2019 级

《物联网数据存储与管理》课程

实 验 报 告

姓　名 周飞

学　号 U201915183

班　号 计算机 1908 班

日　期 2022.04.14

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，分析性能问题，架设应用实践。

二、实验背景

对象存是一种将数据作为对象进行管理的计算机数据存储体系结构。对象存储，将数据存储作为不同单元（对象）进行管理和操作。这些对象保存在一个单独的仓库中，并没有根植于其他文件夹内的文件中。相反，对象存储组合了构成文件的数据片段，将其所有相关元数据添加到该文件，并附加自定义标识符。

对象存储，主要操作对象是对象。存储协议是 S3、swift 等。主要的接口命令有 PUT/GET/DELETE/POST 等。

对象存储呈现出来的是一个 bucket，可以往 bucket 里面放对象。这个对象包括三个部分：Key、Data、Metadata。 key 是该对象的全局唯一标识符（UID）。Key 是用于检索对象，服务器和用户不需要知道数据的物理地址，也能通过它找到对象。这种方法极大地简化了数据存储。Data 也就是用户数据本体。MetadataM 叫做元数据。

对象存储的架构主要包含 OSD 对象存储设备、MDS 元数据服务器、Client 客户端三个部分。 OSD 对象存储设备是对象存储的核心，具有 CPU、内存、网络和磁盘系统。它的主要功能是存储数据。同时还能优化数据分布支持数据预读取，提升磁盘性能。 MDS 元数据服务器控制 client 和 OSD 的交互，管理限额控制、目录和文件的创建和删除。 Client 客户端则提供文件系统接口，方便外部访问。

三、实验环境

实验环境见表 1

操作系统	Windows 10
CPU	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
内存	16.0 GB
PYTHON 版本	Python 3.9.10
客户端	Mock-s3
服务端	osm
测试工具	S3 Bench

表 1 实验环境

四、实验内容

4.1 对象存储技术实践

（编号说明实验内容，按照所观测功能、指标拟定实验案例）

1．选择服务端模拟服务程序 mock-s3。 Git clone mock-s3 仓库并使用 python 进行安装。 运行 mock-s3，搭建好服务器环境。

2. 安装对象存储系统客户端 osm.exe，使用脚本程序 config-osm.cmd 进行配置。 使用 osm 进行桶的创建、删除、查询、文件的上传等工作。

4.2 对象存储性能分析

1. 安装 s3 bench 作为性能测试工具。
2. 将 objSize 作为因变量观察其对读写带宽和延迟的影响。
3. 将 client_num 作为因变量观察其对吞吐率和延迟的影响。
4. 整理数据并绘制表格和折线图。
5. 分析。

五、实验过程

5.1 对象存储技术实践

(1) 对象存储系统服务端 mock_s3 的安装配置和启动。

Clone mock_s3 仓库。 使用 python3 setup.py install 命令进行安装 。

使用 python3 mock_s3/main.py --hostname 0.0.0.0 --port 9000 --root ./root 命令进行启动。

在浏览器输入 127.0.0.1：9000 能够成功访问并在服务端输出状态信息如图 5.1.1

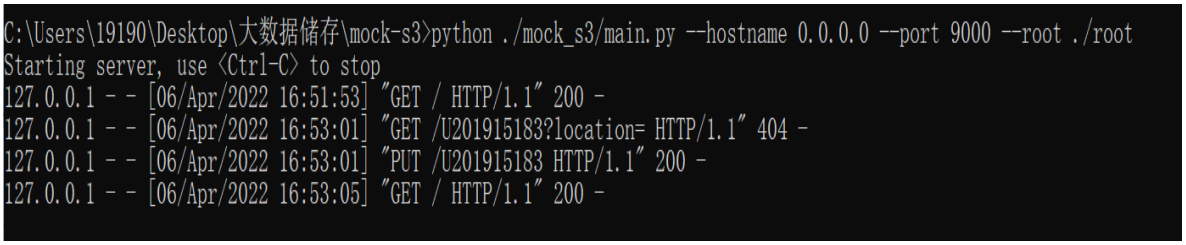


图 5.1.1 mock_s3 启动状态。

（2）对象存储系统客户端 osm 的安装、配置和基本操作。

直接下载 osm.exe 可执行程序。将 config-osm.cmd 脚本文件放在与 osm 同一个目录，不修改脚本文件的内容直接运行。 运行结束后可以在终端中找到并使用 osm 命令。

osm 初步使用如图 5.1.2 和 5.1.3

```
C:\Users\19190\Desktop\大数据储存\osm>osm lc
Found 0 container in

C:\Users\19190\Desktop\大数据储存\osm>osm mc U201915183
Successfully created container U201915183

C:\Users\19190\Desktop\大数据储存\osm>osm lc
U201915183
Found 1 container in
```

图 5.1.2　osm 基本操作

```
C:\Users\19190\Desktop\大数据储存\osm>osm push -c U201915183 ./img/s.jpg s.jpg
Successfully pushed item s.jpg

C:\Users\19190\Desktop\大数据储存\osm>osm ls U201915183
s.jpg
Found 1 item in container U201915183

C:\Users\19190\Desktop\大数据储存\osm>
```

图 5.1.3 osm 文件上传

在 浏览器访问 127.0.0.1：9000 能够看到创建的桶如图 5.1.4

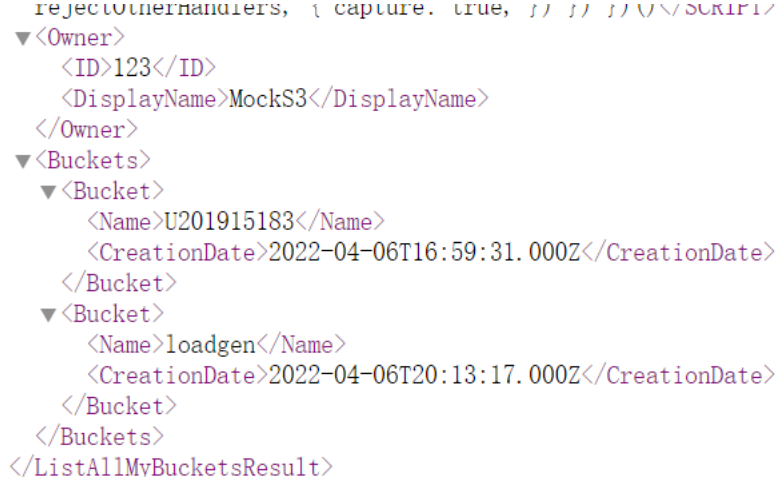


图 5.1.4 浏览器访问

服务段能够看到文件的成功上传并输出状态信息如图 5.1.5

```
127.0.0.1 - - [07/Apr/2022 09:30:17] "GET /U201915183?location= HTTP/1.1" 200 -
127.0.0.1 - - [07/Apr/2022 09:30:38] "GET /U201915183?location= HTTP/1.1" 200 -
127.0.0.1 - - [07/Apr/2022 09:30:38] "PUT /U201915183/x.jpg HTTP/1.1" 200 -
127.0.0.1 - - [07/Apr/2022 09:30:38] "HEAD /U201915183/x.jpg HTTP/1.1" 200 -
```

图 5.1.5 上传文件状态信息

5.2 对象存储性能分析

(1) 直接下载可执行文件 s3_bench.exe 和 脚本文件 run-s3bench.cmd。 并将两个文件放在同一个目录下。

(2) 服务端保持运行状态，不修改 run-s3bench.cmd 的内容，首次运行结果如图 5.2.1

```
Test parameters
endpoint(s):      [http://127.0.0.1:9000]
bucket:           loadgen
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:  0.48 MB/s
Total Duration:    0.523 s
Number of Errors:  0
-----------------------------------
Write times Max:       0.522 s
Write times 99th %ile: 0.013 s
Write times 90th %ile: 0.010 s
Write times 75th %ile: 0.008 s
Write times 50th %ile: 0.007 s
Write times 25th %ile: 0.006 s
Write times Min:       0.003 s


Results Summary for Read Operation(s)
Total Transferred: 0.250 MB
Total Throughput:  0.48 MB/s
Total Duration:    0.522 s
Number of Errors:  0
-----------------------------------
Read times Max:       0.522 s
Read times 99th %ile: 0.520 s
Read times 90th %ile: 0.014 s
Read times 75th %ile: 0.011 s
Read times 50th %ile: 0.008 s
Read times 25th %ile: 0.006 s
Read times Min:       0.005 s
```

图 5.2.1 run-s3bench.cmd 示例程序运行结果

观察服务端输出如图 5.2.2

```
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen239 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen240 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen242 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen241 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen243 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen244 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen245 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen246 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen247 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen248 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen249 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen250 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen251 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen252 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen253 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen254 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen255 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen6 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "PUT /loadgen/loadgen1 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "GET /loadgen/loadgen3 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "GET /loadgen/loadgen1 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "GET /loadgen/loadgen0 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "GET /loadgen/loadgen5 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "GET /loadgen/loadgen2 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "GET /loadgen/loadgen8 HTTP/1.1" 200 -
127.0.0.1 - - [06/Apr/2022 17:43:44] "GET /loadgen/loadgen9 HTTP/1.1" 200 -
```

图 5.2.2 run-s3bench 示例程序服务端输出

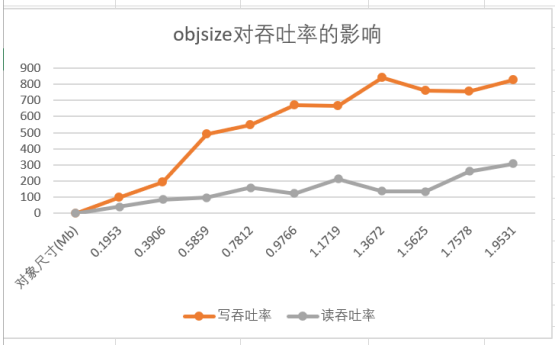
（3） 将 objsize 作为变量

客户端数量为 8，样本数量 256 保持不变。

设置循环每次均匀增加 objsize 的值，每次增加 10240 个字节并将结果重定向到文本文件。 从文本文件中提取数据并绘制 excel 表格和折线图。

图表如图 5.2.3

对象尺寸(Mb)	客户端数量(个)	写吞吐率(MB/S)	写99th延迟(s)	写90th延迟(S)	读吞吐率(MB/S)	读99th延迟(s)	读90th延迟(s)
0.1953	8	96.78	0.106	0.007	39.88	0.533	0.03
0.3906	8	191.71	0.107	0.007	86.51	0.061	0.045
0.5859	8	489.91	0.013	0.011	97.22	0.065	0.054
0.7812	8	547.91	0.017	0.013	158.15	0.069	0.047
0.9766	8	668.45	0.015	0.013	123.79	0.087	0.072
1.1719	8	665.49	0.055	0.015	212.43	0.551	0.048
1.3672	8	841.57	0.016	0.015	137.68	0.573	0.094
1.5625	8	761.04	0.114	0.014	134.72	0.589	0.106
1.7578	8	753.26	0.116	0.016	260.96	0.55	0.058
1.9531	8	827.29	0.116	0.017	307.57	0.09	0.063



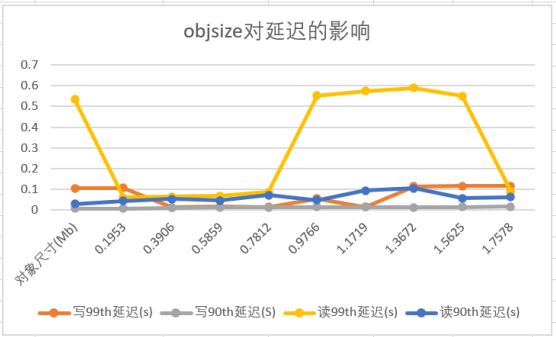


图 5.2.3 objsize 对吞吐率和延迟的影响

分析：

由折线图可见：

吞吐率方面：随着对象尺寸的线性增加，读写吞吐率也近似地呈现出线性增加的趋势，在对象尺寸达到一定大小之后，或许是受限于其他因素的限制，读写吞吐率开始趋于饱和和稳定。 也就是说其他因素不变，在一定范围内吞吐率与 objsize 呈正相关，随后趋于饱和。

延迟方面：对象尺寸的改变对读 99th 的延迟影响较大。 对 90th 的读写延迟几乎没有影响。其中 90th 的延迟基本稳定在 0-0.1s 之间。 可以推测对象尺寸对于读写延迟的影响较小。

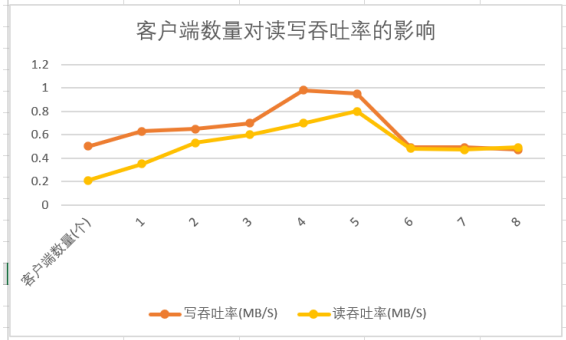
（4） 将并发客户端数量作为变量

样本数量 256、objsize1024B 保持不变。

设置循环每次均匀增加 num_clients 的值，每次增加一个客户端数量（初始为一）。 将输出结果重定向到文本文件，提取数据并绘制 excel 表格和折线图。

图表如图 5.2.4

对象尺寸(Mb)	客户端数量(个)	写吞吐率(MB/S)	写99th延迟(s)	写90th延迟(S)	读吞吐率(MB/S)	读99th延迟(s)	读90th延迟(s)
0.001	1	0.5	0.003	0.002	0.21	0.009	0.006
0.001	2	0.63	0.006	0.004	0.35	0.009	0.007
0.001	3	0.65	0.008	0.006	0.53	0.01	0.007
0.001	4	0.7	0.009	0.008	0.6	0.013	0.008
0.001	5	0.98	0.008	0.006	0.7	0.01	0.008
0.001	6	0.95	0.04	0.006	0.8	0.013	0.009
0.001	7	0.49	0.008	0.007	0.48	0.012	0.008
0.001	8	0.49	0.107	0.007	0.47	0.527	0.009
0.001	9	0.47	0.526	0.008	0.49	0.513	0.007



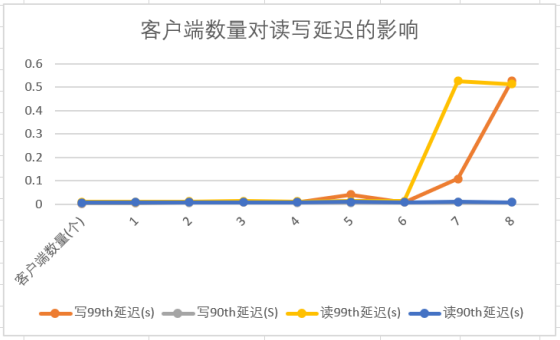


图 5.2.4 客户端数量对吞吐率和延迟的影响

分析：

吞吐率方面： 随着客户端数量的线性增加，吞吐率并没有太大的变化。 整体而言，吞吐率趋于稳定。

延迟方面： 客户端数量的改变对读写 90th 延迟几乎不造成变化。 但是偶尔会造成 99th 延迟的大范围波动。可见随着客户端数量的增加，读写延迟也会缓慢增加。当客户数量爆满时，读写延迟可能会很大。

分析总结：对象尺寸的大小主要影响力了 IO 吞吐率的变化。 一般而言采用更大的对象尺寸会带来更大的吞吐率，适用于对吞吐率有需求的应用。 并发客户端数量的增加主要影响 IO 延迟，对于需要 IO 延迟较低的应用，或许应采用更少的并发客户端数量。

六、实验总结

通过这次实验，我对对象存储有了基本的了解。 通过相关资料的查询，知道了一个对象存储存的是对象，并把对象放在 bucket 里。 通过几个软件的安装，知道如何部署一个简单的对象存储服务端和客户端。 虽然没有实际动手写代码和分析代码，但通过对实验现象的观察，了解到对象存储的一些基本概念，也知道如何使用 s3bench 对系统进行性能测试。 Cmd 脚本文件的编写很有趣，能够轻松地完成一些重要的功能，它会成为我日常编程中重要的工具。

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.