

**《现代数据中心》**

**实验报告**

**课程名称： 现代数据中心**

**专业班级： 硕2110**

**学 号： M202173857**

**姓 名： 方应明**

**指导教师： 施展**

**报告日期： 2021年12月29日**

**计算机科学与技术学院**

目录

[1.实验一：系统搭建 3](#_Toc91939788)

[2.实验二：性能观测 3](#_Toc91939789)

[3.实验三：尾延迟挑战 6](#_Toc91939790)

# 1.实验一：系统搭建

代码管理和仓库：GitHub

系统环境：PYTHON 3.9

服务端：minio

客户端：minio client

完成系统搭建后，运行run-minio.cmd，set MINIO\_ROOT\_USER=hust

set MINIO\_ROOT\_PASSWORD=hust\_obs，完成服务端与客户端的链接，面向对象存储服务开始运行。我们可以做检验，打开浏览器窗口，输入127.0.0.1:9000，然后输入 use 和passport，可以看到如下图1-1所示界面，表示成功完成系统搭建任务，开始运行面向对象存储服务。

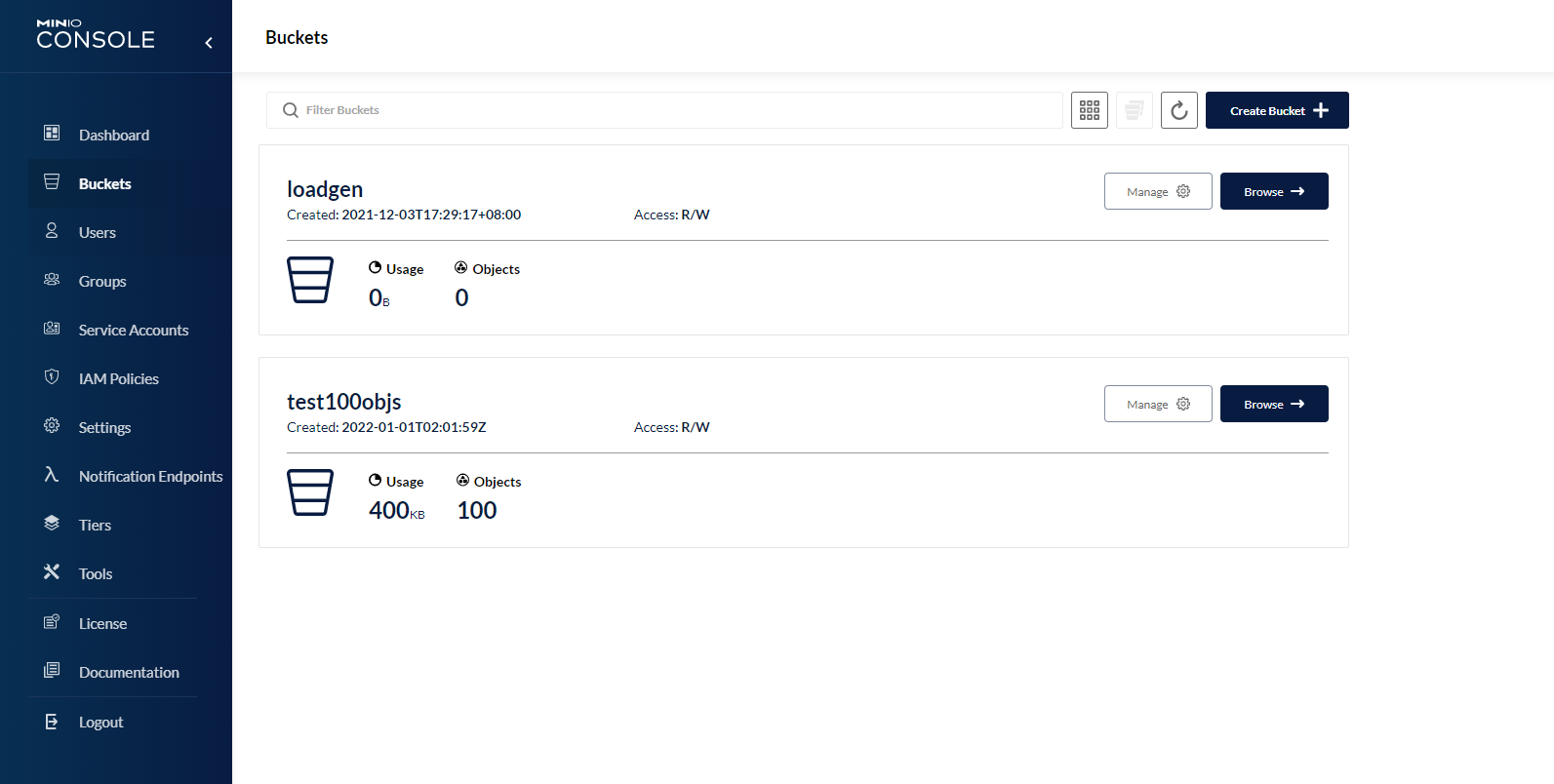


图1-1. 面向对象存储服务

# 2.实验二：性能观测

测评工具S3 Bench

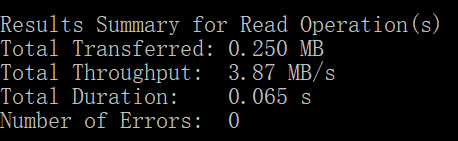
吞吐率原指一个业务系统在单位时间内提供的产量（或服务量）。在计算机或数据通信系统，指的是单位时间内通过某通信信道（a communication channel）或某个节点成功交付数据的平均速率，通常以每秒比特数（bps， bits per second ）为单位。吞吐受带宽限制，带宽越大，吞吐率的上限才可能越高。

带宽在数字设备中，带宽指单位时间能通过链路的数据量。通常以bps来表示，即每秒可传输之位数。带宽越大，其传输能力就越强。

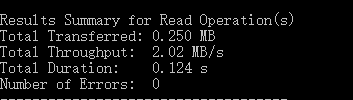
延时，表示请求数据包发送后，收到对端响应，所需要的时间延迟。不同的场景有着不同的含义，比如可以表示建立 TCP 连接所需的时间延迟，或一个数据包往返所需的时间延迟。

测试case：

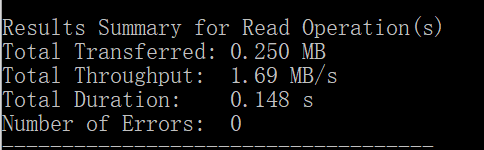
  -numClients=8 ，-numSamples=1024，-objectSize=1024，时候，测试数据如下：



-numClients=16，-numSamples=1024，-objectSize=1024，时候，测试数据如下：



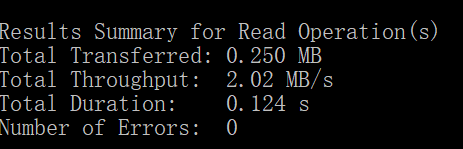
-numClients=32，-numSamples=1024，-objectSize=1024，时候，测试数据如下：



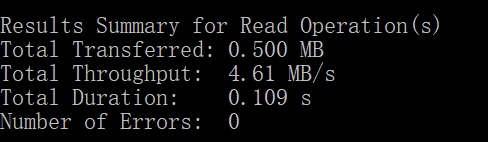
我们可以看到，随着numclient数目的增加，总的throughput和duration都在减少。

Case：

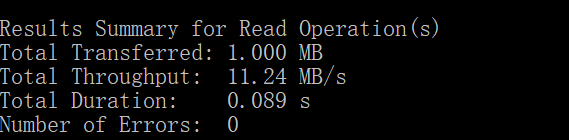
  -numClients=8 ，-numSamples=256，-objectSize=1024，时候，测试数据如下：



-numClients=8，-numSamples=256，-objectSize=2048，时候，测试数据如下：



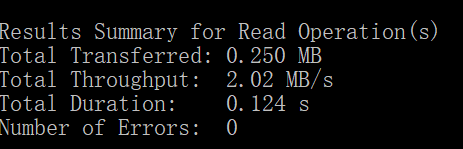
-numClients=8，-numSamples=256，-objectSize=4096，时候，测试数据如下：



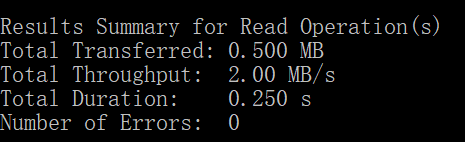
我们可以看到，随着objectsize数目的增加，总的throughput在增加，duration在减少。

Case：

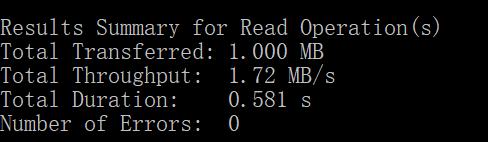
  -numClients=8 ，-numSamples=256，-objectSize=1024，时候，测试数据如下：



-numClients=8，-numSamples=512，-objectSize=1024，时候，测试数据如下：



-numClients=8，-numSamples=1024，-objectSize=1024，时候，测试数据如下：



我们可以看到，随着objectsize数目的增加，总的throughput在增加，duration在减少。

可以看到，当加大负载也就是增多numberclient和objectsize时候，尾大不掉的情况一直存在，随着负载增加，读写的总数值都在增加，最长最短读写时间增加、相同百分比的读写完成率所用时间增加，尾延迟现象明显。。

# 3.实验三：尾延迟挑战

在尾延迟挑战实验部分，首先准备负载, 按照几种不同请求到达率设置，在初始化本地数据文件以后，发起请求并计算系统的停留时间，按照请求到达率来限制并执行跟踪请求，其中，分辨按照100ms内不超过两个请求，四个请求，八个请求进行设置。完成到达率设置之后，按照到达率发起请求，并采集延迟，将延迟保存在CSV文件中，最后通过运行latency-plot.ipynb进行画图。

其中我们可以看到，其一百个请求的延迟分布情况如下图3-1所示。我们可以看到，虽然平均延迟满足用户需求，但是总是有例外，在平均延迟之上，我们看到的平均延迟是被平均之后的指标。在排序后，我们可以看到一个很明显的长尾分布图，大部分情况下延迟比较低，但是少部分情况会有较高延迟。

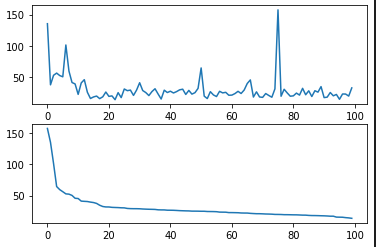


图3-1 一百个请求的延迟分布情况

最后，我们还画出百分比延迟的图，并与绘制的排队论的模型进行拟合，可以看到阶梯状的蓝色的线，本质是一个指数的累计概率分布，表示对应的百分位的出现可能更小或者更大，我们发布对冲请求的合理时机与分布有关，而与平均延迟，延迟的最大值和最小值无关。我们可以看到多达百分之七十五的请求在十毫秒左右完成，对于百分之九十五的请求在七十毫秒左右完成。

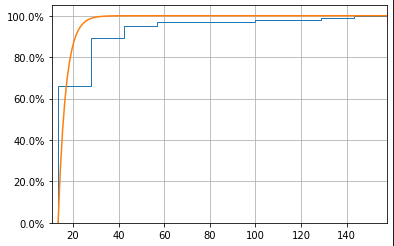


图3-2 百分比延迟图

为了降低尾延迟带来的影响，我们加入对冲请求，当请求超过一定数值时候重新发送请求，设置界限为220ms。其中加入对冲请求后延迟分布情况和百分比分布情况如下图3-3，图3-4所示。我们可以看到，加入对冲请求后，尾延迟现象得到明显抑制，百分之九十请求都在150ms内发出。

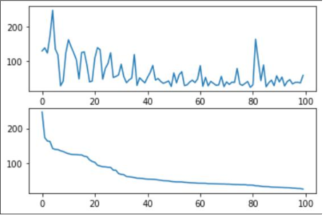


图3-3 对冲请求后延迟情况

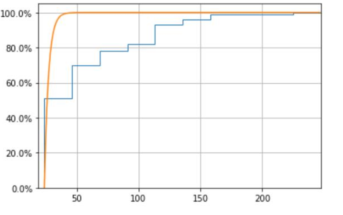


图3-4 对冲请求百分比延迟情况