



2016 级

《物联网数据存储与管理》课程

实 验 报 告

姓 名 罗洋

学 号 U201614906

班 号 物联网 1601 班

日 期 2018.06.05

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.....	3
五、实验过程.....	3
六、实验总结.....	9
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

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：



图 1: 操作系统版本

java 版本:

```
pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$ java -version
java version "1.8.0_202"
Java(TM) SE Runtime Environment (build 1.8.0_202-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.202-b08, mixed mode)
```

图 2: java 版本

python 版本:

```
pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$ python --version
Python 2.7.15rc1
```

图 3: python 版本

go 版本:

```
pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$ go version
go version go1.10.4 linux/amd64
```

图 4: go 版本

四、实验内容

内容包含三个部分：由易到难分别使用不同的服务器和客户端

4.1 实验一

使用 minio 作为服务器，mc 作为客户端，使用测评工具进行测试存储性能。

4.2 实验二

使用 s3proxy 或 mock-s3 作为服务器，使用 s3cmd/osm/aws-shell 作为客户端，选择自己感兴趣的课题研究。

4.3 实验三

实验三使用 swift/Ceph 作为服务器，使用 awssdk/boto3 作为编程 api 自行进行编程实现。

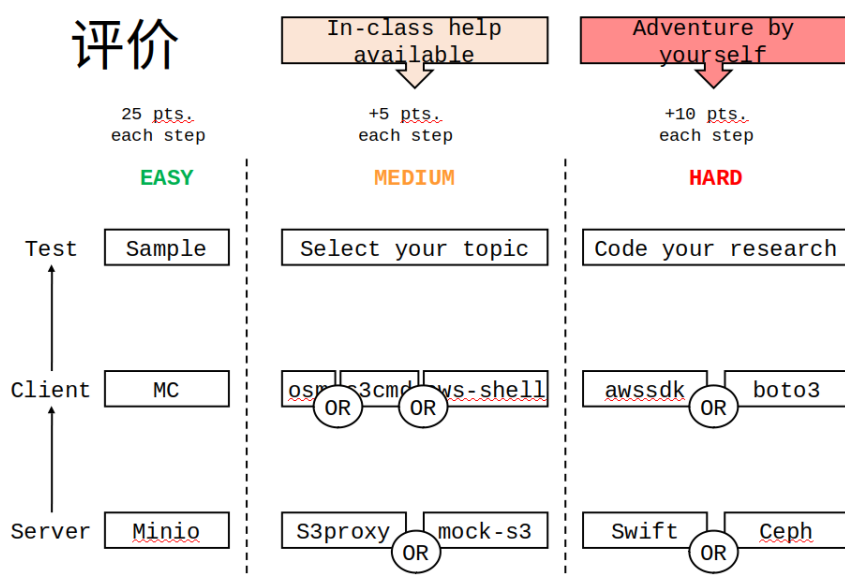


图 5: 实验内容

五、实验过程

5.1 实验一

(1) minio 作为服务器

1. 安装 ubuntu 系统下的 minio 软件。
2. 在 minio 运行程序所在的目录下用命令运行服务器程序

```
pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$ minio -C ./ server ./root

Endpoint: http://10.14.119.211:9000 http://172.17.0.1:9000 http://127.0.0.1:9000
AccessKey: V4MHN23U63R1WNPZRBJM
SecretKey: JUsViRUXFaZq1fDC90fcqr6UnS9HgGc6yHpuiNSb

Browser Access:
http://10.14.119.211:9000 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://10.14.119.211:9000 V4MHN23U63R1WNPZRBJM JUsViRUXFaZq1fDC90fcqr6UnS9HgGc6yHpuiNSb

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 6: 运行 minio 服务器端程序

在浏览器内输入地址 `http://127.0.0.1:9000`，使用 minio 生成的 accesskey 和 secretkey 登陆，可以看到服务器的存储内容；

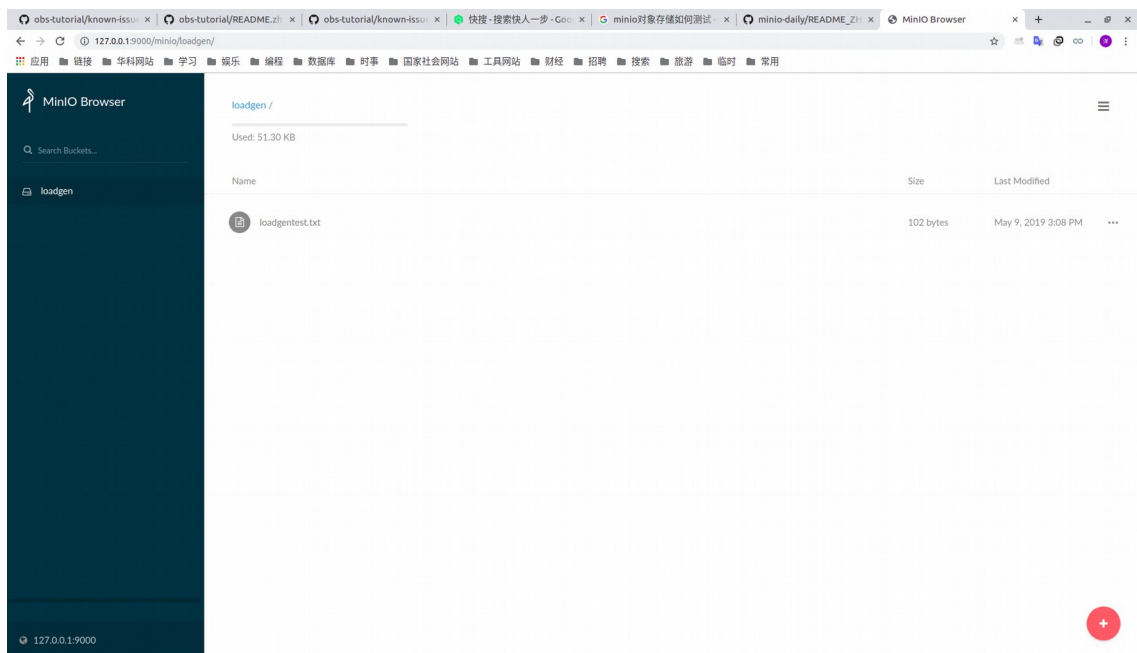
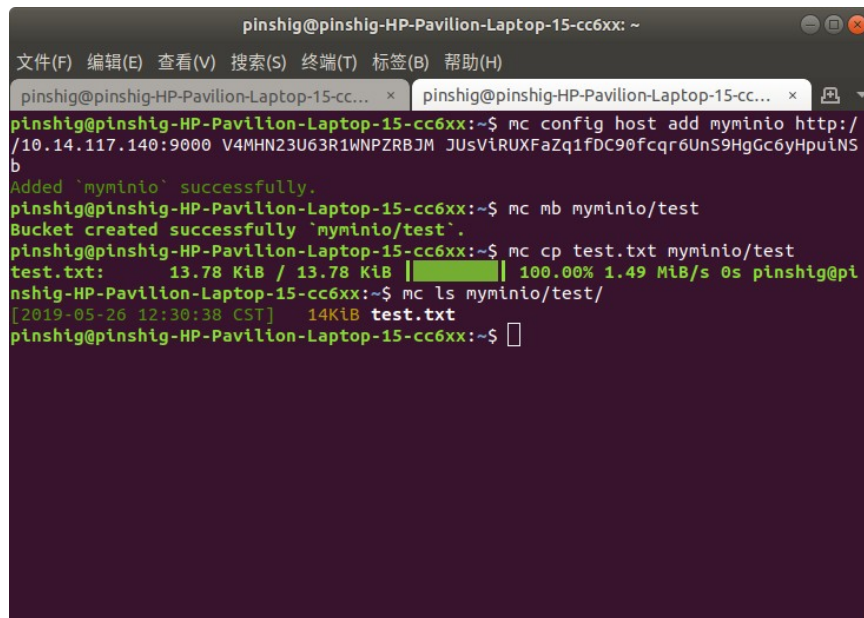


图 7: 登陆 minio 查看

(2) mc 作为客户端进行

1. 安装 ubuntu 系统下的 mc 软件。
2. 配置 mc

```
mc config host add myminio http://10.11.33.34:9000
V4MHN23U63R1WNPZRBJM JUsViRUXFaZq1fDC90fcqr6UnS9HgGc6yHpuiNSb
```

A terminal window titled 'pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx: ~' with a menu bar (文件(F), 编辑(E), 查看(V), 搜索(S), 终端(T), 标签(B), 帮助(H)). The terminal shows the following commands and output:

```
pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$ mc config host add myminio http://10.14.117.140:9000 V4MHN23U63R1WNPZRBJM JUsviRUXFaZq1fDC90fcqr6UnS9HgGc6yHpuINSb
Added 'myminio' successfully.
pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$ mc mb myminio/test
Bucket created successfully 'myminio/test'.
pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$ mc cp test.txt myminio/test
test.txt: 13.78 KiB / 13.78 KiB | ██████████ | 100.00% 1.49 MiB/s 0s pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$ mc ls myminio/test/
[2019-05-26 12:30:38 CST] 14KiB test.txt
pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$
```

图 8: 使用 *mc* 客户端程序对 *minio* 服务器进行操作

3. 在命令行中使用 *mc* 命令对 *minio* 服务器下的 *myminio* 主机进行创建 *bucket*，上传文件（需要先在本地创建一个需要上还有的文件），显示文件等操作。
- (3) 使用测评工具 *s3bench* 进行测试存储性能


```

pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$ s3bench -accessKey=V4MHN
=1024
Test parameters
endpoint(s): [http://172.17.0.1:9000]
bucket: loadgen
objectNamePrefix: loadgen
objectSize: 0.0010 MB
numClients: 10
numSamples: 100
verbose: %!d(bool=false)

Generating in-memory sample data... Done (13.189µs)

Running Write test...

Running Read test...

Test parameters
endpoint(s): [http://172.17.0.1:9000]
bucket: loadgen
objectNamePrefix: loadgen
objectSize: 0.0010 MB
numClients: 10
numSamples: 100
verbose: %!d(bool=false)

Results Summary for Write Operation(s)
Total Transferred: 0.098 MB
Total Throughput: 0.95 MB/s
Total Duration: 0.103 s
Number of Errors: 0
-----
Write times Max: 0.018 s
Write times 99th %ile: 0.018 s
Write times 90th %ile: 0.014 s
Write times 75th %ile: 0.010 s
Write times 50th %ile: 0.009 s
Write times 25th %ile: 0.008 s
Write times Min: 0.005 s

Results Summary for Read Operation(s)
Total Transferred: 0.098 MB
Total Throughput: 2.66 MB/s
Total Duration: 0.037 s
Number of Errors: 0
-----
Read times Max: 0.014 s
Read times 99th %ile: 0.014 s
Read times 90th %ile: 0.006 s
Read times 75th %ile: 0.005 s
Read times 50th %ile: 0.003 s
Read times 25th %ile: 0.002 s
Read times Min: 0.001 s

Cleaning up 100 objects...
Deleting a batch of 100 objects in range {0, 99}... Succeeded
Successfully deleted 100/100 objects in 14.44316ms
pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$

```

图 9: s3bench 测试存储性能

5.2 实验二

使用 s3proxy 或 mock-s3 作为服务器，使用 s3cmd/osm/aws-shell 作为客户端，选择自己感兴趣的课题研究。


- (1) 下载安装 mock-s3
- (2) 启动 mock-s3 服务器程序

```
pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$ mock_s3 -h
usage: mock_s3 [-h] [--hostname HOSTNAME] [--port PORT] [--root ROOT]
               [--pull-from-aws]

A Mock-S3 server.

optional arguments:
  -h, --help            show this help message and exit
  --hostname HOSTNAME   Hostname to listen on.
  --port PORT           Port to run server on.
  --root ROOT           Defaults to $HOME/s3store.
  --pull-from-aws       Pull non-existent keys from aws.
pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$ mock_s3 --hostname localhost --port 9100 --root ./s3store
Starting server, use <Ctrl-C> to stop
```

图 10: 打开 mock-s3 服务器



The screenshot shows a web browser address bar with the URL `127.0.0.1:9100`. The page content displays an XML document tree for a mock S3 bucket listing. The root element is `<ListAllMyBucketsResult xmlns="http://doc.s3.amazonaws.com/2006-03-01">`. It contains an `<Owner>` element with `<ID>123</ID>` and `<DisplayName>MockS3</DisplayName>`, and a `<Buckets>` element containing a single `<Bucket>` element. The `<Bucket>` element has `<Name>.s3cfg</Name>` and `<CreationDate>2019-05-09T23:54:59.000Z</CreationDate>`.

图 11: 登陆端口进行验证

- (3) 使用 aws-shell 作为客户端操作

```
pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$ aws-shell
aws> --endpoint-url http://127.0.0.1:9100 s3 ls s3://
2019-05-10 07:54:59 .s3cfg
2019-05-27 19:14:30 mybucket
aws> --endpoint-url http://127.0.0.1:9100 s3 mb s3://test
make_bucket: test
aws> --endpoint-url http://127.0.0.1:9100 s3 cp test.txt s3://test
The user-provided path test.txt does not exist.
aws> --endpoint-url http://127.0.0.1:9100 s3 cp test.txt s3://test
upload: ./test.txt to s3://test/test.txt
aws> --endpoint-url http://127.0.0.1:9100 s3 ls s3://test
2019-05-27 11:43:11      14113 test.txt
aws> --endpoint-url http://127.0.0.1:9100 s3 rm s3://test/test.txt
delete: s3://test/test.txt
```

图 12: 使用 aws-shell 作为客户端试验

- (4) 选择测评工具进行测评
- 这里使用 s3bench 进行测评，输入命令如下：

```
pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$ ./s3bench \
> -accessKey=hust \
> -accessSecret=hust2019 \
> -endpoint=http://127.0.0.1:9100 \
> -bucket=loadgen \
> -objectNamePrefix=loadgen \
> -numClients=10 \
> -numSamples=100 \
> -objectSize=1024
```

图 13: s3bench 测评命令

```
Test parameters
endpoint(s):      [http://127.0.0.1:9100]
bucket:          loadgen
objectNamePrefix: loadgen
objectSize:      0.0010 MB
numClients:      10
numSamples:      100
verbose:         %!d(bool=false)

Generating in-memory sample data... Done (12.329µs)

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://127.0.0.1:9100]
bucket:          loadgen
objectNamePrefix: loadgen
objectSize:      0.0010 MB
numClients:      10
numSamples:      100
verbose:         %!d(bool=false)

Results Summary for Write Operation(s)
Total Transferred: 0.098 MB
Total Throughput:  0.10 MB/s
Total Duration:   1.012 s
Number of Errors: 0
-----
Write times Max:    1.011 s
Write times 99th %ile: 1.011 s
Write times 90th %ile: 0.012 s
Write times 75th %ile: 0.011 s
Write times 50th %ile: 0.009 s
Write times 25th %ile: 0.008 s
Write times Min:    0.004 s

Results Summary for Read Operation(s)
Total Transferred: 0.098 MB
Total Throughput:  0.09 MB/s
Total Duration:   1.058 s
Number of Errors: 0
-----
Read times Max:     1.026 s
Read times 99th %ile: 1.026 s
Read times 90th %ile: 0.019 s
Read times 75th %ile: 0.010 s
Read times 50th %ile: 0.009 s
Read times 25th %ile: 0.008 s
Read times Min:     0.006 s

Cleaning up 100 objects...
Deleting a batch of 100 objects in range {0, 99}... Succeeded
Successfully deleted 100/100 objects in 9.573439ms
pinshig@pinshig-HP-Pavilion-Laptop-15-cc6xx:~$
```

图 14: 测评结果

(5) 选择主题进行探索

选择主题：如果客户端爆满会怎样？

(mock_s3 作为服务器，s3bench 作为测评工具，测评其他基本信息如图 13)

测评结果如下表：

客户端数量	write		read	
	吞吐量	持续时间	吞吐量	持续时间
10	0.09 MB/s	1.036 s	0.09 MB/s	1.055 s
30	0.08 MB/s	1.257 s	0.05 MB/s	1.980 s
50	0.03 MB/s	3.034 s	0.03 MB/s	3.053 s
70	0.02 MB/s	4.345 s	0.01 MB/s	7.561 s
75	0.02 MB/s	5.321 s	0.01 MB/s	14.342 s
77	0.01 MB/s	9.768 s	0.00 MB/s	27.400 s
78	0.02 MB/s	4.329 s	0.00 MB/s	53.930 s
80	0.02 MB/s	4.642 s	0.00 MB/s	108.433 s
90	0.03 MB/s	3.279 s	0.00 MB/s	127.738 s
100	0.01 MB/s	9.844 s	0.00 MB/s	155.183 s

从表中可以看出客户端数量在 77 左右开始爆满，爆满之后对写数据操作的影响不大，但是对读出数据的影响比较剧烈每增加一个客户端所消耗的时间几乎以成倍速率增长。

5.3 实验三

实验三使用 swift/Ceph 作为服务器，使用 awssdk/boto3 作为编程 api 自行进行编程实现。

由于安装 ceph 和 swift 遇到一些困难，所没能完成该部分实验。

六、实验总结

通过这次实验，学会了搭建 minio、mock-s3 和 proxy 服务器，学会了通过网页查看存储内容，学会了使用 mc、aws-shell 客户端程序上传下载数据，学会了使用 s3bench 测试对象存储性能。另外在第一节课上认真地学习了 github 和 git 的使用，真正明白了 git 的结构层次，明白了仓库、提交、克隆等概念的真正含义，为今后的学习工作打下了一个好的基础。在这次实验里，最大的问题是 cosbench 在我的电脑上无法运行，换了多种版本和使用容器运行都无法有效使用 cosbench 测评，没能更直观地对一些参数的修改进行对比，但是 s3bench 的使用也让我对对象存储测评的有了很好的了解。另外在搭建 ceph 和 swift 服务器上，没有找到比较容易的教程，在搭建的过程中屡屡出错，所以最后放弃了这部分的内容。这次的实验还是让我获得很多收获的，让我了解了当下热门的对象存储，让我理解了对对象存储的服务器和客户端之间的交互，希望今后能不断学习知识，接触更多前沿

的计算机技术。

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.