2016 级

《物联网数据存储与管理》课程

实 验 报 告

姓　名　赵文浩

学　号　U201614895

班　号　物联网 1601 班

日　期　2019.05.26

目　　录

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（Object-based Storage)是一种新的网络存储架构，基于对象存储技术的设备就是对象存储设备（Object-based Storage Device）简称 OSD。1999 年成立的全球网络存储工业协会（SNIA）的对象存储设备工作组发布了 ANSI 的 X3T10 标准。总体上来讲，对象存储综合了 NAS 和 SAN 的优点，同时具有 SAN 的高速直接访问和 NAS 的分布式数据共享等优势，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。

对象存储比传统的文件系统存储在规模上要大得多，这是由于前者比后者着实要简单得多。对象存储系统并非将文件组织成一个目录层次结构，而是在一个扁平化的容器组织中存储文件(在亚马逊 S3 系统中被称作“桶”)，并使用唯一的 ID(在 S3 中被称作“关键字”)来检索它们。其结果是对象存储系统相比文件系统需要更少的元数据来存储和访问文件，并且它们还减少了因存储元数据而产生的管理文件元数据的开销。这意味着对象存储能够通过增加节点而近乎无限制地扩展规模。

对象存储自出现以后一直随着互联网时代的需求不断发展，目前已处于第三次浪潮，已有许多比较成熟的开源项目和产品，例如出现了大量的开源项目和产品，例如：OpenStack Swift、Skylable、Manta(来自 Joyent 公司)、Minio、 OpenIO、Ambry(来自 LinkedIn 公司)、Torus、Microsoft Azure、Google Cloud Storage 等。

三、实验环境

硬件环境	处理器	Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz
	内存容量	8.0GB
	硬盘	1、PH4-CE120 (120GB) 2、ST1000LM024

		HN-M101MBB (1.0TB)
	主板	X556UB (1.0)
宿主机环境	操作系统	Windows 10 64 位
	虚拟机软件	VirtualBox 5.2.22 r126460 (Qt5.6.2)
虚拟机资源配置	内存分配	4537MB
	虚拟磁盘分配	150GB
	处理器数量	2
	处理器运行峰值	100%
虚拟机系统	系统版本	Ubuntu 16.04.4 LTS
	内核版本	4.13.0-39-generic
	java 环境	java version "1.8.0_211" Java(TM) SE Runtime Environment (build 1.8.0_211-b12) Java HotSpot(TM) 64-Bit Server VM (build 25.211-b12, mixed mode)
	python 环境	Python 2.7.12
	go 环境	go version go1.12.5 linux/amd64

四、实验内容

本次实验主要是对对象存储系统进行了解、实践和分析，采用目前常用的 minio/mc，s3proy/s3cmd，OpenStack Swift 等配置服务端和客户端，并通过 cosbench，s3bench 等测试工具对服务端进行测试。

在本次实验中我选择 minio 服务端与 s3bench 测试、OpenStack Swift 服务端与

cosbench 测试进行实践，并通过测试结果来对对象存储系统进行分析。

五、实验过程

5.1 minio 服务端与 s3bench 测试

1. 配置 minio 服务端

在终端执行如下命令，下载 minio 客户端并设置执行权限。

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
chmod +x minio
```

然后运行服务端，运行结果如图 1 所示。可以看到随机生成了 AccessKey 和 SecretKey。

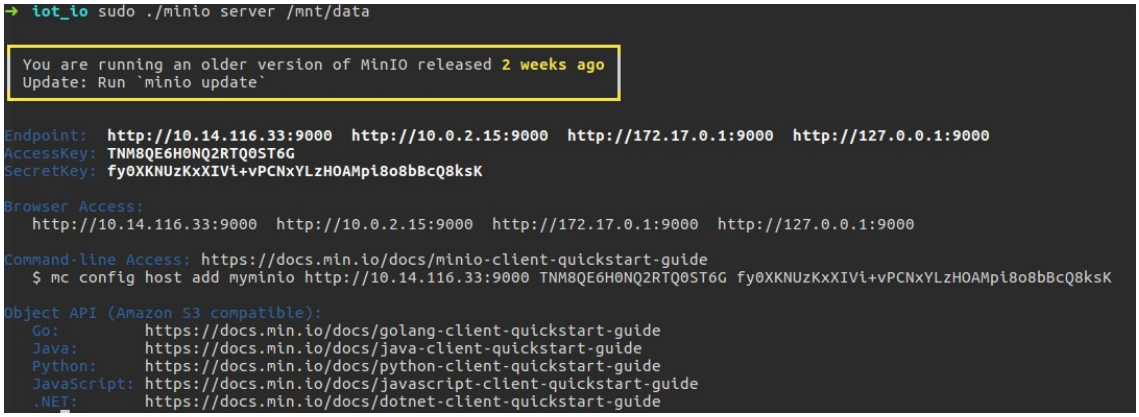

图 1 运行 minio

在浏览器中输入地址 http://10.14.116.33:9000 即可连接到 minio server，输入随机生成的密钥即可登陆，登陆后如图 2 所示。可自行创建 bucket 并上传文件，并且可以链接的方式分享已上传的文件，同时可以设置分享文件的有效期。

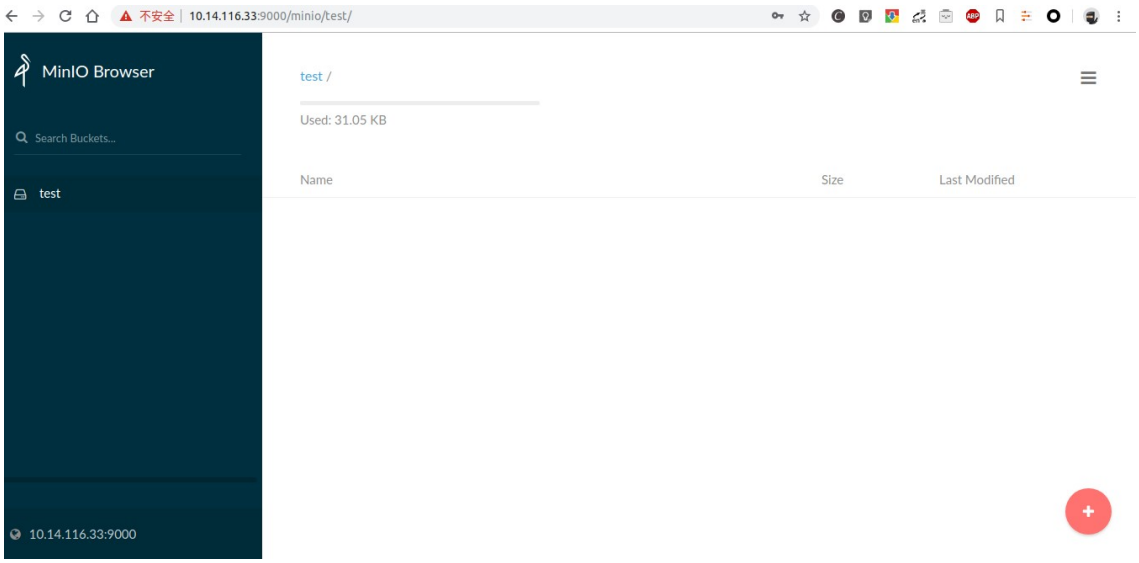

图 2 minio 服务端

2. 配置 minio 客户端 mc

在终端执行如下命令，下载 mc 客户端并设置执行权限。

```
wget https://dl.min.io/client/mc/release/linux-amd64/mc
chmod +x mc
```

然后根据服务端信息对客户端进行配置，如图 3 所示。

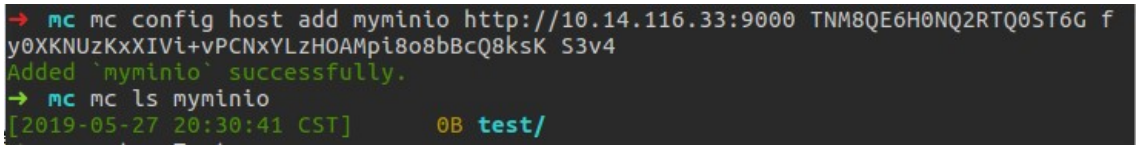

图 3 配置 mc 客户端

然后利用客户端进行测试，上传一个文件，结果如图 4 所示。

图 4 上传文件测试

3. 配置 s3bench 并进行测试。

首先配置好 go 语言环境，将 go 语言环境变量添加到系统环境变量中去。配置好后检查，如图 5 所示。

图 5 检查 go 环境

然后利用 go 执行如下命令安装 s3bench。

```
go get -u github.com/igneous-systems/s3bench
```

然后写一个脚本文件，改变 ObjectSize 的大小，依次进行测试，并将测试结果重定向到日志文件中，方便查看。

测试命令为：

```
echo "start---1kb" >> log
./s3bench -accessKey=TNM8QE6H0NQ2RTQ0ST6G -accessSecret=fy0XKNUzKxXIVi+vPCNxYLzHOAMpi8o8bBcQ8ksK -bucket=test -endpoint=http://10.14.116.33:9000 -numClients=2 -numSamples=10 -objectNamePrefix=minio -objectSize=1024 >> log
```

```
    echo "start---512kb" >> log
    ./s3bench -accessKey=TNM8QE6H0NQ2RTQ0ST6G
-accessSecret=fy0XKNUzKxXIVi+vPCNxYLzHOAMpi8o8bBcQ8ksK -bucket=test
-endpoint=http://10.14.116.33:9000 -numClients=2 -numSamples=10
-objectNamePrefix=minio -objectSize=524288 >> log
    echo "start---1Mb" >> log
    ./s3bench -accessKey=TNM8QE6H0NQ2RTQ0ST6G
-accessSecret=fy0XKNUzKxXIVi+vPCNxYLzHOAMpi8o8bBcQ8ksK -bucket=test
-endpoint=http://10.14.116.33:9000 -numClients=2 -numSamples=10
-objectNamePrefix=minio -objectSize=1048576 >> log
    echo "start---10MB" >> log
    ./s3bench -accessKey=TNM8QE6H0NQ2RTQ0ST6G
-accessSecret=fy0XKNUzKxXIVi+vPCNxYLzHOAMpi8o8bBcQ8ksK -bucket=test
-endpoint=http://10.14.116.33:9000 -numClients=2 -numSamples=10
-objectNamePrefix=minio -objectSize=10485760 >> log
    echo "start---100MB" >> log
    ./s3bench -accessKey=TNM8QE6H0NQ2RTQ0ST6G
-accessSecret=fy0XKNUzKxXIVi+vPCNxYLzHOAMpi8o8bBcQ8ksK -bucket=test
-endpoint=http://10.14.116.33:9000 -numClients=2 -numSamples=10
-objectNamePrefix=minio -objectSize=104857600 >> log
    echo "start---300MB" >> log
    ./s3bench -accessKey=TNM8QE6H0NQ2RTQ0ST6G
-accessSecret=fy0XKNUzKxXIVi+vPCNxYLzHOAMpi8o8bBcQ8ksK -bucket=test
-endpoint=http://10.14.116.33:9000 -numClients=2 -numSamples=10
-objectNamePrefix=minio -objectSize=314572800 >> log
    echo "start---500MB" >> log
    ./s3bench -accessKey=TNM8QE6H0NQ2RTQ0ST6G
-accessSecret=fy0XKNUzKxXIVi+vPCNxYLzHOAMpi8o8bBcQ8ksK -bucket=test
-endpoint=http://10.14.116.33:9000 -numClients=2 -numSamples=10
-objectNamePrefix=minio -objectSize=524288000 >> log
    echo "start---10MB" >> log
    ./s3bench -accessKey=TNM8QE6H0NQ2RTQ0ST6G
-accessSecret=fy0XKNUzKxXIVi+vPCNxYLzHOAMpi8o8bBcQ8ksK -bucket=test
-endpoint=http://10.14.116.33:9000 -numClients=2 -numSamples=10
-objectNamePrefix=minio -objectSize=1073741820 >> log
```

其中客户端数量为 2，sample 数量为 10，objectSize 依次递增，从 1kB 增加到 1GB。经测试并统计测试结果，结果如图 6 所示。

ObjectSize	Total Transferred	Write	Read
0.001MB	0.01MB	0.26 MB/s	0.88 MB/s
0.5MB	5MB	35.1 MB/s	141.35 MB/s
1MB	10MB	61.53 MB/s	625.81 MB/s
10MB	100MB	172.62 MB/s	797.97 MB/s
100MB	1000MB	67.61 MB/s	59.96 MB/s
300MB	3000MB	68.3 MB/s	65.04 MB/s
500MB	5000MB	61.11 MB/s	45.66 MB/s
1024MB	10240MB	23.82 MB/s	38.9 MB/s

图 6 s3bench 测试结果统计

由测试结果可知，随着 objectSize 增加，读写速度先增加后减小。在 ObjectSize 从 0.001Mb 增加到 10MB 时读写速度一直增加，且读的速度远大于写的速度。读写速度增加是因为当文件很小时，系统用于读写的时间占比极小，而准备工作与其他操作的时间开销站主要部分，但这些时间都算作读写时间，造成大量的假延迟。而随着读写规模增大，读写的时间开销占主导地位，所以测试得到的读写速度就原来越接近真实磁盘读写的速度。读的速度远大于写的速度，这是由磁盘的物理特性导致的。

在 ObjectSize 从 10Mb 增加到 100MB 时读的速度剧烈降低，写的速度也降低明显。而随着读写规模的增加，读写速度越来越慢。经分析推测，这是由于在测试时多个 sample 和客户端并行发送请求和传输数据，且所有服务端和客户端都是在一台电脑上运行的，且电脑的内存和磁盘资源有限，使得服务器拥塞。当内存无法同时存放这些数据时就由操作系统将数据存放到虚存即磁盘中，当用到时再从磁盘中读出。而且当数据越大，该操作越多，占用的时间越长，所以使得总速率才越来越低。

5.2 OpenStack Swift 服务端与 cosbench 测试

首先使用 git 将 OpenStack Swift 从 github 上克隆下来，然后执行以下指令构建 docker 镜像和准备 datavolume。

```
docker build -t openstack-swift-docker .
docker run -v /srv --name SWIFT_DATA busybox
```

然后在 docker 中运行 OpenStack Swift，结果如图 7 所示。

```
→ ~ sudo docker run -d -p 12345:8080 --volumes-from SWIFT_DATA -t openstack-swi
ft-docker
[sudo] loin 的密码:
fa299b7b46757ed6cc0af01e14643afd56b0d49a253f5ec39779a765836eabae
```

图 7 懂 docker 运行 OpenStack Swift

然后运行 cosbench，如图 8 所示。

```
Starting    cosbench-driver_0.4.2    [OK]
Starting    cosbench-driver-web_0.4.2    [OK]
Successfully started cosbench driver!
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 !!!
-----------------------------------------------


=========================================================

Launching osgi framwork ...
Successfully launched osgi framework!
Booting cosbench controller ...
....
Starting    cosbench-log_0.4.2    [OK]
.
Starting    cosbench-tomcat_0.4.2    [OK]
Starting    cosbench-config_0.4.2    [OK]
Starting    cosbench-core_0.4.2    [OK]
Starting    cosbench-core-web_0.4.2    [OK]
Starting    cosbench-controller_0.4.2    [OK]
Starting    cosbench-controller-web_0.4.2    [OK]
Successfully started cosbench controller!
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 !!!
-----------------------------------------------
➜  cos
```

图 8 启动 cosbench

在 github 上下载实验提供的 cosbench 配置文件 workload-example.xml，并修改其配置，使其能对 OpenStack Swift 进行测试，修改后内容如图 9 所示。

```
<?xml version="1.0" encoding="UTF-8" ?>

<workload name="swift sample" description="sample benchmark">

  <auth type="swauth" config="username=test:tester;password=testing;url=http://127.0.0.1:12345/auth/v1.0"/>
    <storage type="swift" config=""/>

  <workflow>

    <workstage name="init">
      <work type="init" workers="1" config="cprefix=s3obstest;containers=r(1,8)" />
    </workstage>
```

图 9 修改 workload 配置

然后再浏览器中打开 http://127.0.0.1:19088/controller/index.html，选择“submit new

workloads”，选择刚配置好的文件上传，即可自动进行评测。测试方案如图 10 所示。

ID	Name	Works	Workers	Op-Info	State	Link
w18-s1-init	init	1 wks	1 wkrs	init	completed	view details
w18-s2-prepare	prepare	8 wks	64 wkrs	prepare	completed	view details
w18-s3-8kb	8kb	1 wks	8 wkrs	read, write	completed	view details
w18-s4-16kb	16kb	1 wks	8 wkrs	read, write	completed	view details
w18-s5-32kb	32kb	1 wks	4 wkrs	read, write	completed	view details
w18-s6-64kb	64kb	1 wks	4 wkrs	read, write	completed	view details
w18-s7-128kb	128kb	1 wks	1 wkrs	read, write	completed	view details
w18-s8-256kb	256kb	1 wks	1 wkrs	read, write	completed	view details
w18-s9-512kb	512kb	1 wks	1 wkrs	read, write	completed	view details
w18-s10-1mb	1mb	1 wks	1 wkrs	read, write	completed	view details
w18-s11-cleanup	cleanup	1 wks	1 wkrs	cleanup	completed	view details
w18-s12-dispose	dispose	1 wks	1 wkrs	dispose	completed	view details

图 10 cosbench 测试方案

评测结束后整体报告如图 11 所示。

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	1095.25 ms	1090.62 ms	7.68 op/s	61.43 KB/S	100%
op2: prepare -write	8 ops	128 KB	391.12 ms	388.38 ms	22.02 op/s	352.36 KB/S	100%
op3: prepare -write	8 ops	256 KB	886.12 ms	879.88 ms	9.66 op/s	308.98 KB/S	100%
op4: prepare -write	8 ops	512 KB	950.12 ms	939.75 ms	8.49 op/s	543.53 KB/S	100%
op5: prepare -write	8 ops	1.02 MB	1100.25 ms	1077.5 ms	7.31 op/s	935.77 KB/S	100%
op6: prepare -write	8 ops	2.05 MB	1194.5 ms	1159.5 ms	6.66 op/s	1.7 MB/S	100%
op7: prepare -write	8 ops	4.1 MB	1306.88 ms	1262.5 ms	6.72 op/s	3.44 MB/S	100%
op8: prepare -write	8 ops	8 MB	969 ms	758.25 ms	8.88 op/s	8.88 MB/S	100%
op1: read	2.14 kops	17.14 MB	75.88 ms	75.85 ms	71.51 op/s	572.11 KB/S	100%
op2: write	598 ops	4.78 MB	128.6 ms	128.23 ms	19.96 op/s	159.64 KB/S	100%
op1: read	2.26 kops	36.08 MB	75.79 ms	75.78 ms	75.41 op/s	1.21 MB/S	100%
op2: write	558 ops	8.93 MB	122 ms	121.44 ms	18.66 op/s	298.55 KB/S	100%
op1: read	2.15 kops	68.7 MB	38.79 ms	38.77 ms	71.6 op/s	2.29 MB/S	100%
op2: write	522 ops	16.7 MB	70.01 ms	69.11 ms	17.41 op/s	557.09 KB/S	100%
op1: read	2.15 kops	137.73 MB	37.36 ms	37.32 ms	71.95 op/s	4.6 MB/S	100%
op2: write	561 ops	35.9 MB	69.68 ms	68.93 ms	18.75 op/s	1.2 MB/S	100%
op1: read	739 ops	94.59 MB	29 ms	28.4 ms	24.63 op/s	3.15 MB/S	100%
op2: write	166 ops	21.25 MB	51.4 ms	48.67 ms	5.53 op/s	708.31 KB/S	100%
op1: read	1.01 kops	258.82 MB	20.13 ms	18.02 ms	33.71 op/s	8.63 MB/S	100%
op2: write	217 ops	55.55 MB	44.23 ms	39.88 ms	7.24 op/s	1.85 MB/S	100%
op1: read	642 ops	328.7 MB	30.41 ms	24.65 ms	21.4 op/s	10.96 MB/S	100%
op2: write	186 ops	95.23 MB	56.16 ms	48.91 ms	6.2 op/s	3.17 MB/S	100%
op1: read	538 ops	538 MB	39.43 ms	26.81 ms	17.95 op/s	17.95 MB/S	100%
op2: write	129 ops	129 MB	67.78 ms	54.34 ms	4.3 op/s	4.3 MB/S	100%
op1: cleanup - delete	128 ops	0 B	31.88 ms	31.88 ms	31.3 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

图 11 测试结果概览

其中某一个具体读写测试图如图 12 所示。

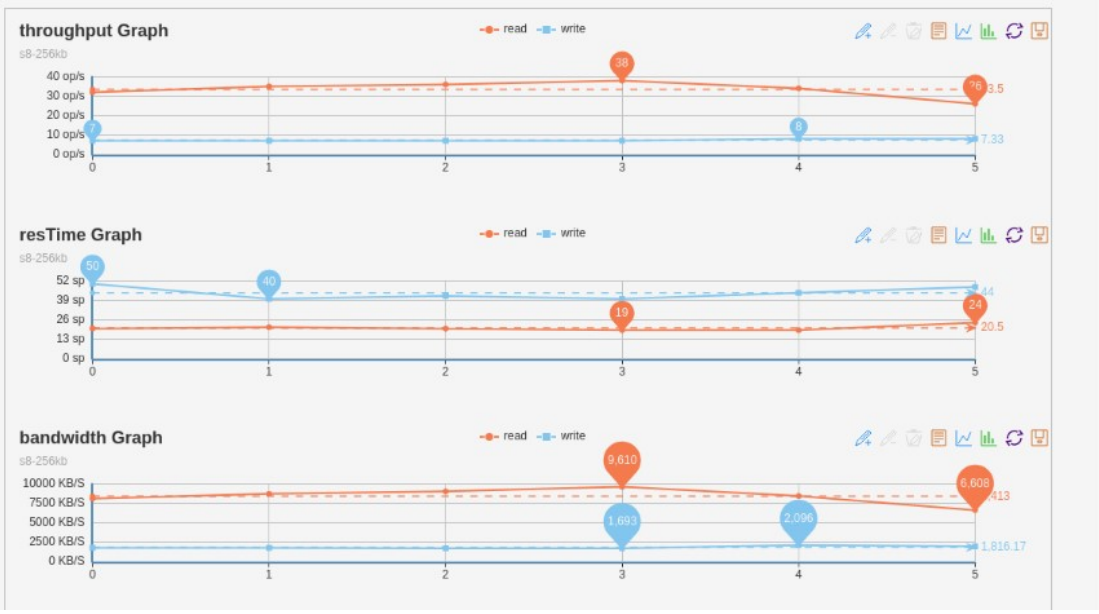

图 12 块大小为 256kb 的具体测试图表

由测试方案和测试结果可知当 worker 增加一倍时，其带宽也增加一倍。随着块大小增加，其带宽也增加，读写速度越来越快。而且读的速度明显大于写的速度。当数据块较小时，服务器读写操作中的其他操作占据了主要延迟，即出现假延迟，使得测试得到的读写速度较慢。而随着数据块增大，对磁盘的 I/O 逐渐成为主要延迟，所以测得的读写速度逐渐增加。

由于测试中读写的数据块并不大，并且 worker 较小，服务器资源充足，处理能力与带宽能力都能充分满足实验需求，所以测试结果与期待的一样，成功率也都为 100%，但是当服务器超荷负载时，其性能就会明显降低。

六、实验总结

本次实验对对象存储系统有了一个基本的认识，并进行搭建和测试。对象存储与其他存储方式相比，具有更好的扩展性，可将多台服务器看做多个存储节点，配置服务端也比较方便，然后再用其他的服务器作为控制管理节点，负责调度调整存储节点。当用户请求一个文件时，会先由控制节点查询存储对象的属性得知该文件在哪些存储节点上，然后存储节点可并行的传输数据，使得用户对文件的操作速度成倍增加，并且存储服务节点越多，存取的速度越快。除此之外，对象存储系统易于文件的共享。

经过本次实验，还对 git 的使用更加熟练，初次使用了 docker，认识到容器的便捷性。除此之外，还配置了各种实验环境，锻炼了自己的动手能力。

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.