2019 级

《物联网数据存储与管理》课程

实 验 报 告

姓　名　余逸飞

学　号　U202015329

班　号　计算机 2002 班

日　期　2023.05.27

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

对象存储是一种数据存储架构，对象存储系统将数据按照非结构化数据形式进行存储。对象是对象存储的基本单元，每个对象通常包括数据本身、与数据相关的元数据 Metadata 和一个全局唯一的标识符 Identifier 等等。对象存储擅于处理图片、视频、音频等复杂格式数据，具有易扩展、可持久、灵活性好、可编程的特点。研究表明，大多数企业中大约 80%的数据都是非结构化的。

下图给出了对象存储与其他典型存储架构的比较：

Object	File	Block	Archive
Object Storage	**NAS**	**SAN**	**“Tape”**
Videos, photos serving streaming	All kinds of file	Attach to server	The file needs to be saved permanently
Read (download) data regularly	Read data regularly, install as a network drive	Run data directly on Storage	Rarely to download
High upload / download speed	High upload / download speed	Very High upload / download speed	High upload speed, slow download
Use with CDN	Many usage scenarios	Use with server (VM)	Use independently

图 2-1 各种存储架构的比较

（来源: https://usdc.vn/object-storage-vs-traditional-storage/）

三、实验环境

表 3-1 实验环境说明

操作系统	WSL2 + Ubuntu 20.04.6 LTS
处理器	12th Gen Intel(R) Core(TM) i7-12700H 2.30 GHz
对象存储服务端	Minio
对象存储客户端	Aws-cli/2.11.20
评测工具	S3bench

四、实验内容

实验内容分为以下三个部分：对象存储技术实践、对象存储性能分析、对象存储尾延迟观察。下面依次进行说明。

4.1 对象存储技术实践

首先是 Minio Server，直接在官网下载。这里选用已编译好的 Binary 版本，具体指令如下：

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
chmod +x minio
sudo mv minio /usr/local/bin/
```

下载完成后，运行老师提供的 run-minio.sh 脚本，在浏览器中输入 http://localhost:9090 并输入用户名和密码后，即可进入 Minio Console，具体如图所示：

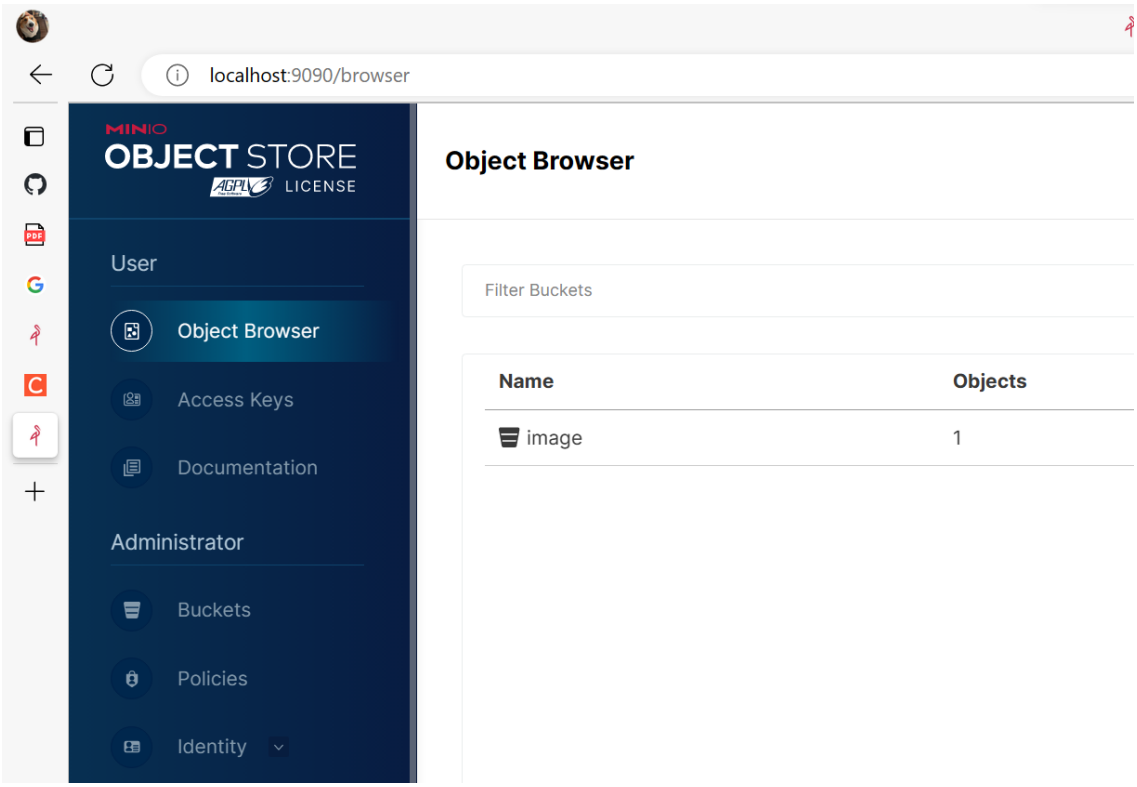

图 4-1 登录 Minio Console

我们可以在 Minio Console 中完成 Bucket 的添加、修改，并在 Bucket 内完成资源的上传、分享等等。由于本实验使用 Aws-cli 完成客户端搭建，Minio Console 部分就不做重点介绍了。

接下来介绍 Aws-cli 的配置，同样在官网下载，具体指令如下：

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

安装完成后，在终端输入 aws configure 并完成相关信息的输入，即可使用 aws-cli 访问 Minio Server。例如，我们可以展示当前 Server 中的所有 bucket，具体如图所示：

```
jackyu@Bello:~$ aws --endpoint-url http://localhost:9000 s3 ls
2023-05-18 21:04:07 image
```

图 4-2 使用 aws-cli 访问 Minio Server

4.2 对象存储性能分析

利用 s3bench 来测试对象存储系统，分别以客户端数量 numClients、对象个数 numSamples 以及对象大小 objectSize 为自变量，展开性能测试。

4.2.1 以客户端数量 numClients 为自变量展开的测试

设置对象个数 numSamples 为 256，对象大小为 32KB，客户端数量从 1 变化到 30，运行 s3bench 测试程序。根据经验分析，并行度越高，系统读写性能越好。实际测试后，结果如下图所示：

实验序号	numClients	objectSize(KB)	numSamples	w_throughput(MB/s)	w_duration(s)	r_throughput(MB/s)	r_duration(s)
1	1	31.2	256	14.63	0.547	34.71	0.23
2	3	31.2	256	33.2	0.241	98.58	0.081
3	5	31.2	256	47.73	0.168	150.32	0.053
4	7	31.2	256	62.91	0.127	185.67	0.043
5	10	31.2	256	72.3	0.111	214.72	0.037
6	13	31.2	256	89.38	0.09	266.04	0.03
7	16	31.2	256	88.02	0.091	285.9	0.028
8	19	31.2	256	100.22	0.08	288.86	0.028
9	22	31.2	256	90.78	0.088	309.01	0.026
10	25	31.2	256	103.95	0.077	296.09	0.027
11	28	31.2	256	102.37	0.078	280.46	0.029
12	30	31.2	256	109.63	0.073	278.77	0.029

图 4-3 以客户端数量为自变量的测试结果

接着绘制出读写吞吐率和读写延迟随客户端数量的变化曲线图如下：

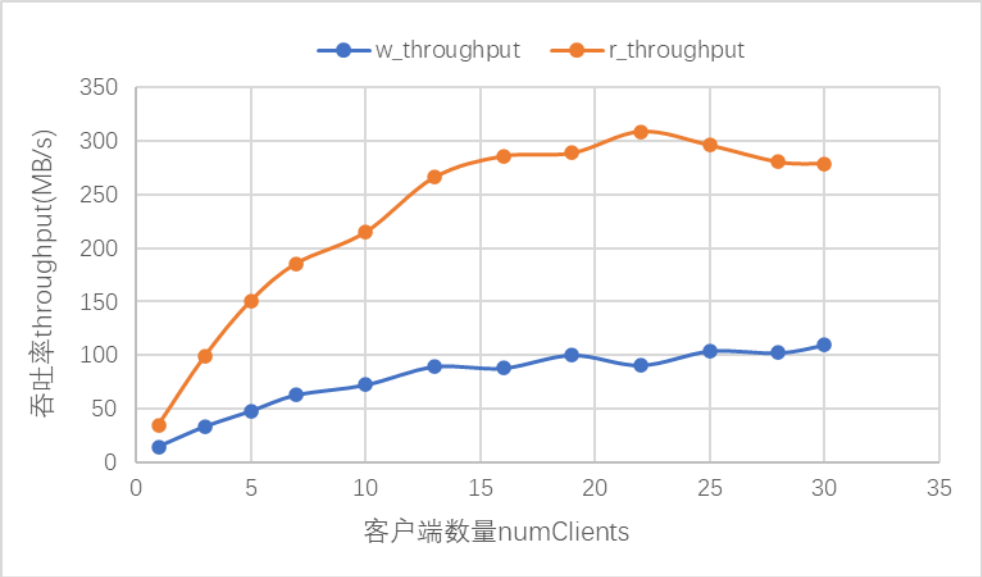

图 4-4 读写吞吐率(MB/s)随客户端数量的变化曲线图

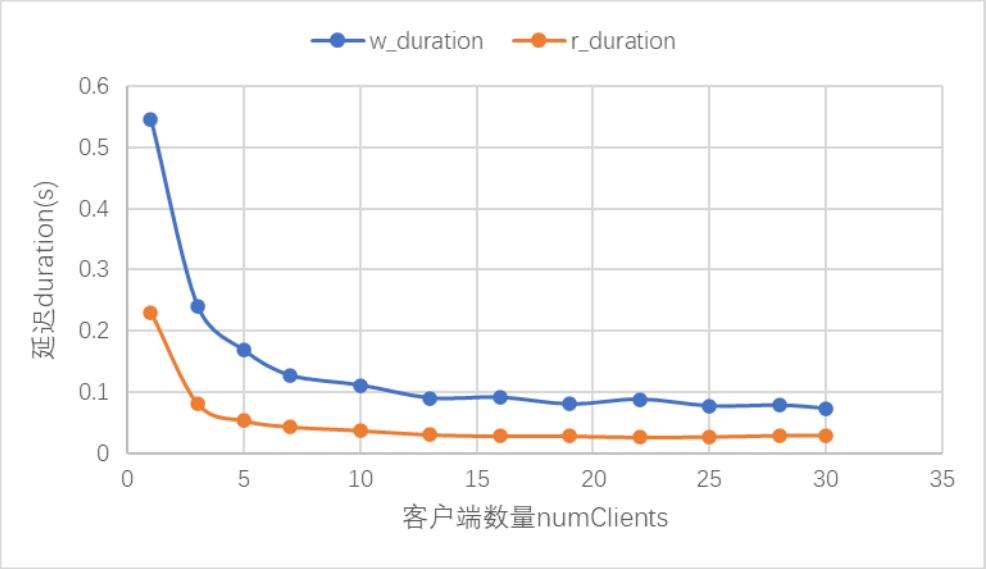

图 4-5 读写延迟(s)随客户端数量的变化曲线图

从上面的测试结果可以看出：随着客户端数量的增加，读写吞吐率呈增加趋势，读写延迟呈减小趋势，不过在客户端数量大于 16 以后，读写吞吐率增加的趋势大大放缓（读写延迟减小的趋势也大大放缓）。这里推测是对象存储系统有并行度的限制，同时并行度的增加会带来额外开销。除此之外，观察可知在不同并行度下，读吞吐率基本是写吞吐率的 2~3 倍，读的速度更快。总的来说，测试结果符合预期。

4.2.2 以对象个数 numSamples 为自变量展开的测试

设置客户端数量 numClients 为 1，对象大小为 32KB，对象个数从 1 变化到 2048，运行 s3bench 测试程序。由于测试样例规模较小，所以推测测试得到的系统读写效率与对象个数（对象存储系统规模）呈线性增长趋势。测试完成后，结果如下图所示：

实验序号	numClients	objectSize(KB)	numSamples	w_throughput(MB/s)	w_duration(s)	r_throughput(MB/s)	r_duration(s)
1	1	31.2	1	3.94	0.008	24.98	0.001
2	1	31.2	2	11.7	0.005	32.88	0.002
3	1	31.2	4	13.64	0.009	35.29	0.004
4	1	31.2	8	13.73	0.018	33.1	0.008
5	1	31.2	16	14.99	0.033	38.01	0.013
6	1	31.2	32	13.99	0.071	33.77	0.03
7	1	31.2	64	13.88	0.144	34.58	0.058
8	1	31.2	128	15.27	0.262	33.34	0.12
9	1	31.2	256	15.04	0.532	36.6	0.219
10	1	31.2	512	14.96	1.07	35.95	0.445
11	1	31.2	1024	15.2	2.106	37.79	0.847
12	1	31.2	2048	15.25	4.197	33.13	1.932

图 4-6 以对象个数为自变量的测试结果

绘制出读写吞吐率和读写延迟随对象个数的变化而变化的曲线图如下：

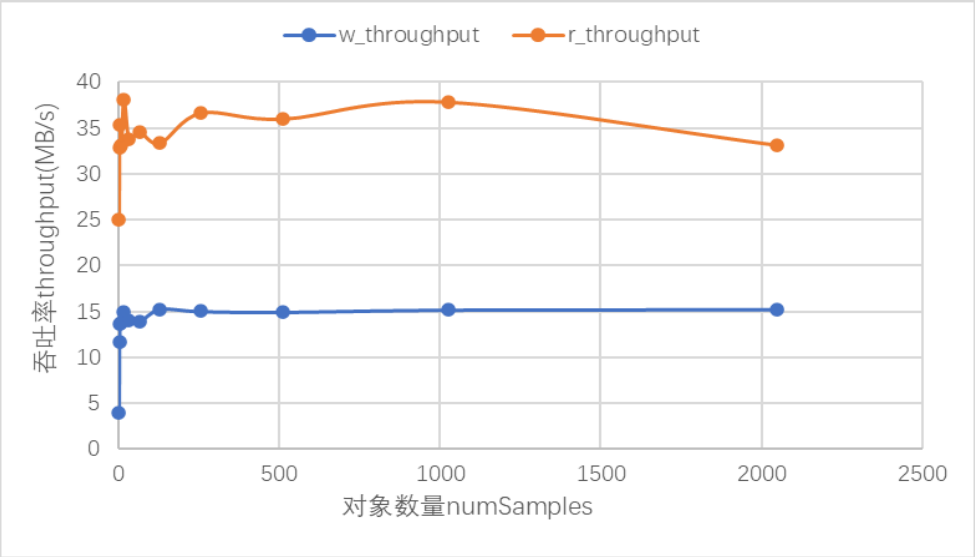

图 4-7 读写吞吐率(MB/s)随对象数量的变化曲线图

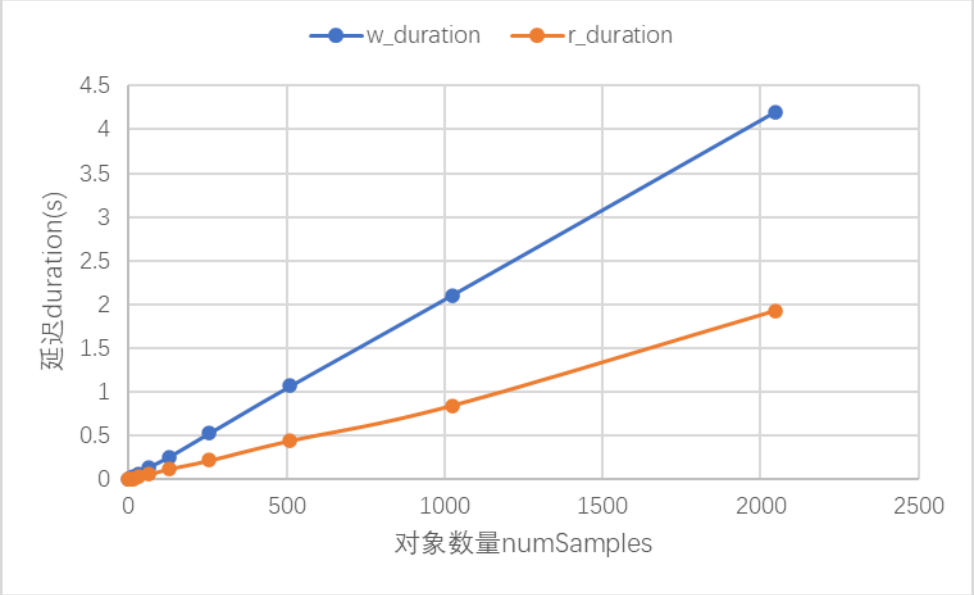

图 4-8 读写延迟(s)随对象数量的变化曲线图

从读写延迟曲线图中分析可知，当对象存储系统规模较小时，读写效率和对象存储系统基本呈线性关系。结果符合预期。

4.2.3 以对象大小 objectSize 为自变量展开的测试

设置客户端数量 numClients 为 1，对象个数为 256 个，对象大小数量从 0.001MB 变化到 4MB(实际为 3.9062MB)，运行 s3bench 测试程序。根据经验分析，对象大小和对象个数均是影响对象存储系统规模的因素，故预计测试结果与上节中针对对象个数的测试结果一致。实际测试后，结果如下图所示：

numClients	objectSize(MB)	numSamples	w_throughput(MB/s)	w_duration(s)	r_throughput(MB/s)	r_duration(s)
1	0.001	256	0.53	0.471	1.43	0.174
1	0.002	256	0.94	0.531	2.85	0.175
1	0.0039	256	2.03	0.492	5.71	0.175
1	0.0098	256	5.19	0.482	14.61	0.171
1	0.0195	256	10.59	0.472	25.89	0.193
1	0.0391	256	18.11	0.552	43.51	0.23
1	0.0977	256	36.54	0.684	119.21	0.21
1	0.1953	256	52.13	0.959	120.85	0.414
1	0.3906	256	87.64	1.141	192.2	0.52
1	0.9766	256	136.96	1.825	230.78	1.083
1	1.9531	256	167.37	2.987	744.78	0.671
1	3.9062	256	235.03	4.255	987.49	1.013

图 4-9 以对象大小为自变量的测试结果

绘制出读写吞吐率和读写延迟随对象大小的变化而变化的曲线图如下：

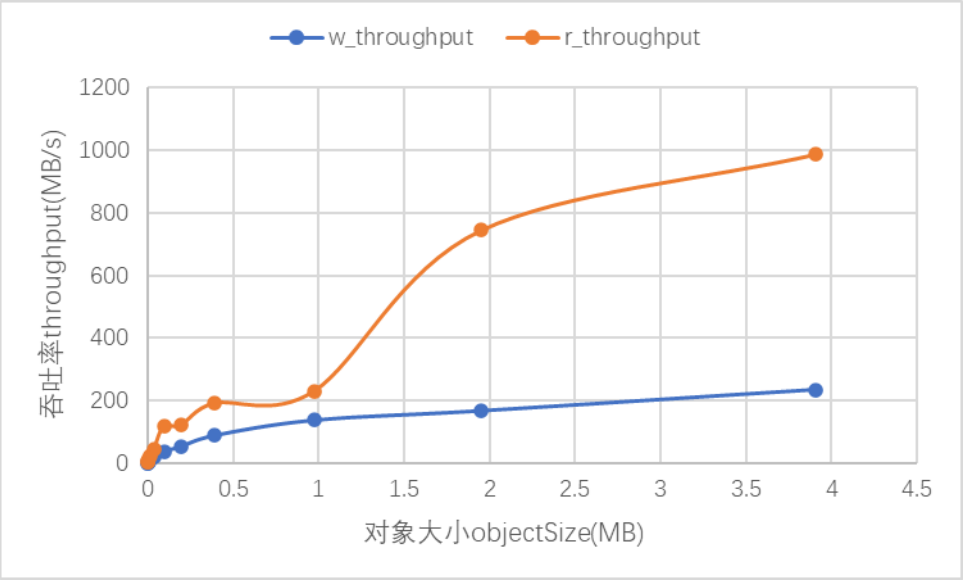

图 4-10 读写吞吐率(MB/s)随对象大小的变化曲线图

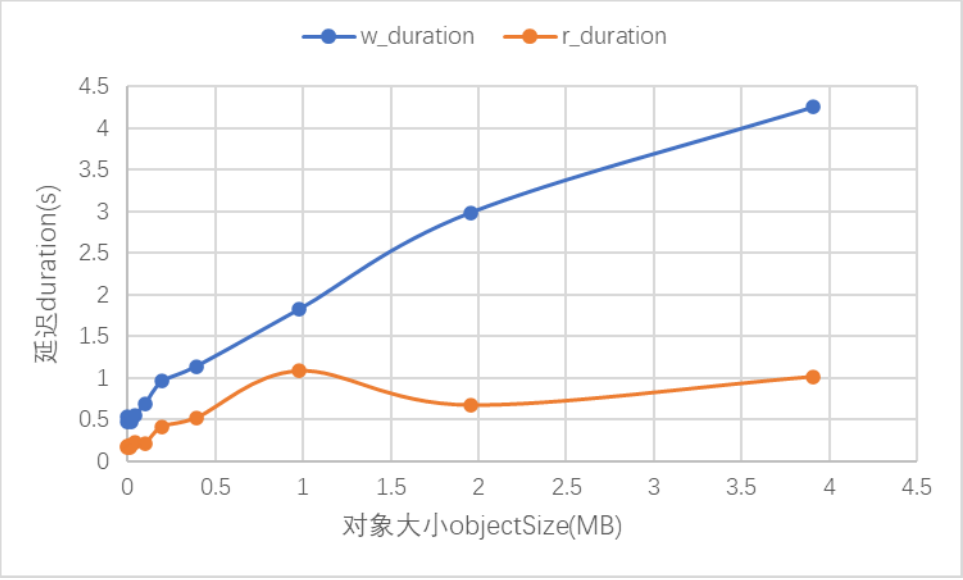

图 4-11 读写延迟(s)随对象大小的变化曲线图

从上述图表中分析，测试样例中对象大小为 1MB 的点为测试异常点，将该点排除后，上节测试得到的近似线性关系是成立的，符合结果预期。

4.3 对象存储尾延迟观察

以客户端数量 numClients 为 1，对象大小 objectSize 为 1MB，对象个数 numSamples 为 256 为参数进行 s3bench 测试，结果如图所示：

```
Result Summary for Write Operations
Total Transferred:      256.000 MB
Total Throughput: 1655.954 MB/s
Total Duration:         6.46857 s
Number of Errors: 0
-----------------------------------------------------
Write times Max  :      0.04129 s
Write times 99%th:      0.03961 s
Write times 90%th:      0.02819 s
Write times 75%th:      0.02591 s
Write times 50%th:      0.02471 s
Write times 25%th:      0.02337 s
Write times Min  :      0.02136 s


Result Summary for Read Operations
Total Transferred:      256.000 MB
Total Throughput: 378.522 MB/s
Total Duration:         1.47860 s
Number of Errors: 0
-----------------------------------------------------
Read times Max  : 0.07391 s
Read times 99%th: 0.02427 s
Read times 90%th: 0.00673 s
Read times 75%th: 0.00544 s
Read times 50%th: 0.00465 s
Read times 25%th: 0.00406 s
Read times Min  : 0.00322 s
```

图 4-12 s3bench 测试结果

将读写操作完成的比例同延迟 duration 之间的关系绘制折线图，结果如下：

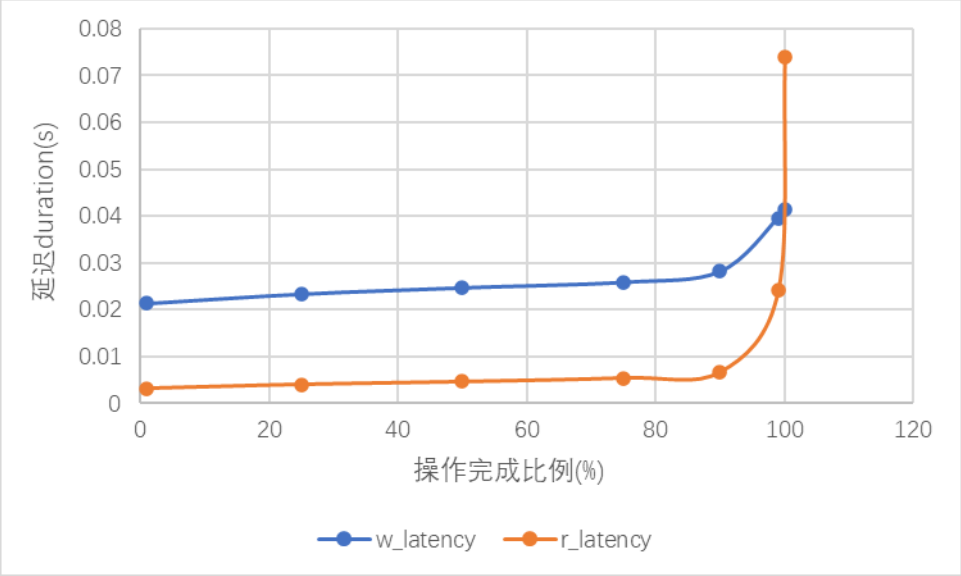

图 4-13 操作完成比例与读写延迟的关系

从上表可知，最后 10%的响应约占总响应时间的 90%，尾延迟现象很明显。

针对尾延迟现象，我们可以采用对冲请求进行应对。对冲请求的思想是如果当前请求的时延在 95%的预设请求时延内没有完成，则重新发起一个新的相同的请求，继续等待。一旦收到响应结果，客户端则会取消剩余的未处理请求。

修改 s3bench 测试程序，添加对冲请求部分代码。以相同的代码运行测试程序后，测试结果如图所示：

```
Result Summary for Write Operations
Total Transferred:      256.000 MB
Total Throughput: 1567.657 MB/s
Total Duration:         6.12366 s
Number of Errors: 0
-------------------------------------------------------
Write times Max  :      0.03675 s
Write times 99%th:      0.03531 s
Write times 90%th:      0.02588 s
Write times 75%th:      0.02444 s
Write times 50%th:      0.02302 s
Write times 25%th:      0.02198 s
Write times Min  :      0.01980 s


Result Summary for Read Operations
Total Transferred:      256.000 MB
Total Throughput: 345.651 MB/s
Total Duration:         1.35020 s
Number of Errors: 0
-------------------------------------------------------
Read times Max  : 0.03457 s
Read times 99%th: 0.02128 s
Read times 90%th: 0.00571 s
Read times 75%th: 0.00479 s
Read times 50%th: 0.00434 s
Read times 25%th: 0.00385 s
Read times Min  : 0.00179 s
```

图 4-14 加入对冲请求后的 s3bench 测试结果

将读写操作完成的比例同延迟 duration 之间的关系绘制折线图，结果如下：

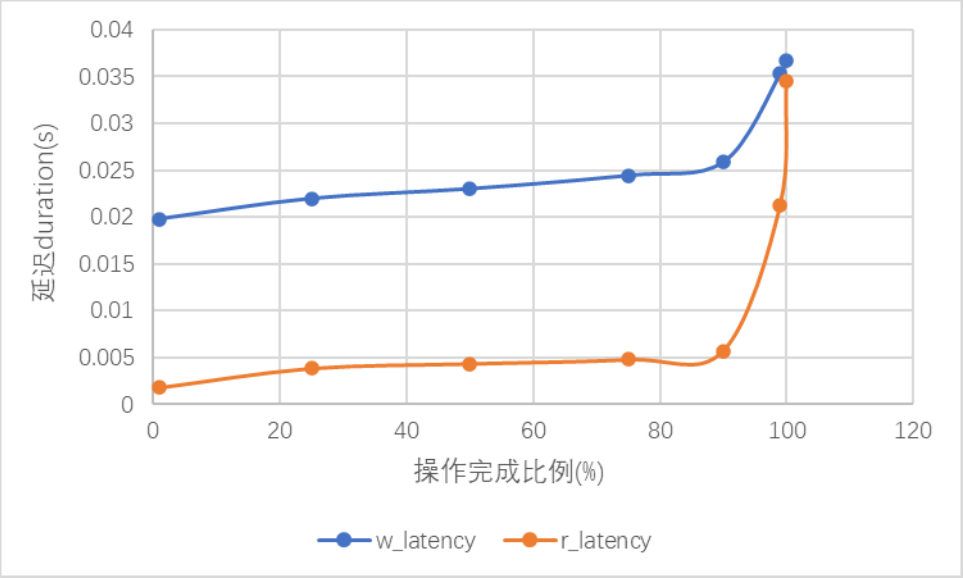

图 4-15 优化后操作完成比例与读写延迟的关系

从折线图中分析可知，加入对冲请求后，尾延迟的现象得到了减弱，最大读延迟从原来的 73.9ms 减小到现在的 34.5ms。

五、实验总结

本次实验中，我使用 minio 实现服务端、aws-cli 作为客户端构建了一个简单的对象存储系统，使用 s3bench 测试程序测试了对象存储系统的性能，系统地分析了各种因素对对象存储系统的影响。最后，我还观测了“尾延迟”现象，尝试利

用对冲请求缓建尾延迟现象，并观察了优化前后的测试结果。

针对本次实验，我也有一些遗憾之处。一开始，我想尝试上手 ceph，也在网上查阅了许多关于 ceph 和 docker 的资料，但因为还是刚刚接触这两个工具不太熟悉其使用，在尝试实现的时候遇到了不少困难，最终放弃这个方案。不过查阅资料的过程也是非常宝贵的学习经历，它让我接触了不少关于对象存储系统底层的知识，理解了许多先前没有接触过的分布式对象存储系统的概念，也让我上手 minio server 更加游刃有余。

总的来说，实验带给我的收获还是非常大的。在一个月左右的时间内，我从一个对对象存储系统毫无接触的小白，到自己亲手实现一个简单的对象存储系统并动手测试其性能，再到最后实验观测了尾延迟现象。整体实验下来带给我不少成就感，也让我学到了许多知识。通过测试分析系统性能的过程，让我初步体会到一些做研究的感觉。不仅如此，实验全程的引导我认为也非常合适，设计了丰富的实验选择并提供详细的学习资料链接，实验过程中需要我们自己动手查阅资料，自己学习使用各种工具，自己观测各种现象，对我们的自学能力是一个很好的锻炼。总的来说，这是我上大学以来做过的最映像深刻的实验之一！

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.