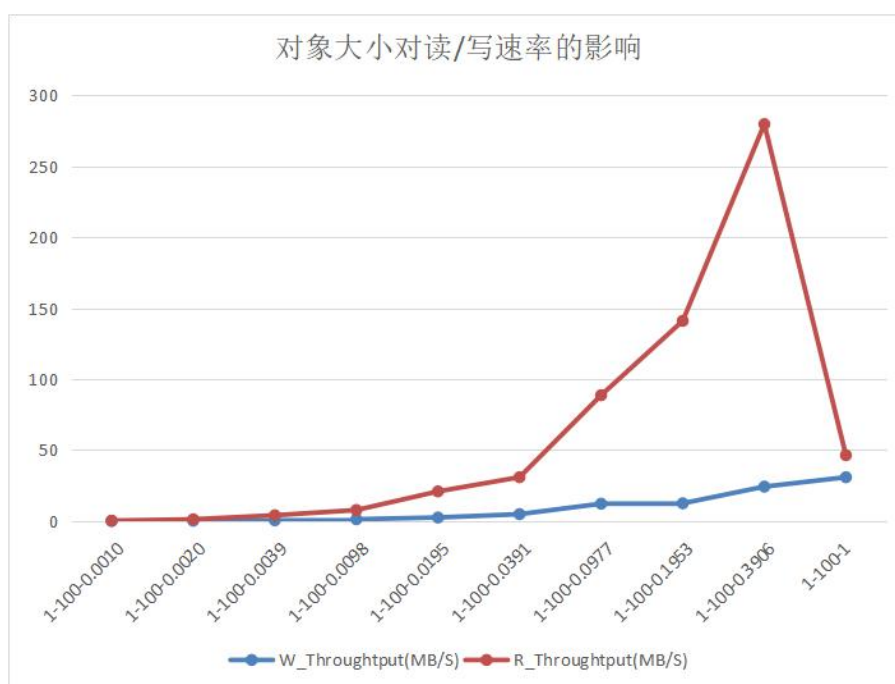


图 15 对象大小对传输速率的影响

从图 15 中可以看出，当对象大小增加时，写入速率与读取速率均整体增加，且可发现同等条件下读取速率要显著高于写入速率。而上图中的异常数据可能是网络波动导致的。

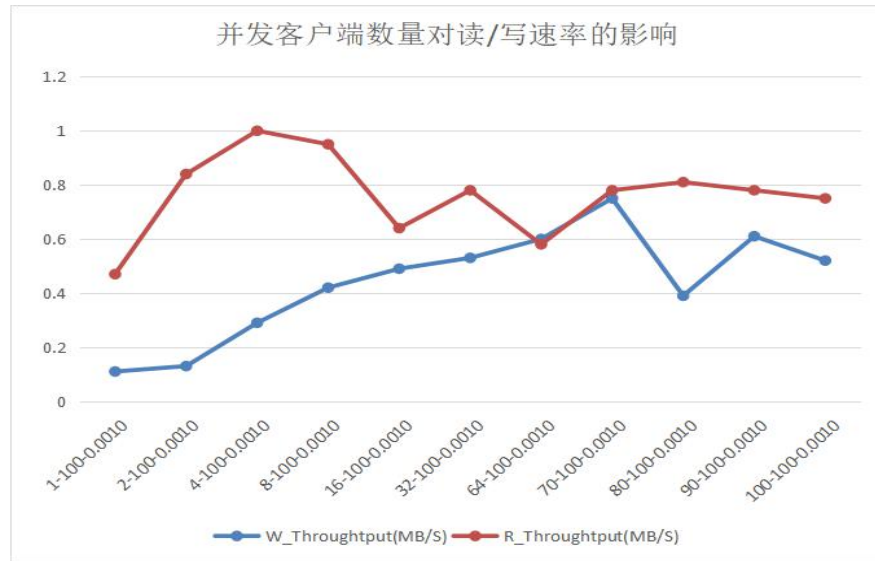


图 16 并发客户端数量对传输速率的影响

从图 16 中可看出，当并发客户端数量增加时，写入速率与读取速率均先增加然后整体趋于平稳。

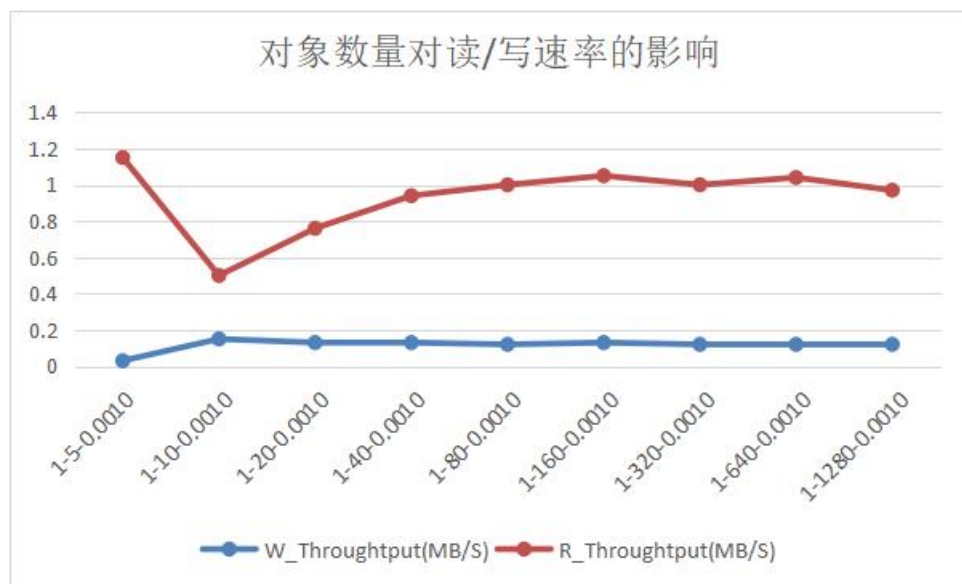


图 17 对象数量对传输速率的影响

从图 17 中可以看出，当对象数量增加时，写入速率与读取速率均先增加然后整体趋于平稳。且写入速率要显著高于读取速率。

6. 结论

依据以上三个测试的结果，可知 I/O 延迟主要的影响因素如下：

①对象的大小。对象大小越大，延迟也就越大。是因为对象越大时需要从磁盘中读取的数据越大，耗时也就越长。

②并发客户端数量，并发客户端越多，延迟越大。是因为过多的连接同时请求时会产生拥塞，需要排队处理请求。

六、实验总结

总的来说，本次实验整体上的难度并不算很大，老师所提供的实验指导更是进一步的为实验过程提供了大量的参考经验，整体的实验过程还是比较顺利的。

在这次实验中，在老师的指导下，我学习并接触了很多新的知识和技术，比如通过命令行使用 GIT，学习使用 minio 和 GO 的一些相关模块。尤其是在安装使用 GO 模块时，经常会发生找不到相应文件或文件夹的情况，后来才知道是 GOPATH 没有设置正确。经过查阅相关资料，才很好的解决了问题。

在实验过程中，老师提供的脚本给予了我很大的帮助。尤其是实验中所采用的 S3 Bench 并没有批量测试和输出结果文件的功能，利用 shell 脚本才实现了批量测试和终端输出的重定向。由于此前很少接触过脚本文件的编辑和操作，通过阅读老师提供的脚本的内容，也进一步学习了脚本文件的相关知识。

本次实验中老师和同学给我提供了莫大的帮助，同时我也参考了相关的资料。在实验过程中虽然经常遇到困难与挫折，但成功完成后也使我受益颇多。

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] URL: https://gitee.com/shi_zhan/obs-tutorial
- [5] URL: <https://github.com/cs-course/iot-storage-experiment-assignment-2019>
- [6] URL: https://blog.csdn.net/code_segment/article/details/78195630
- [7] URL: https://blog.csdn.net/szj_huhu/article/details/77541345
- [8] URL: https://blog.csdn.net/weixin_30412167/article/details/96778212