

**2020** 级

《物联网数据存储与管理》课程

**实 验 报 告**

**姓 名 范亚锋**

**学 号 U201814580**

**班 号 物联网2001班**

**日 期 2023.5.16**

**目 录**

[一、实验目的 2](#_Toc23505)

[二、实验背景 2](#_Toc5758)

[三、实验环境 2](#_Toc23566)

[四、实验内容 3](#_Toc23431)

[4.1 对象存储技术实践 3](#_Toc6606)

[4.2 对象存储性能分析 4](#_Toc24911)

[五、实验过程 4](#_Toc6778)

[六、 实验总结 10](#_Toc15328)

[参考文献 10](#_Toc15663)

# 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。

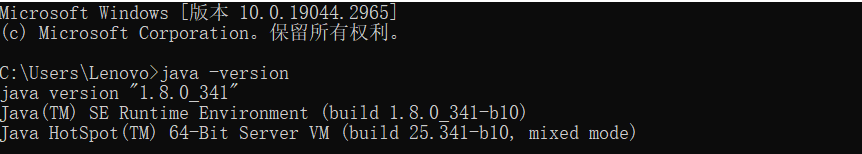
# 三、实验环境

操作系统规格如图1：



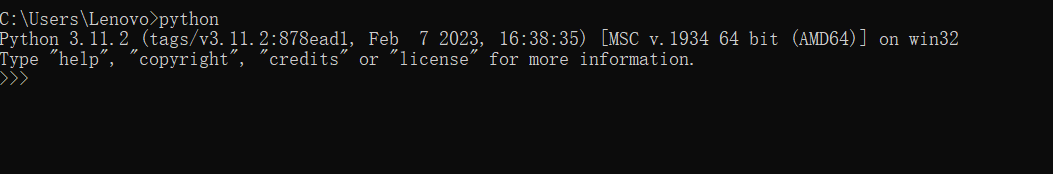
**图1 操作系统规格**

Java版本如图2：



**图2 Java版本**

Python版本如图3：



**图3 python 版本**

# 四、实验内容

本实验在windows环境下完成相应的环境配置，基础的实验环境安装完毕后部署对象存储的服务端、客户端和测试工具，然后启动cosbench测试程序，提交所给的测试文件，观察负载测试的结果，分析吞吐率、延迟等技术指标。

## 4.1 对象存储技术实践

1.在windows下配置java环境。

2.在官网上下载安装对象存储服务端minio和客户端minio client

3. 安装后使用浏览器访问http://127.0.0.1:9000，如果可以访问说明安装成功，否则重新安装

4.下载测试程序COSBench，配置完成后提交负载测试样例，观察运行结果。

## 4.2 对象存储性能分析

1.读写性能对比

2.分析块的大小对运行结果的影响

将负载样例中8kb~1mb work的workers数量改为一致，分析块的大小对存储性能的影响。

3.分析workers数量对运行结果的影响

将负载样例中块的大小改为一致，修改workers的数量，分析workers数量对存储性能的影响。

# 五、实验过程

Minio：

1.安装minio服务器：

从以下 URL 下载 MinIO 可执行文件：

https://dl.min.io/server/minio/release/windows-amd64/minio.exe

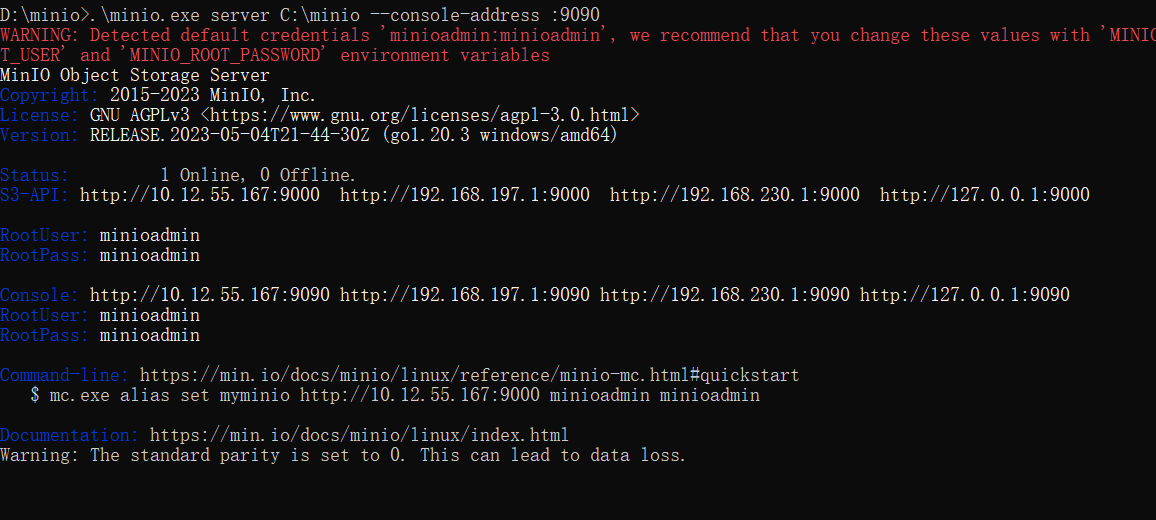
下一步包括运行可执行文件的说明。 不能从资源管理器或通过双击文件来运行可执行文件，可以调用可执行文件来启动服务器。

2.启动

将文件的路径添加到系统。使用以下命令在文件夹中启动本地 MinIO 实例。

.\minio.exe server C:\minio --console-address :9090

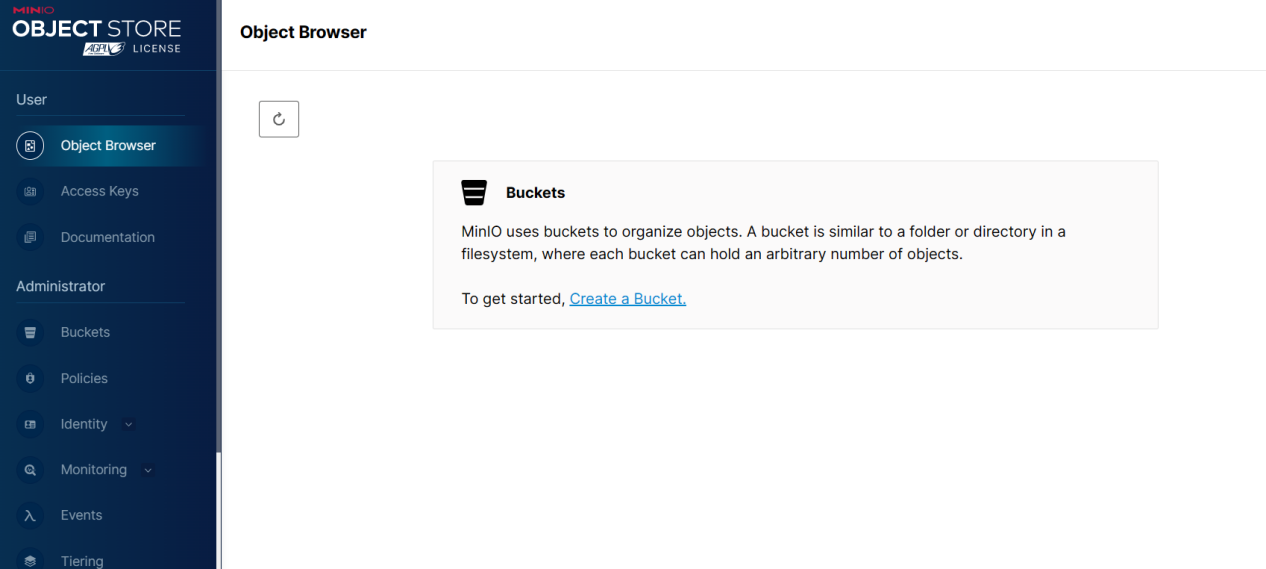
该过程将其输出打印到系统控制台，如图4所示：



**图4 server**

1. 将浏览器连接到minio服务器

通过浏览器访问图4中显示的地址，然后输入账户密码登录，默认账户密码都是minioadmin，登录后的界面如图5所示：



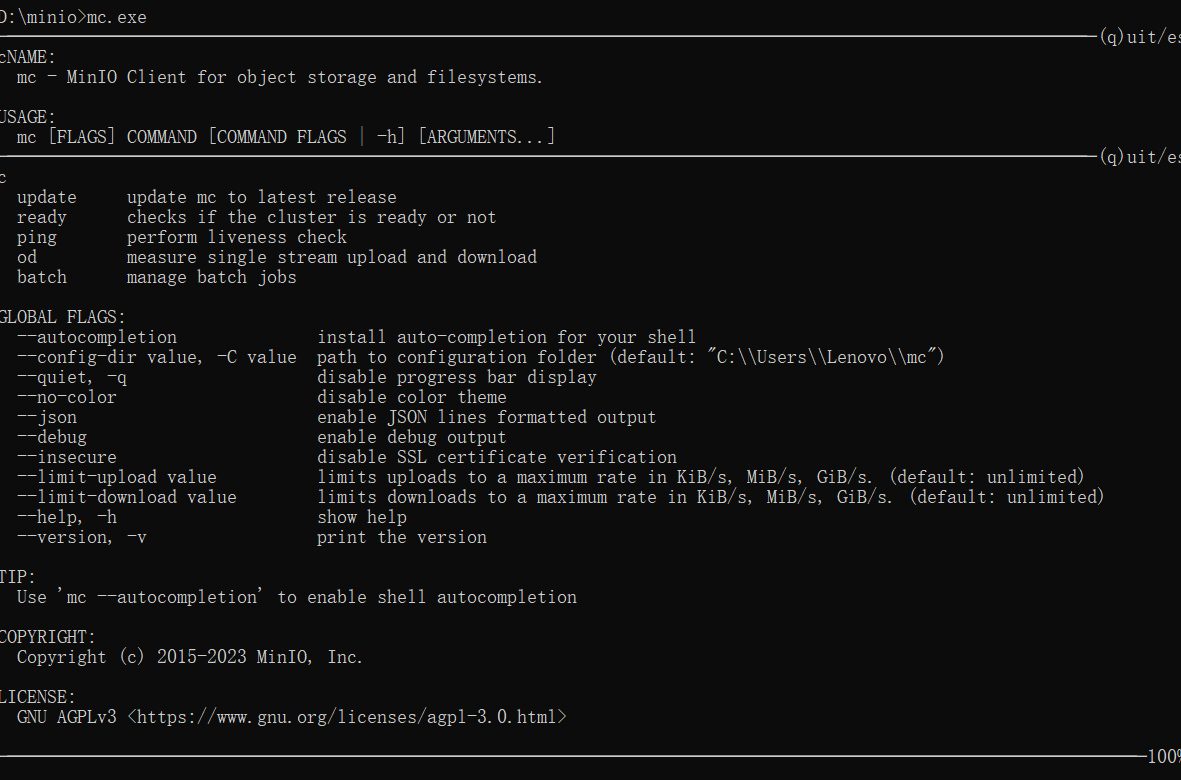
**图5 minio控制台**

1. 安装minio客户端

从以下链接下载适用于 Windows 的独立 MinIO 服务器：

[https://dl.min.io/client/mc/release/windows-amd64/mc.exe](https://dl.min.io/client/mc/release/windows-amd64/mc.exe" \t "https://min.io/docs/minio/windows/_blank)

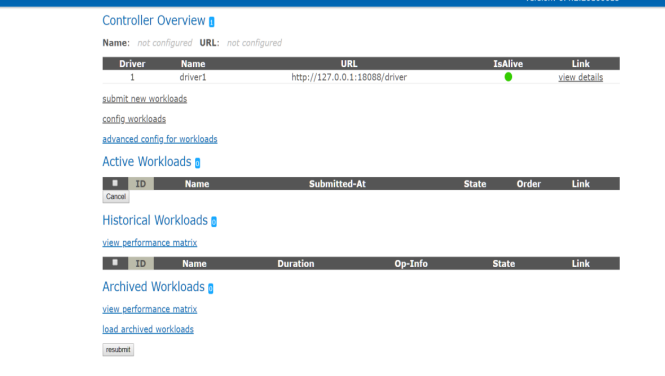
运行该文件，如图6所示：



**图6 运行mc.exe**

5. 在github上下载COSBench压缩包，解压，完成配置后运行start-all.bat文件，该批处理文件打开了driver和controller两个窗口，分别在端口18089和19089监听。

6.运行web.bat，会在浏览器中打开如图7所示的页面：



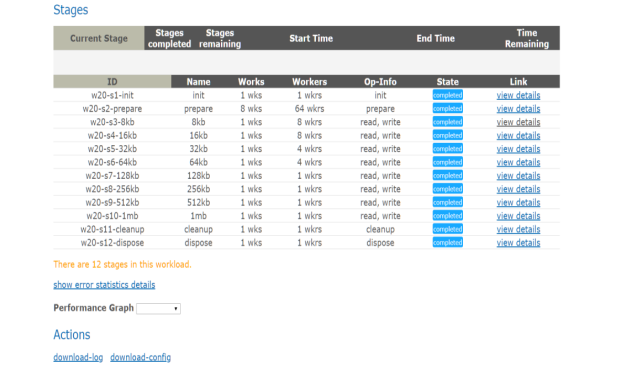
**图7 运行web.bat**

1. 点击submit new workloads进入下一界面，选择需要提交的文件，如图8所示：

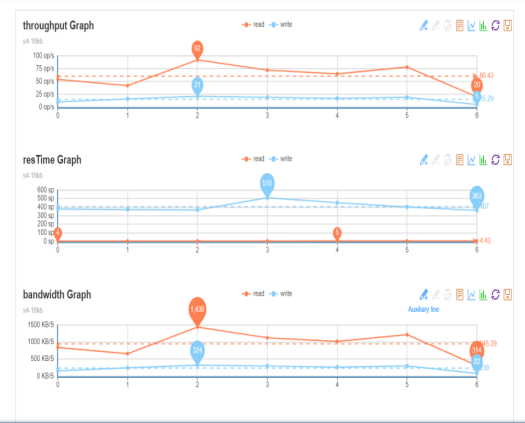


**图8 选择文件**

1. 点击view details查看刚才提交的负载的详细处理结果，处理结果如图如图9和图10所示：



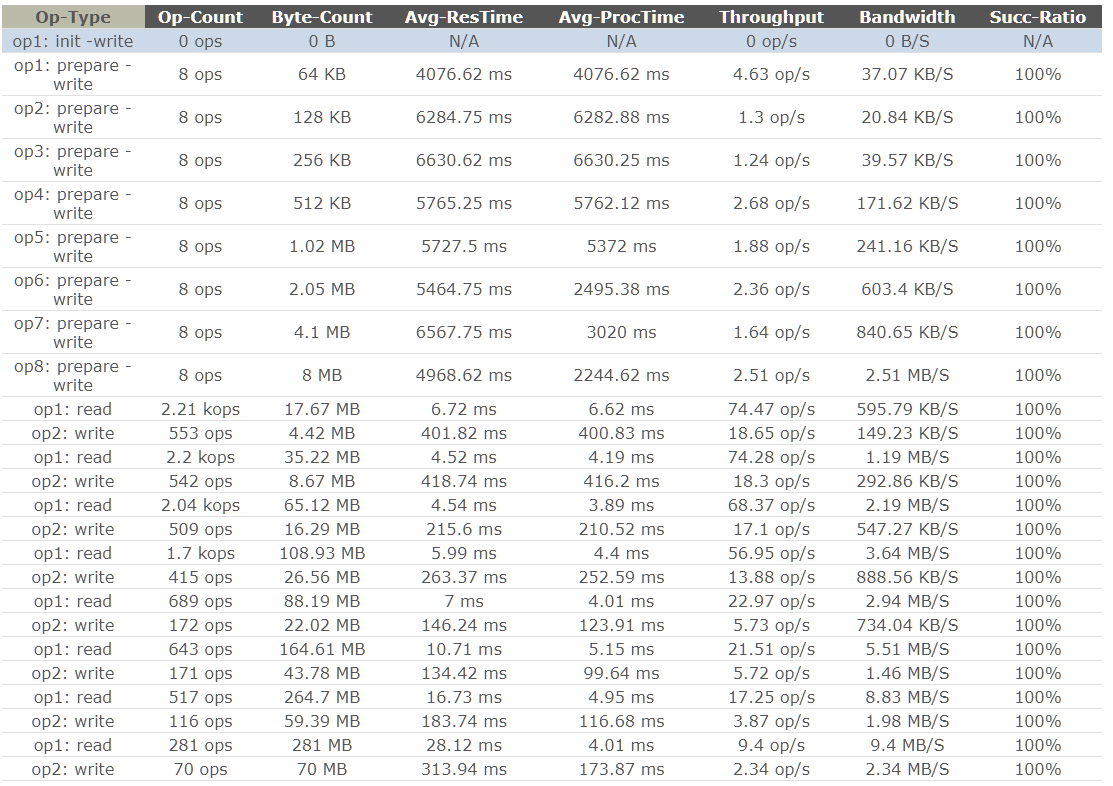
**图9 处理结果（1）**



**图10 处理结果（2）**

1. 读写性能比较

在final result下的General Report,中，可以看到 Op-Type、 Op-count、Byte-count 、 Avg-ResTime 、 Avg-ProcTime 、 Throughput 、 Bandwidth 、 Succ-Radio等信息，如图11。

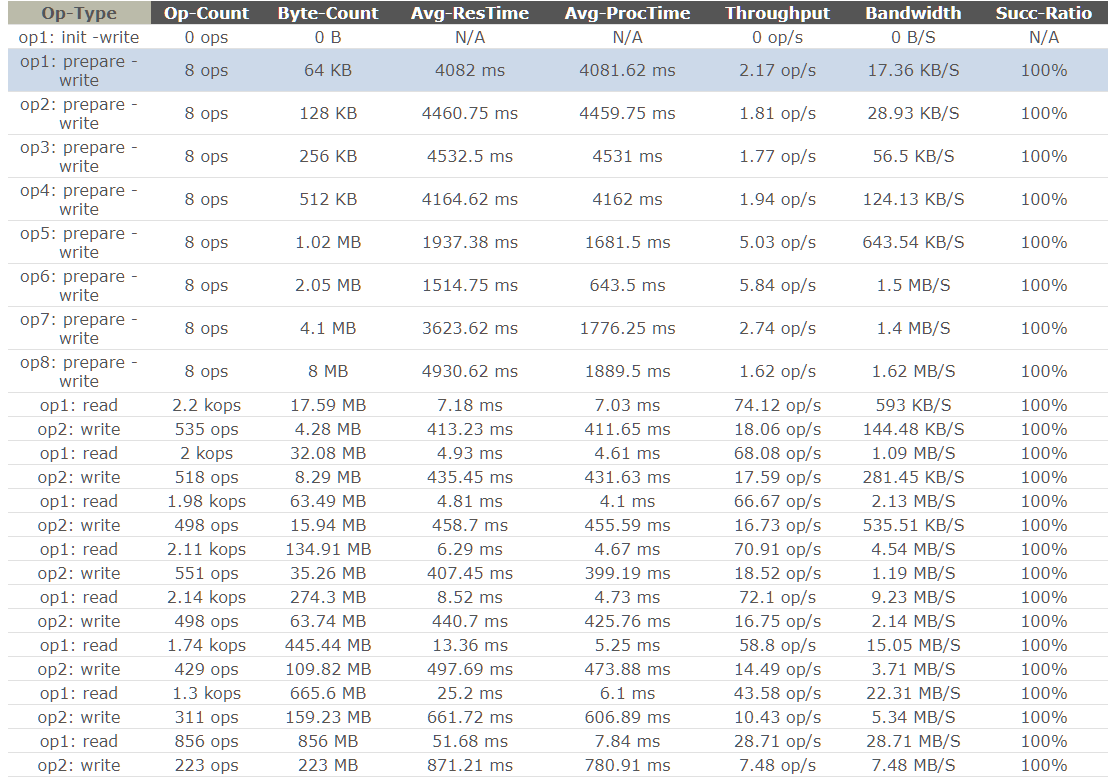


**图11 final result**

从图中我们可以看到，读操作的op数和字节数比写操作更多，但平均处理时间却比写操作平均处理时间要短的多，吞吐量和带宽也是写操作的数倍，说明读性能远远优于写性能。

1. 块大小对处理结果的影响

将负载样例中8kb~1mb work的workers数量改为8，块的大小保持不变，提交修改后的负载样例，运行的最终结果如图12。



**图12 块性能比较**

通过与图11的对比可以发现，调整workers数量后，出现了如下变化：

1. 平均处理时间相比之前的结果有所增加，且块越大，增加的越快；
2. 吞吐量先在原来吞吐量的附近波动，之后迅速减小；
3. 带宽先成倍增加，然后增加速度逐渐减小。

分析：当块较小时，服务器的带宽和处理能力足够，吞吐量能基本保持原来的水平。当块的大小成倍增加时，带宽的增长速度跟不上块的大小的增长速度，服务器的处理能力逐渐达到饱和，无法保持之前的处理速度，导致吞吐量开始下降。且块的大小越大，下降程度越大。

1. 吞吐量与带宽比较

如图13、图14分别为吞吐量变化图和带块变化图：

**图13 吞吐量变化图**

**图14 带宽变化图**

从图中可以看到，相比于写操作的吞吐量变化，读操作的吞吐量变化更大，随着workers数的增加，基本上呈现出先增加后减小的趋势，同时，随着workers数的增加，读操作带宽逐渐增加，且增加速度逐渐减小；写操作带宽基本在原值附近波动，变化较小。综合比较，可以发现当workers数为64时，读写性能最好。

# 实验总结

这次实验遇到的困难主要集中在环境的配置和对象存储系统的搭建等方面。在尝试了分别在Windows和Linux环境下下搭建对象存储系统后，我选择了Windows系统，不过网上关于在Windows系统下配置cosbench的教程比较少，同时由于版本问题也踩了很多坑，好在最终解决了问题。

此次主要进行了以下实验：

1. 块大小对性能的影响实验：我们通过修改块的大小，比较了不同块大小下的性能表现。实验结果表明，块大小对性能有一定的影响，较小的块大小可以提高吞吐量，但同时也会增加带宽的消耗。
2. Workers数量对性能的影响实验：我们通过修改workers数量，比较了不同workers数量下的性能表现。实验结果表明，workers数量对性能有明显的影响，增加workers数量可以提高吞吐量和带宽，但同时也会增加CPU的消耗。
3. 带宽和吞吐量的变化实验：我们通过修改数据的大小，比较了不同数据大小下的带宽和吞吐量的变化。实验结果表明，数据大小对带宽和吞吐量都有明显的影响，数据越大，带宽和吞吐量越高。

通过以上实验，我们可以得到以下结论：

1. 块大小、workers数量和数据大小都会对存储性能产生影响，需要根据实际需求进行调整。
2. 对于需要高吞吐量的场景，可以适当增加workers数量和数据大小。
3. 对于需要高带宽的场景，可以适当减小块大小和数据大小。 综上所述，本次实验通过实践对象存储技术，深入探究了大数据存储与管理的相关技术，并对存储性能进行了分析，对于今后的大数据存储与管理工作具有一定的指导意义。

# 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.