

2020 级

《物联网数据存储与管理》课程

实 验 报 告

姓 名 吾米提江·阿不来孜

学 号 U202015518

班 号 IOT2001

日 期 2023 年 5 月 20 日

目 录

一、实验目的	1
二、实验背景	1
三、实验环境	2
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	3
六、实验总结	19
参考文献	20

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

对象存储是面向对象/文件的、海量的互联网存储，它也可以直接被称为“云存储”。对象尽管是文件，它是已被封装的文件（编程中的对象就有封装性的特点），也就是说，在对象存储系统里，你不能直接打开/修改文件，但可以像 ftp 一样上传文件，下载文件等。另外对象存储没有像文件系统那样有一个很多层级的文件结构，而是只有一个“桶”（bucket）的概念（也就是存储空间），“桶”里面全部都是对象，是一种很扁平化的存储方式。其最大的特点就是它的对象名称就是一个域名地址，一旦对象被设置为“公开”，所有网民都可以访问到它；它的拥有者还可以通过 REST API 的方式访问其中的对象。因此，对象存储最主流的使用场景，就是存储网站、移动 app 等互联网/移动互联网应用的静态内容（视频、图片、文件、软件安装包等等）。

对象存储在很多重要方面与 SAN 和 NAS 迥然不同，对存储管理员而言最显著的区别在于对象存储没有 LUNs，卷以及 RAID 等要素。对象数据不是存储在固定的块，而是在大小可变的“容器”里。鉴于元数据（metadata）和数据本身可通过传统数据访问方法进行访问，对象存储允许数据被直接访问。此外，支持对象级和命令级的安全策略设置。

本次实验采用 Minio 作为实验的存储服务端。Minio 是一个基于 Apache License v2.0 开源协议的对象存储服务。它兼容亚马逊 S3 云存储服务接口，非常适合于存储大容量非结构化的数据，例如图片、视频、日志文件、备份数据和容器/虚拟机镜像等，而一个对象文件可以是任意大小，从几 kb 到最大 5T 不等。

由于 Minio 本身包含服务端和客户端。故本次实验使用的对象存储客户端是 Minio 的客户端部分即(minio client) Mc。

本次实验采用 s3bench 作为实验的测评工具。s3bench 是使用了 AWS Go SDK 实现的对象存储服务器测试程序，可以通过设置请求对象大小和数量、并行客户端数量等参数，根据吞吐率、延迟等结果评测对象存储系统的性能。

三、实验环境

本次实验主要实验环境如下表所示：

硬件环境	CPU	Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz (8 CPUs), ~2.4GHz
	内存	8.0 GB
软件环境	操作系统	Windows 10 专业版 64 位系统
	其它软件	服务端： minio 客户端： mc (minio client) 测试工具： s3bench

四、实验内容

本次实验主要内容如下：

1. 搭建系统环境，包括代码版本控制系统 Git，各种语言运行环境。
2. 对对象存储系统进行实践，采用 minio/mc 配置服务器端和客户端，并进行简单的创建或删除 bucket、上传或删除文件等操作。
3. 对对象存储系统进行测试，采用 s3bench 进行负载测试。

4.1 对象存储技术实践

1. 在 Windows 环境下配置 minio server 端，通过浏览器登陆 127.0.0.1:9000 查看效果，并熟悉各项 minio server 相关的操作。
2. 在 Windows 环境下配置 minio 客户端 mc，在命令行下输入命令实现创建、删除 bucket 等操作，并通过访问网址查看新建及删除的结果。

4.2 对象存储性能分析

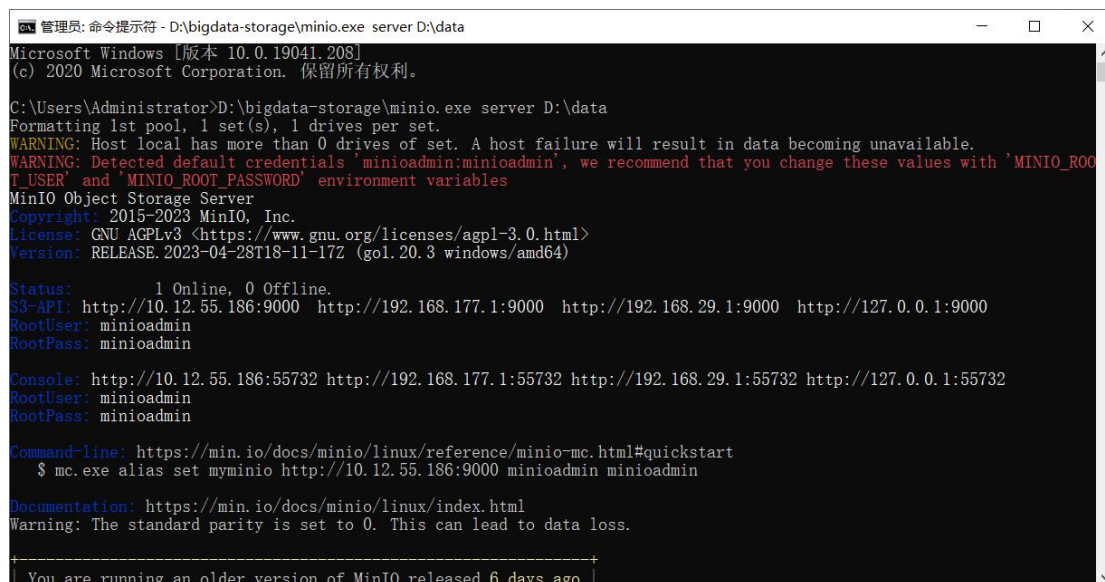
1. 分析其读写性能之间的对比。
2. 修改脚本范例中 numClients 参数的值，从而修改同一时间产生的请求数，即并发数，分析并发数对存储性能的影响。
3. 修改脚本范例中 numSamples 参数的值，从而修改 workers 的数量，分析 workers 数量对存储性能的影响。
4. 修改脚本范例中 objectSize 参数的值，从而修改文件的大小，分析块文件

大小对存储性能的影响。

五、实验过程

5.1 对象存储技术实践

进入 Minio 官网下载并安装 Windows 系统对应版本的 minio server 和 minio client。在本地 CMD 窗口下进入指定安装位置，并按照官网上给出的例子和规范的运行命令来运行 minio.exe server，运行结果如图 5.1 所示：



```
管理员: 命令提示符 - D:\bigdata-storage\minio.exe server D:\data
Microsoft Windows [版本 10.0.19041.208]
(c) 2020 Microsoft Corporation. 保留所有权利。

C:\Users\Administrator>D:\bigdata-storage\minio.exe server D:\data
Formatting 1st pool, 1 set(s), 1 drives per set.
WARNING: Host local has more than 0 drives of set. A host failure will result in data becoming unavailable.
WARNING: Detected default credentials 'minioadmin:minioadmin', we recommend that you change these values with 'MINIO_ROOT_USER' and 'MINIO_ROOT_PASSWORD' environment variables
MinIO Object Storage Server
Copyright: 2015-2023 MinIO, Inc.
License: GNU AGPLv3 <https://www.gnu.org/licenses/agpl-3.0.html>
Version: RELEASE.2023-04-28T18-11-17Z (go1.20.3 windows/amd64)

Status: 1 Online, 0 Offline.
S3-API: http://10.12.55.186:9000 http://192.168.177.1:9000 http://192.168.29.1:9000 http://127.0.0.1:9000
RootUser: minioadmin
RootPass: minioadmin

Console: http://10.12.55.186:55732 http://192.168.177.1:55732 http://192.168.29.1:55732 http://127.0.0.1:55732
RootUser: minioadmin
RootPass: minioadmin

Command-line: https://min.io/docs/minio/linux/reference/minio-mc.html#quickstart
$ mc.exe alias set myminio http://10.12.55.186:9000 minioadmin minioadmin

Documentation: https://min.io/docs/minio/linux/index.html
Warning: The standard parity is set to 0. This can lead to data loss.

+-----+
| You are running an older version of MinIO released 6 days ago |
```

图 5.1 Minio server 运行结果

根据 Minio server 运行结果中的打印信息可以看到，调用 minio 服务端的 API 的 url 为本机网络 IP 的 9000 端口，Minio 服务端控制台的 url 为本机网络 IP 的 158841 端口。同时，还可以看出，两个监听端口对应的 RootUser 和 RootPass 均为默认的 minioadmin。

其中红字警告表示系统检测到目前 Minio 服务端的账号密码为默认值，推荐使用 MINIO_ROOT_USER 和 MINIO_ROOT_PASSWORD 的环境变量去更改默认值。此时有两个方案一个是忽略警告（不会都后续实验造成很大影响，但是实际应用场景中会出现安全问题等一系列不好的事情），二是通过命令改变环境变量的值自定义账户和密码以及端口，实验结果如下图 5.2 所示：

```
管理员: 命令提示符 - D:\bigdata-storage\minio.exe server D:\minio\file --console-address "127.0.0.1:9000" --address "127.0.0.1:9090"
Microsoft Windows [版本 10.0.19041.208]
(c) 2020 Microsoft Corporation. 保留所有权利。

C:\Users\Administrator>set MINIO_ROOT_USER=admin
C:\Users\Administrator>set MINIO_ROOT_PASSWORD=12345678
C:\Users\Administrator>D:\bigdata-storage\minio.exe server D:\minio\file --console-address "127.0.0.1:9000" --address "127.0.0.1:9090"
Formatting 1st pool, 1 set(s), 1 drives per set.
WARNING: Host local has more than 0 drives of set. A host failure will result in data becoming unavailable.
MinIO Object Storage Server
Copyright: 2015-2023 MinIO, Inc.
License: GNU AGPLv3 <https://www.gnu.org/licenses/agpl-3.0.html>
Version: RELEASE.2023-04-28T18-11-17Z (go1.20.3 windows/amd64)

Status:          1 Online, 0 Offline.
S3-API: http://127.0.0.1:9090
RootUser: admin
RootPass: 12345678

Console: http://127.0.0.1:9000
RootUser: admin
RootPass: 12345678

Command-line: https://min.io/docs/minio/linux/reference/minio-mc.html#quickstart
$ mc.exe alias set myminio http://127.0.0.1:9090 admin 12345678

Documentation: https://min.io/docs/minio/linux/index.html
Warning: The standard parity is set to 0. This can lead to data loss.
```

图 5.2 Minio server 接触警告后的运行结果

在图 1.2 中我们将默认端口改为了 9090 但是后续实验仍是在图 1.1 结果上进行的,默认端口为 9000。且经测试发现,如果借助浏览器访 `http://127.0.0.1:9000` 等 API 的 url,浏览器仍会重定向至控制台所监听的端口。此时在浏览器中访问的结果如下图 5.3 所示:

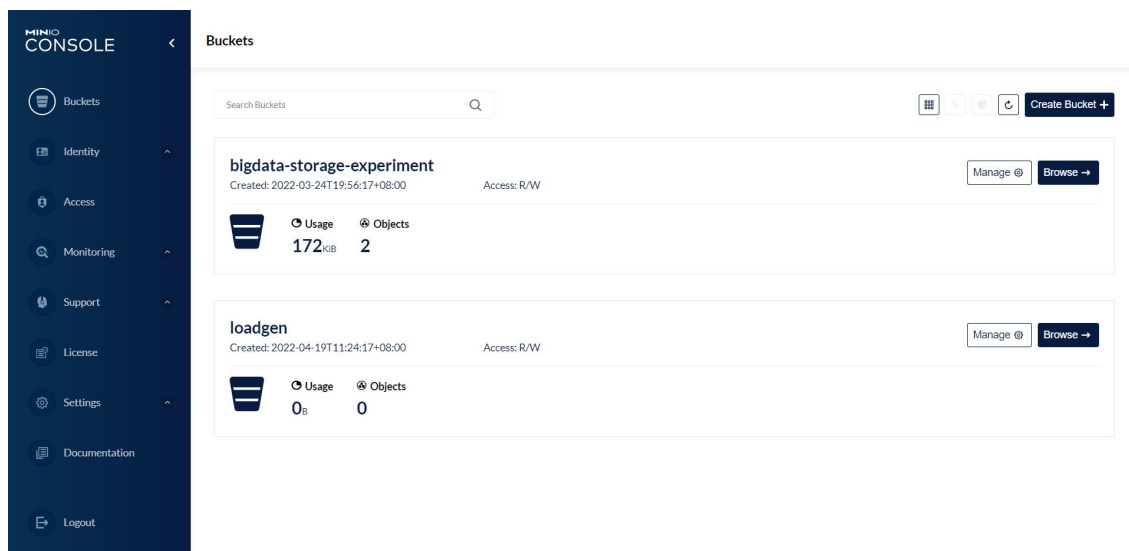


图 5.3 Minio server 控制台

在控制台中我们可以尝试着创建一个 bucket,点击页面中的 Create Bucket 可以创建新的 bucket,创建过程如下图 5.4 所示:

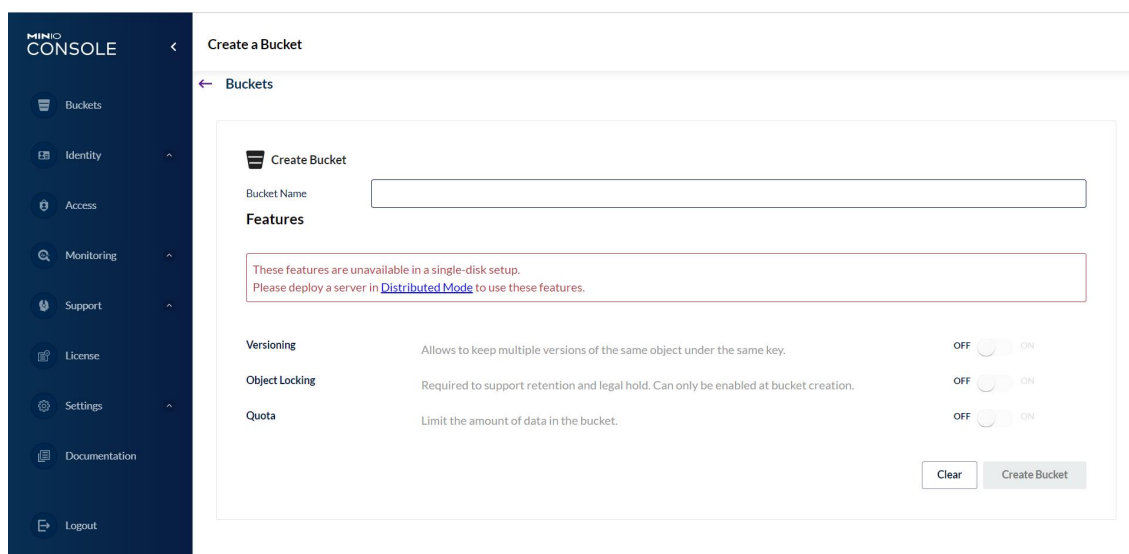


图 5.4 Minio server 控制台创建 bucket

在控制台中我们也可以尝试着创建一个用户，点击页面中的 **Create User** 可以创建一个新的用户，创建过程如下图 5.5 所示：

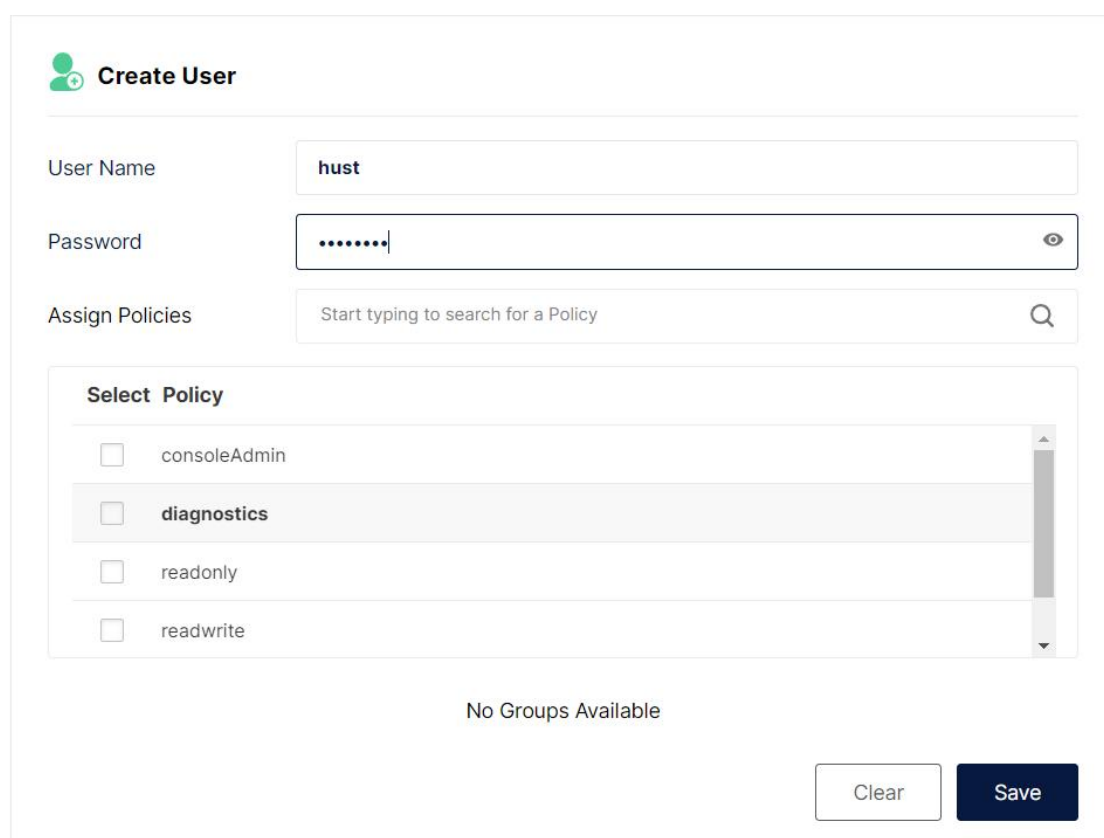


图 5.5 Minio server 控制台创建 User

在控制台我们也可以查看新创建的 bucket 里的详细信息，如下图 5.6 所示：

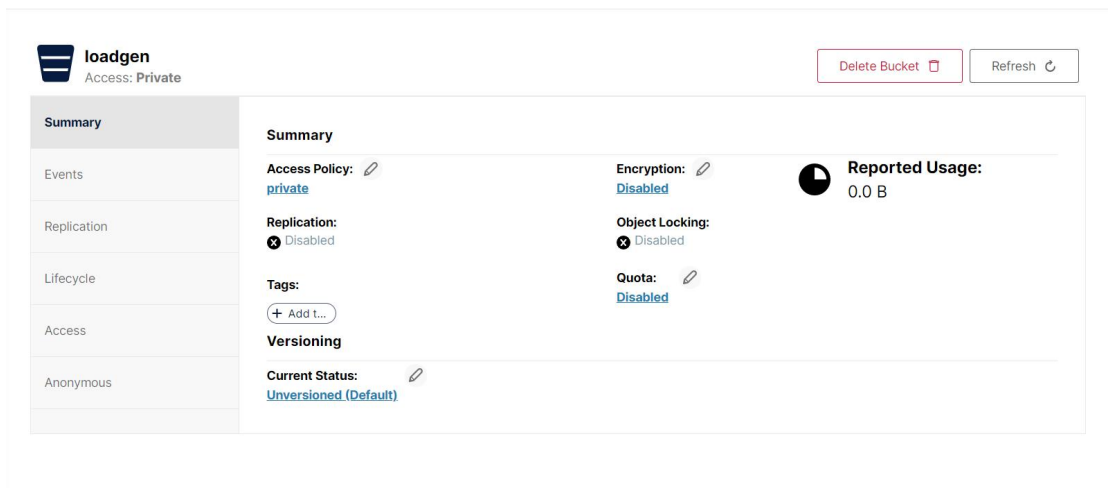


图 5.6 Minio server 控制台观察 bucket

以上创建 bucket、用户等操作我们将会 Minio Client 里通过命令行的形式再次实现，以更好的熟悉 MC 端的使用。

接下来，按照官网上的例子在命令行下运行 mc.exe，添加一个名为 umut001 的存储服务。访问存储服务中的 bucket 和对象，可以看到之前创建的 bucket。我们也可以通过指令，从客户端再重新创建一个名为 test1 的新 bucket，创建过程如下图 5.7 所示。再次访问存储服务中的 bucket 和对象，可以看到新加入的 bucket。结果如图 5.8 所示：

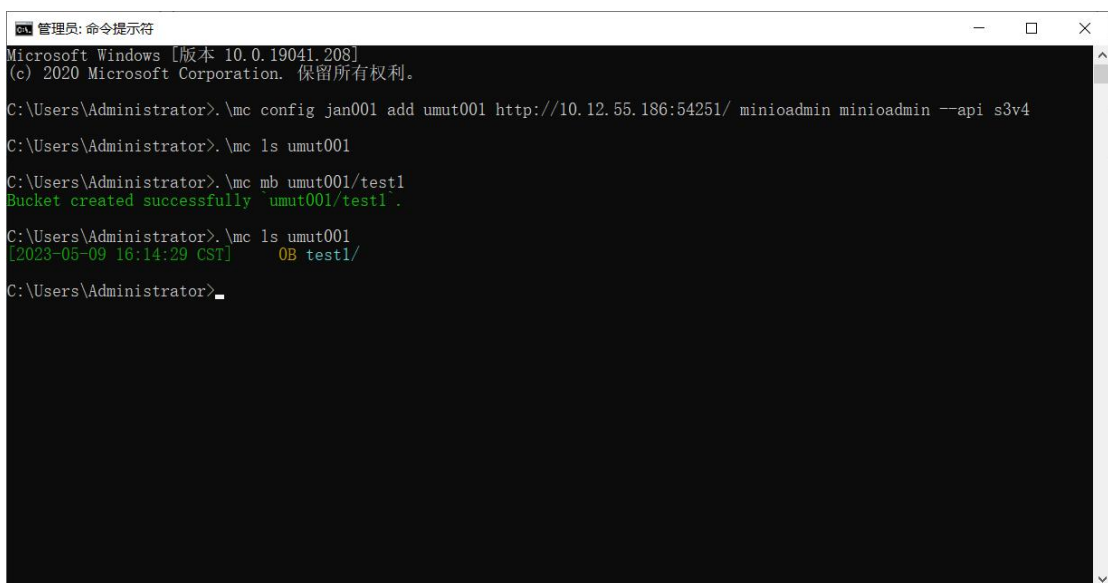


图 5.7 在 Mc 下创建存储服务(用户)和 bucket

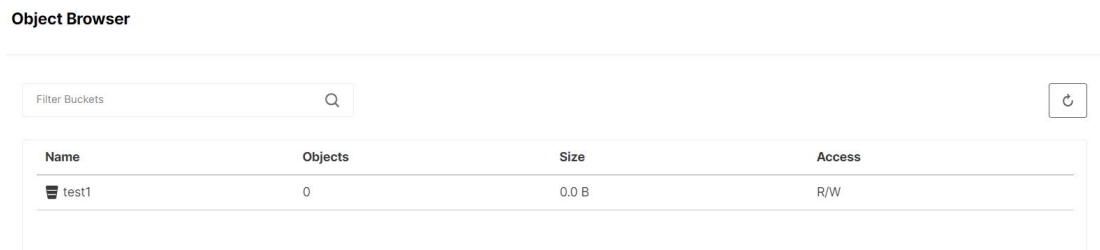


图 5.8 在浏览器中的控制台看到新添加的 bucket

在实验过程我发现一个问题就是，由于大家实验环境有大小差异，在我第一次做实验时通过参考以往的使用过程从而写出的命令：

`.\mc config xxx(名字任取) add xxx(服务名任取) http://10.12.55.186:54251/(从自身服务端中获取) minioadmin minioadmin (用户名和密码) --api s3v4`

只会在本地 minio client 存在的文件夹内添加一个新的名为 xxx 服务文件夹，并不会真正写到服务端指定的 config 里，也不会浏览器里的控制台显示有新的 bucket 创建。原因是：minio 更新迭代较快有些比较老式的命令行语句不在适用，故很有可能导致操作失败。

解决：用 minio server 中提示的（详细可见图 5.1）最新指令形式来操作其客户端 minio client。我使用的 minio 版本需要的正确命令格式如下图 5.9 所示：

```
C:\Users\Administrator>mc.exe alias set myminio http://10.12.55.186:9000 minioadmin minioadmin
Added myminio successfully.
```

图 5.9 正确的 mc 端命令

再次重复图 1.7 和 1.8 中的过程，创建一个叫 myminio 的服务，并添加两个 bucket，再去观察其结果在浏览器中控制台是否真的添加，也可以在 C 盘里对应的 config 文件中观察自己写的这段命令(添加的服务和 bucket)是否真的写到了 config 文件里，实验结果如下图 5.10 和 5.11 所示：

```
[2023-05-09 16:14:29 CST] 0B umut001/
[2022-10-13 09:23:56 CST] 0B usb_driver/
[2022-08-24 21:22:10 CST] 0B 「开始」菜单/

C:\Users\Administrator>. \mc ls myminio
[2023-05-09 16:38:55 CST] 0B loadgen/

C:\Users\Administrator>. \mc ls myminio
[2023-05-09 16:38:55 CST] 0B loadgen/

C:\Users\Administrator>. \mc mb myminio/qweess
Bucket created successfully myminio/qweess .

C:\Users\Administrator>. \mc ls myminio
[2023-05-09 16:38:55 CST] 0B loadgen/
[2023-05-09 17:02:28 CST] 0B qweess/

C:\Users\Administrator>
```

图 5.10 改良后的 mc 端命令的使用

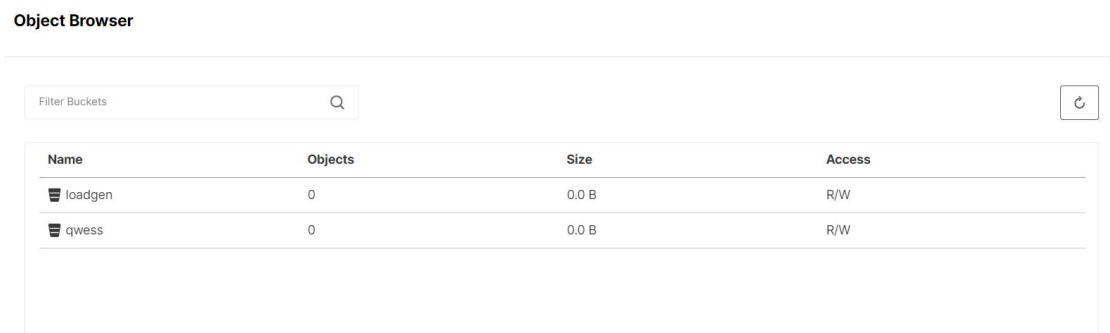


图 5.11 改良后的 mc 端命令控制台验证

我们可以看到，控制台中确实添加成功了一个 loadgen 和另一个 qweess 的 bucket，说明使用最新版本的 Minio 命令是非常有必要的。

5.2 对象存储性能分析

下载 Windows 系统对应版本的 s3bench.exe 和其对应的命令脚本的范例脚本 s3bench.cmd。并最好保存在于 minio server 和 mc 同一个文件夹内。所需文件结构如下图 5.12 所示：



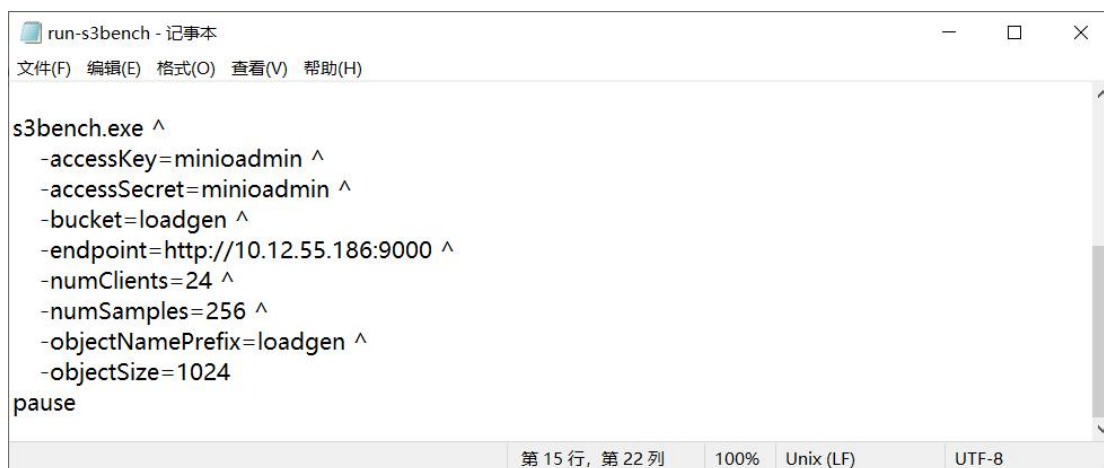
 run-s3bench.cmd	2022/4/19 11:28	Windows 命令脚本	1 KB
 s3bench.exe	2022/4/19 10:27	应用程序	11,596 KB

图 5.12 测试所需的文件 s3bench

修改脚本范例，分别将 `accessKey` 和 `accessSecret` 的值改为登录控制台对应的账号和密码，及 `minioadmin`，将 `bucket` 的值改为我们自己创建的用于测试的 `bucket` 名字。及 `loadgen`，修改后的脚本如下图 5.13 所示：



```
s3bench.exe ^
-accessKey=minioadmin ^
-accessSecret=minioadmin ^
-bucket=loadgen ^
-endpoint=http://10.12.55.186:9000 ^
-numClients=24 ^
-numSamples=256 ^
-objectNamePrefix=loadgen ^
-objectSize=1024
pause
```

图 5.13 s3bench 脚本修改

初步运行 `s3bench.amd` 脚本，运行结果如下图 5.14 所示：

由图中我们可以看到一些信息，首先他会显示我们在脚本中修改和填写的信息，如 `accessKey`、`accessSecret`、`bucket`、`endpoint`、`numClients`、`numSamples`、`objectNamePrefix`、`objectSize` 等信息以便更清晰的确认本次运行的条件配置。

其次分别显示 `Read` 和 `Write` 过程中的 `Throughput`（吞吐量）、`Time-90th%ile`、`Time-99th%ile` 和 `duration`（持续时间）以及发生的 `Errors` 数量等（一般为 0）；

在本次测试中，我们主要是通过控制变量法，每次改变单一的变量。即将脚本范例中 `numClients` 参数的值取 8、16、24，将脚本范例中 `numSamples` 参数的值取 256、512、768，将脚本范例中 `objectSize` 参数的值取 1024、2048、3072。分别再次测试，一共可以得到 $4 \times 4 = 16$ 组数据。得出在参数取这些值的情况下，服务器的吞吐率和延迟如表等。由于篇幅有限，我们只给出分别改变三个值时各一组实验结果图，即如下图 5.14、5.15、5.16 所示：

(numclient= 24 numsamples=256,objectsize=1024)

```
C:\Windows\system32\cmd.exe
C:\Users\Administrator>s3bench.exe -accessKey=minioadmin -accessSecret=minioadmin -bucket=loadgen -endpo
int=http://10.12.55.186:9000 -numClients=24 -numSamples=256 -objectNamePrefix=loadgen -objectSize=1024
Test parameters
endpoint(s): [http://10.12.55.186:9000]
bucket: loadgen
objectNamePrefix: loadgen
objectSize: 0.0010 MB
numClients: 24
numSamples: 256
verbose: %!d(bool=false)
tracing: %!d(bool=false)

Generating in-memory sample data... Done (0s)

Running Write test...

Running Read test...

Test parameters
endpoint(s): [http://10.12.55.186:9000]
bucket: loadgen
objectNamePrefix: loadgen
objectSize: 0.0010 MB
numClients: 24
numSamples: 256
verbose: %!d(bool=false)
tracing: %!d(bool=false)

Results Summary for Write Operation(s)
Total Transferred: 0.250 MB
Total Throughput: 1.28 MB/s
Total Duration: 0.195 s
Number of Errors: 0
-----
Write times Max: 0.035 s
Write times 99th %ile: 0.032 s
Write times 90th %ile: 0.025 s
Write times 75th %ile: 0.020 s
Write times 50th %ile: 0.017 s
Write times 25th %ile: 0.015 s
Write times Min: 0.009 s

Results Summary for Read Operation(s)
Total Transferred: 0.250 MB
Total Throughput: 2.83 MB/s
Total Duration: 0.088 s
Number of Errors: 0
-----
Read times Max: 0.025 s
Read times 99th %ile: 0.024 s
Read times 90th %ile: 0.014 s
Read times 75th %ile: 0.010 s
Read times 50th %ile: 0.006 s
Read times 25th %ile: 0.004 s
Read times Min: 0.001 s

Cleaning up 256 objects...
Deleting a batch of 256 objects in range [0, 255]... Succeeded
Successfully deleted 256/256 objects in 459.4074ms

C:\Users\Administrator>pause
请按任意键继续. . .
```

图 5.14 s3bench 运行结果图 1

(numclient= 16 numsamples=512,objectsize=1024)

```
C:\Windows\system32\cmd.exe

C:\Users\Administrator>s3bench.exe -accessKey=minioadmin -accessSecret=minioadmin -bucket=loadgen -endpoint=http://10.12.55.186:9000 -numClients=16 -numSamples=512 -objectNamePrefix=loadgen -objectSize=1024
Test parameters
endpoint(s):      [http://10.12.55.186:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:        0.0010 MB
numClients:        16
numSamples:        512
verbose:           %!d(bool=false)
tracing:           %!d(bool=false)

Generating in-memory sample data... Done (0s)

Running Write test...

Running Read test...

Test parameters
endpoint(s):      [http://10.12.55.186:9000]
bucket:           loadgen
objectNamePrefix: loadgen
objectSize:        0.0010 MB
numClients:        16
numSamples:        512
verbose:           %!d(bool=false)
tracing:           %!d(bool=false)

Results Summary for Write Operation(s)
Total Transferred: 0.500 MB
Total Throughput:  1.28 MB/s
Total Duration:    0.390 s
Number of Errors:  0

Write times Max:    0.032 s
Write times 99th %ile: 0.032 s
Write times 90th %ile: 0.015 s
Write times 75th %ile: 0.012 s
Write times 50th %ile: 0.011 s
Write times 25th %ile: 0.010 s
Write times Min:    0.007 s

Results Summary for Read Operation(s)
Total Transferred: 0.500 MB
Total Throughput:  3.88 MB/s
Total Duration:    0.129 s
Number of Errors:  0

Read times Max:     0.011 s
Read times 99th %ile: 0.009 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 512 objects...
Deleting a batch of 512 objects in range [0, 511]... Succeeded
Successfully deleted 512/512 objects in 913.8936ms

C:\Users\Administrator>pause
请按任意键继续. . .
```

图 5.15 s3bench 运行结果图 2

(numclient= 16 numsamples=256,objectsize=1536)

```
C:\Windows\system32\cmd.exe
C:\Users\Administrator>s3bench.exe -accessKey=minioadmin -accessSecret=minioadmin -bucket=loadgen -endpoint=http://10.12.55.186:9000 -numClients=16 -numSamples=256 -objectNamePrefix=loadgen -objectSize=1536
Test parameters
endpoint(s): [http://10.12.55.186:9000]
bucket: loadgen
objectNamePrefix: loadgen
objectSize: 0.0015 MB
numClients: 16
numSamples: 256
verbose: %!d(bool=false)
tracing: %!d(bool=false)

Generating in-memory sample data... Done (0s)

Running Write test...

Running Read test...

Test parameters
endpoint(s): [http://10.12.55.186:9000]
bucket: loadgen
objectNamePrefix: loadgen
objectSize: 0.0015 MB
numClients: 16
numSamples: 256
verbose: %!d(bool=false)
tracing: %!d(bool=false)

Results Summary for Write Operation(s)
Total Transferred: 0.375 MB
Total Throughput: 2.02 MB/s
Total Duration: 0.186 s
Number of Errors: 0

Write times Max: 0.026 s
Write times 99th %ile: 0.023 s
Write times 90th %ile: 0.014 s
Write times 75th %ile: 0.012 s
Write times 50th %ile: 0.011 s
Write times 25th %ile: 0.010 s
Write times Min: 0.007 s

Results Summary for Read Operation(s)
Total Transferred: 0.375 MB
Total Throughput: 7.62 MB/s
Total Duration: 0.049 s
Number of Errors: 0

Read times Max: 0.007 s
Read times 99th %ile: 0.007 s
Read times 90th %ile: 0.004 s
Read times 75th %ile: 0.004 s
Read times 50th %ile: 0.003 s
Read times 25th %ile: 0.002 s
Read times Min: 0.001 s

Cleaning up 256 objects...
Deleting a batch of 256 objects in range [0, 255]... Succeeded
Successfully deleted 256/256 objects in 450.3586ms

C:\Users\Administrator>pause
请按任意键继续. . .
```

图 5.16 s3bench 运行结果图 3

重复上述过程，得到 4+4+4=16 组数据，并将 s3bench 运行结果框中显示的关键数据制成表格以便后续可视化处理，得到的表格如下：

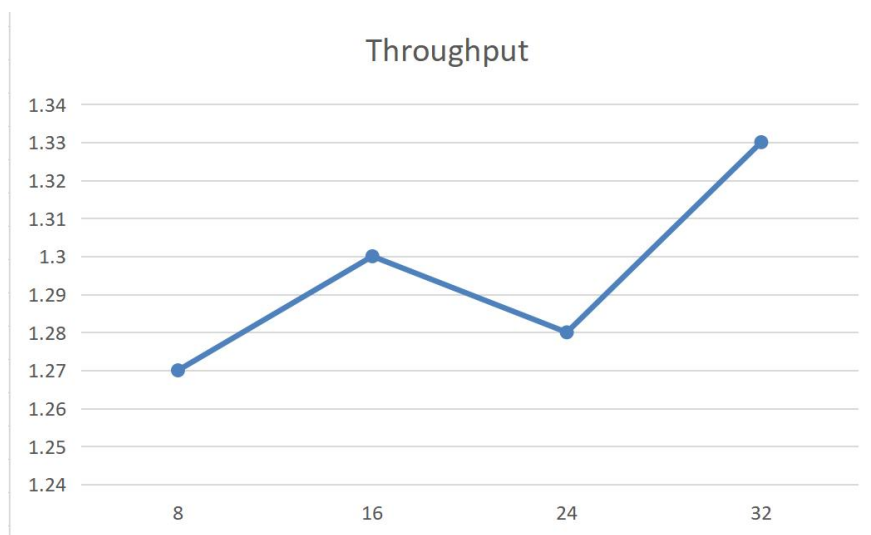
表 1 不同条件下的吞吐率和延迟

Clients (并发数)	Samples (样本数)	objectSiz (文件 大小)	Write (写入)				Read (读取)			
			Throughput (吞吐率)	延迟 90%	延迟 99%	duration	Throughput (吞吐率)	延迟 90%	延迟 99%	duration
8	256	1024	1.27	0.007	0.020	0.197	5.20	0.003	0.002	0.048
16	256	1024	1.30	0.014	0.024	0.192	4.90	0.005	0.006	0.051
24	256	1024	1.28	0.025	0.032	0.195	2.83	0.014	0.024	0.088
32	256	1024	1.33	0.030	0.041	0.187	4.94	0.011	0.013	0.051
16	256	1024	1.30	0.014	0.024	0.192	4.90	0.005	0.006	0.051
16	512	1024	1.28	0.015	0.032	0.390	3.88	0.006	0.009	0.129
16	768	1024	1.30	0.014	0.046	0.557	5.03	0.005	0.007	0.149
16	1024	1024	1.32	0.013	0.054	0.759	5.12	0.004	0.006	0.192
16	256	512	0.42	0.023	0.026	0.295	1.03	0.022	0.043	0.122
16	256	1024	1.30	0.014	0.024	0.192	4.90	0.005	0.006	0.051
16	256	1536	2.02	0.014	0.023	0.186	7.62	0.004	0.007	0.049
16	256	2048	2.73	0.015	0.019	0.183	8.14	0.006	0.011	0.061

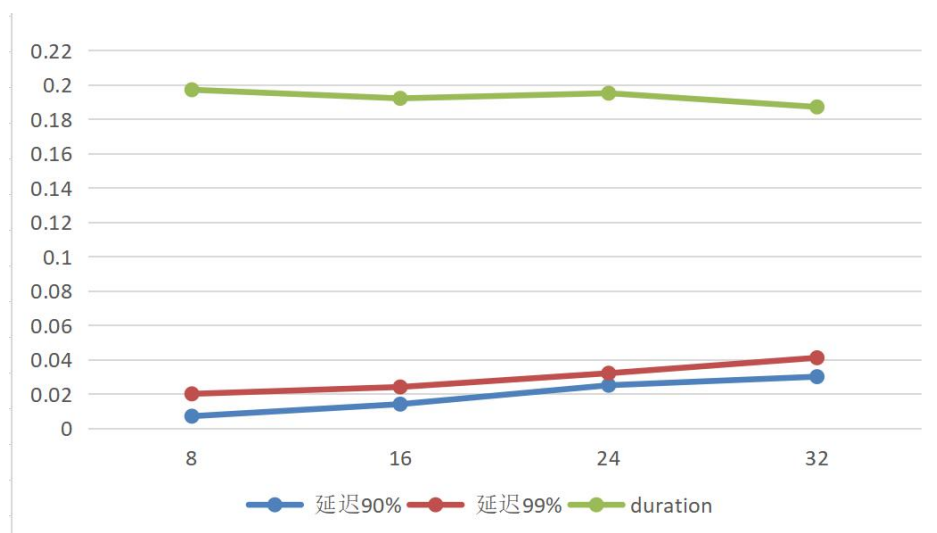
将上述表格制成如下所示图，便于更加直观的观测和分析各参数改变时对读写性能的影响，并得出普适性的结论。

1. (numclient= 8~32 numsamples=256,objectsize=1024)

观察写性能的各项指标

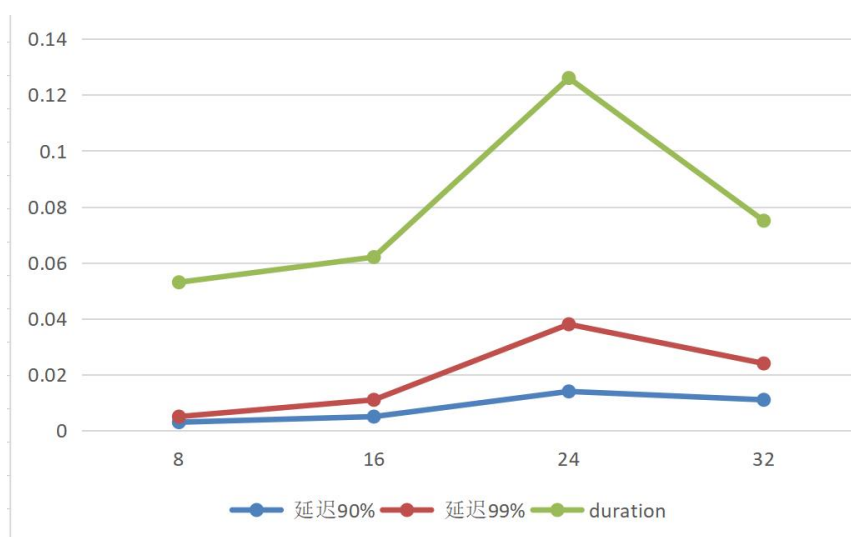


由上图可以看出，随着并发数 client 的增加，吞吐量 Throughput 呈先增长后短暂下降，最后再次增长的趋势。分析可知，当 clients 较小时，随着其增长，吞吐量上升，但并发客户端较多时，也许会出现请求过多，会导致部分请求失败，使得吞吐量有所下降，具体增长还得是看每一次的操作过程。

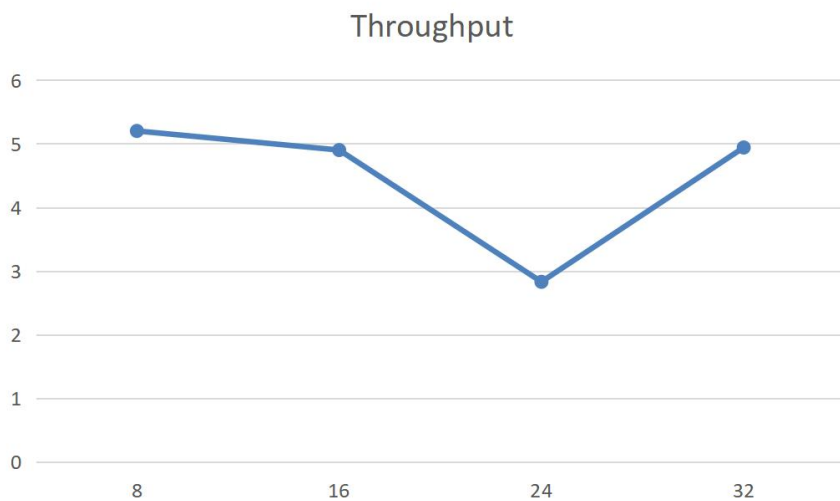


由上图可以看出，随着并发数 client 的增加，90%、99%延迟时间和总用时 duration 整体都呈缓慢的增长趋势，这是因为并发数增加会导致一些用户请求的分配调用问题，需要耗费一定的时间，故实验结果符合实际。

观察读性能的各项指标



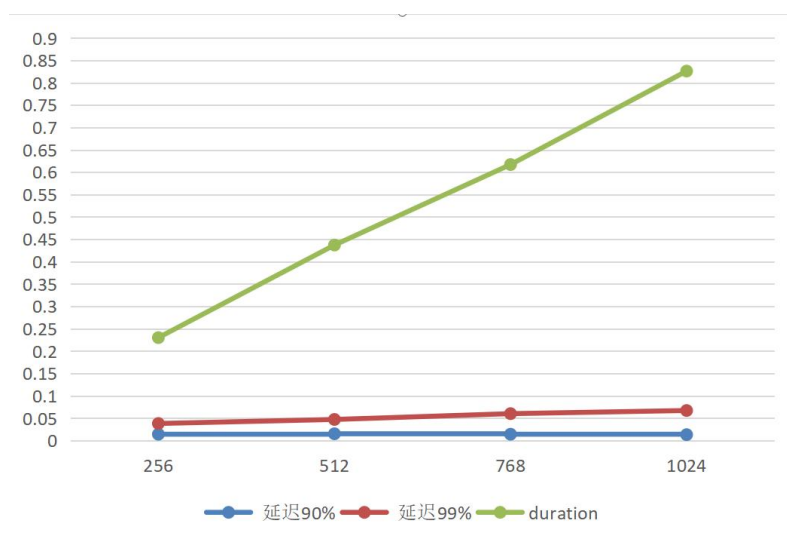
由上图可以看出，在相同配置下，读延迟和总体用时普遍要比写的低很多，这表明在分布式系统中读操作相对于写操作是更容易实现的。并且随着并发数 client 的增加，90%、99%延迟时间和总用时 duration 整体都呈缓慢的增长趋势，这是因为并发数增加会导致一些用户请求的分配调用问题，需要耗费一定的时间，但是在并发数 32 的时候有一丝下降，主要考虑是数据偶然性的问题或较多并发时可能更适合处理相关数据，故实验结果基本符合实际。



由上图可以看出，相同配置下，读吞吐量和要比写的高很多，这表明在分布式系统中读操作相对于写操作是更容易实现的。随着并发数 `client` 的增加，吞吐量 `Throughput` 呈先增长后短暂下降，最后再次增长的趋势。分析可知，当 `clients` 较小时，随着其增长，吞吐量上升，但并发客户端较多时，也许会出现请求过多，会导致部分请求失败，使得吞吐量有所下降。

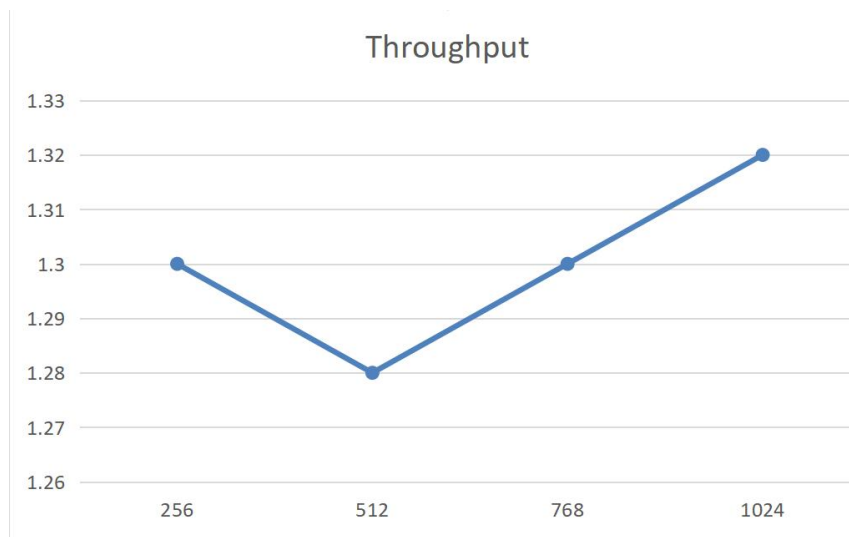
2. （`numclient= 16 numsamples=256~1024,objectsize=1024`）

观察写性能的各项指标

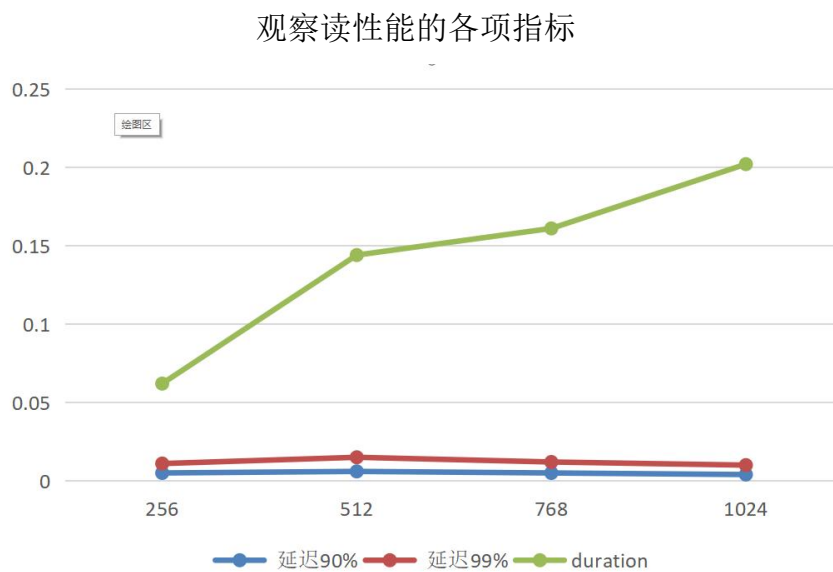


由上图可以看出，随着样本数 `samples` 的增加，90%、99%延迟时间和呈缓慢的增长趋势，而总用时 `duration` 增长趋势较为明显，这是因为样本数增加会导致需要处理的文件增加，需要耗费的时间也固然有所增加，这是符合常理的，故我认

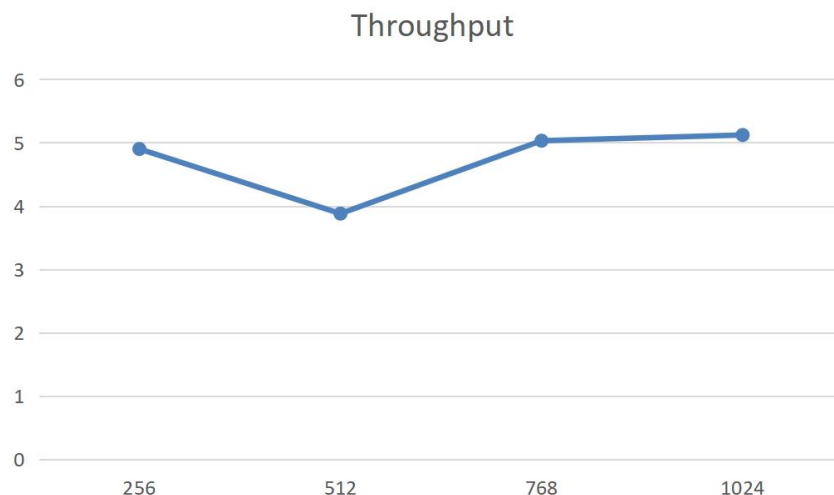
为实验结果是符合实际的。



由上图可以看出，随着样本数量 samples 的增加，吞吐量 Throughput 呈及其微小的变化，经过多次实验认为，样本数量的增加是基本不影响系统吞吐率的，其吞吐量的变化考虑是数据实验偶然性造成的。



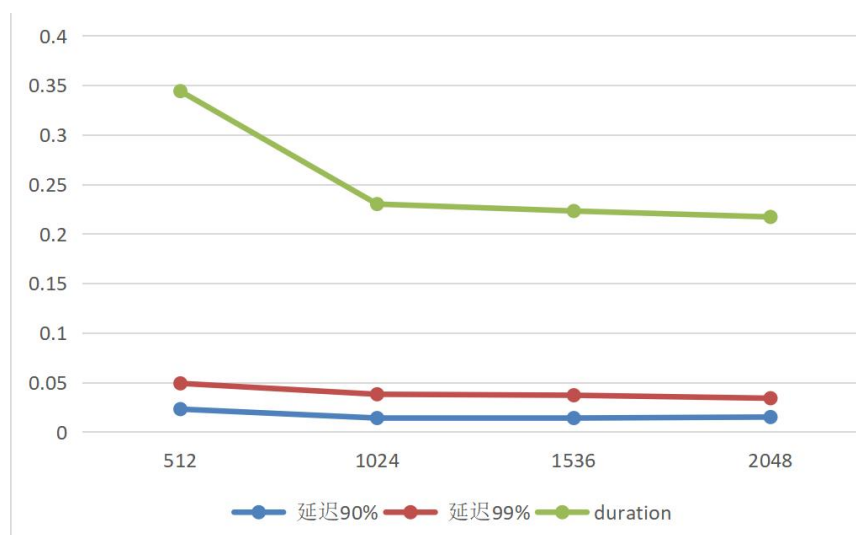
由上图可以看出，在相同配置下，读延迟和总体用时普遍要比写的低很多，这表明在分布式系统中读操作相对于写操作是更容易实现的。并随着样本数 samples 的增加，90%、99%延迟时间和呈缓慢的增长趋势，而总用时 duration 增长趋势较为明显，这是因为样本数增加会导致需要读取的文件增加，需要耗费的时间也固然有所增加，这是符合常理的，故我认为实验结果是符合实际的。



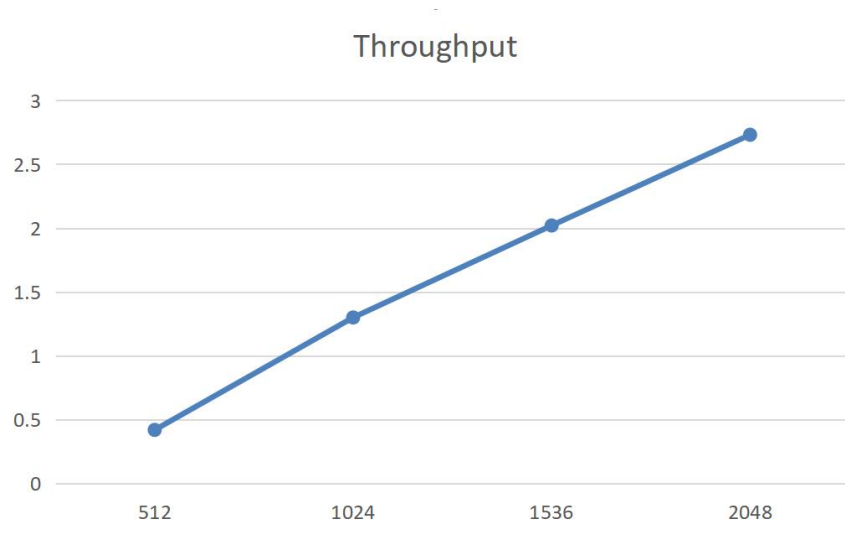
由上图可以看出，相同配置下，读吞吐量和要比写的高很多，这表明在分布式系统中读操作相对于写操作是更容易实现的。随着样本数量 `samples` 的增加，吞吐量 `Throughput` 呈及其微小的变化，经过多次实验认为，样本数量的增加是基本是不影响系统吞吐率的，其吞吐量的变化考虑是数据实验偶然性造成的。

3. (`numclient= 16 numsamples=256,objectsize=512~2048`)

观察写性能的各项指标

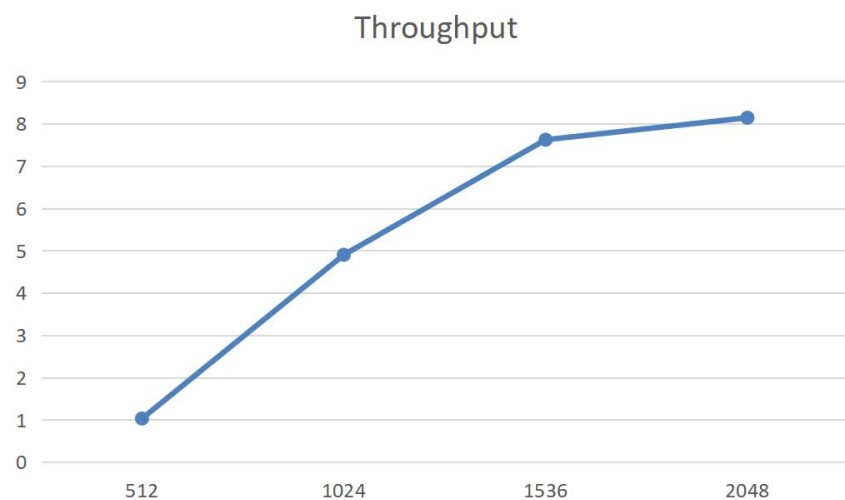


由上图可以看出，随着文件大小 `objectsize` 的增加，90%、99%延迟时间和呈缓慢的降低趋势，而总用时 `duration` 降低趋势为斜率由大到小逐渐平缓，考虑是因为总样本数量不变时，单个文件大小的增加会使得分布式的系统协调工作能力进一步激发，需要耗费的时间也较之前有所减少，故我认为实验结果较为正确。

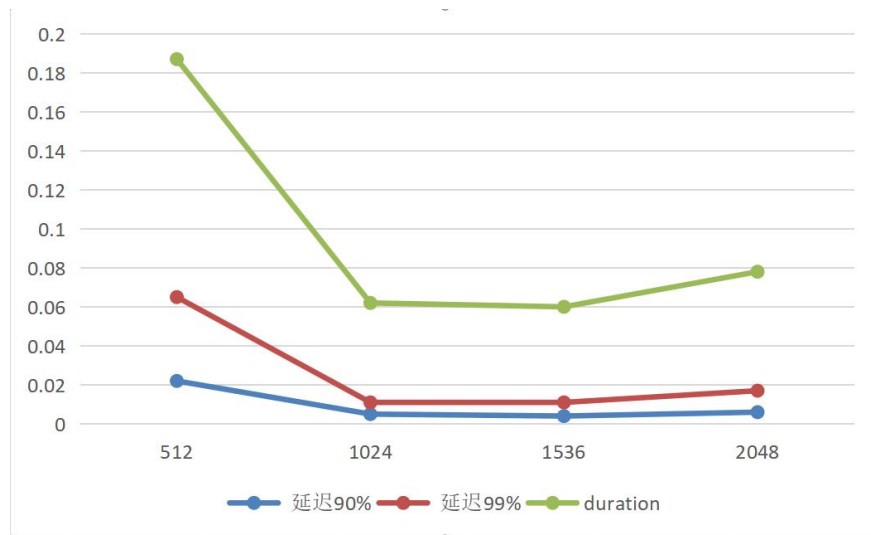


由上图可以看出，随着文件大小 `objectsize` 的增加，吞吐量 `Throughput` 呈较为明显的上升，这符合吞吐量定义的公式，每次处理的文件大小增加意味着每次能够吞吐处理的量变多，故吞吐量增加是符合常理的，实验结果正确。

观察读性能的各项指标



由上图可以看出，相同配置下，读吞吐量和要比写的高很多，这表明在分布式系统中读操作相对于写操作是更容易实现的。并随着文件大小 `objectsize` 的增加，吞吐量 `Throughput` 呈较为明显的上升，这符合吞吐量定义的公式，每次处理的文件大小增加意味着每次能够读取吞吐处理的量变多，故吞吐量增加是符合常理的，实验结果正确。



由上图可以看出，在相同配置下，读延迟和总体用时普遍要比写的低很多，这表明在分布式系统中读操作相对于写操作是更容易实现的。并随着文件大小 `objectsize` 的增加，90%、99%延迟时间和呈缓慢的降低后又有所升高的趋势，而总用时 `duration` 降低趋势为斜率由大到再次缓慢升高，考虑是因为总样本数量不变时，单个文件大小的增加会使得分布式的系统协调工作能力进一步激发，需要耗费的时间也较之前有所减少，但是当单个文件的 `size` 过大时又会导致每次处理所用的时间有所增长。故我认为实验结果较为正确。

六、实验总结

本次实验我之所以选择 Minio Server 和 Minio Client 进行环境搭建，主要是因为 Minio 部署简单：一个 single 二进制文件既是一切，还可支持各种平台；Minio 吃吃海量存储，可按 zone 扩展(原 zone 不受影响)，支持单个对象最大 5TB；兼容 Amazon S3 接口，充分考虑开发人员的需求和体验。低冗余且磁盘损坏高容忍，标准且最高的数据冗余系统为 2(即存储一个 1M 的数据对象，实际占用磁盘空间为 2M)。但在任意 $n/2$ 块 disk 损坏的情况下依然可以读出数据(n 为一个纠删码集合(Erasure Coding Set)中的 disk 数量)。并且这种损坏恢复是基于单个对象的，而不是基于整个存储卷的。读写性能优异。比较适合初学者学习并观察分布式系统的各种性质和特性。

在本次实验过程中我也遇到了许多问题，比如在用 Minio Server 时老产生一些安全相关警报，最后通过自定义端口和账户密码等消除了这些警告。在 minio clietn 的使用过程中出现在建立一个服务并在该服务内创建 bucket 时，只在 minio 所在本地文件夹内进行操作，并没有写到其对应的 Config 文件中，也并没有和 minio server 中指出的链接产生联系，浏览器里的操作台无法观察到新建的服务和 bucket。后面经过施展老师的细心指导终于发现是因为 Minio 系列更新太快，一些老的命令行代码已经不兼容了，故很有可能导致操作无法达到目的，此时应该使用 minio server 中提示用户使用的语句来创建服务和 bucket 才会得到预期的效果。

在分析对象存储性能的部分中，我通过 s3bench 测试工具，更加直观的了解到了各百分比数据的时延，最大延迟，总用时，吞吐量，带宽等各项重要指标的含义，并且通过修改其值为不同的数据，观察到了这些指标的改变对其读写两种性能的影响。且发现其影响还是很明显的。不过唯一小小的遗憾可能就是即便测了很多组数据但是总体测试的数据还是不够多，画出来的折线图可能稍稍存在说服力不足的问题，面对此问题我还是以严谨的太多指出了哪些是比较具有普适性的结论，而哪些可能是由于数据实验偶然性造成的等。

最后真的非常感谢在本次实验中帮助过我的老师和同学们，尤其是施展老师耐心且幽默的指导方式让我印象深刻，而且我个人非常推荐大家选择这门专选课进行学习，这门课是我们学校非常有优势且典型的存储相关课程，能够了解到各种前沿成果以及一些我校在这方面取得的成功等。无论是理论课还是实验课都是实打实能学到东西的。是为想在存储相关领域进一步探索和发现兴趣，并有所提升的同学量身打造的课程

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.