对象存储实验

**实验一：系统搭建**

服务端：Minio

因为minio服务端带有管理操作系统，故不需要额外安装客户端。

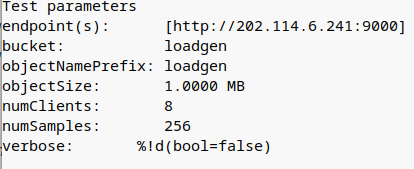
**实验二：性能观测**

工具：S3 Bench

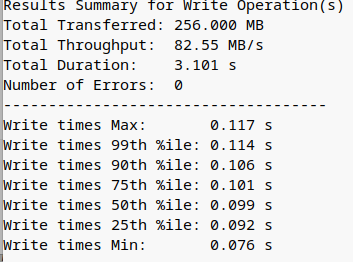
步骤：

1. 安装go环境
2. 安装S3 Bench：go get -u github.com/igneous-systems/s3bench
3. 测试

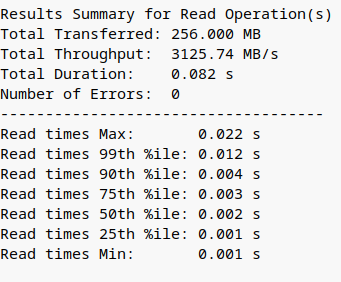
对于步骤3的测试，我选择的测试参数如下图：



得到的写操作指标如下：



读操作指标如下：



思考：

1. 对象尺寸如何影响性能？

直观上给我们的感觉是对象尺寸越大性能就越差，对象存储有两个主要的性能指标，如上的测试中我们可以知道是吞吐量和延迟，我通过选取不同的对象尺寸进行性能测试，发现了对象尺寸越大，吞吐量也会随之增加，延迟在对象尺寸较小时（KB~MB）增加幅度不大，可知当对象尺寸在一定范围内的增加不会明显影响性能，而当尺寸过大时可能需要进行拆分存储等操作，影响了性能。

1. I/O延迟背后的关键影响要素？

(1) 因为对象存储将元数据与数据分别存放，所以请求处理过程中数据定位和元数据读取涉及访问存储设备是耗时操作。

(2) 对象存储系统使用基于事件驱动的并发处理架构,请求到达存储服务器后需要等待被接受。

1. 如果客户端爆满将怎样？

并发数高则吞吐量大，但相应的如果超过了服务端所能同时处理的最大并发数，则势必有客户的请求需要等待处理，就会导致部分客户的延迟过高（尾延迟问题）。

1. 横向扩展系统（Scaling Out）效果如何（向系统中追加更多存储服务器）？

横向扩展系统传统意义上的技术更新。当硬件需要升级时，就只是增加一个新的对象存储节点的问题，一般是通过标准以太网进行连接。无需数据迁移。除了自修复算法，纠删码或者多路拷贝镜像都将重新创建新节点上的数据。随着容量的增加，数据将自动分布于所有的对象存储节点。

**实验三：尾延迟挑战**

尾延迟问题：假设有90%的请求都会在某个时间限之前得到回应，而剩下的10%就有可能发生尾延迟问题，即在超出两倍的时间限后才得到响应，所以与其继续等待请求，不如再发送一个请求，这样就有99%的可能性在规定时间内得到响应，避免尾延迟问题。

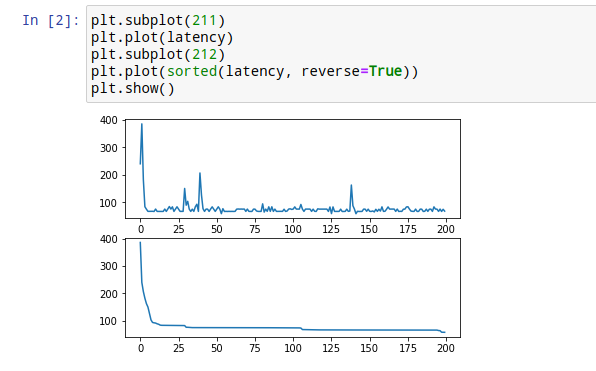
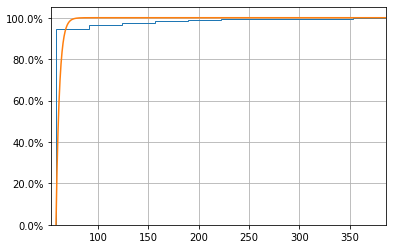
解决方案：对冲请求

实验设计：

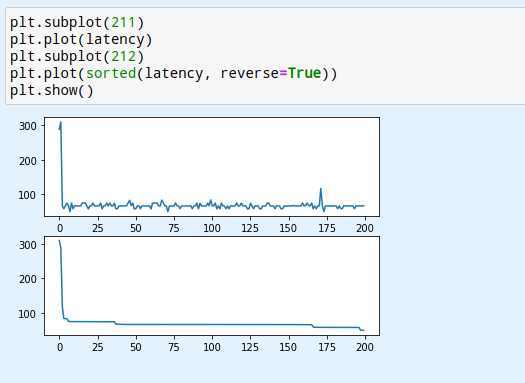
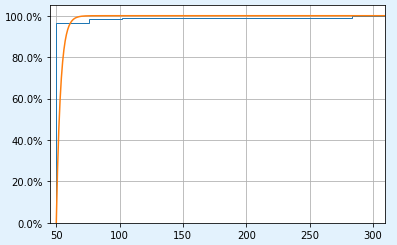
1. 在设定好实验参数后（请求数=200，对象尺寸=1024\*64）,进行测试，用一个文件latency.csv保存每个请求完成的时间，之后画出延迟的分布图，并用排队论拟合实测数据，便可以从图中读取大部分请求会在哪个时间限内完成。
2. 模拟对冲的环境，我们将请求数翻倍即为400（每两个请求负责同一个对象，即第一个请求立即发送，第二个请求我们设置其在等待上诉得到的时间限后再发送，之后比较两者花费的时间，取更短的时间即为最后的延迟）对象尺寸一致，我们用400个线程来完成上诉请求的发送。

**实验结果如下**

步骤一：如下图所示（忽略最开始几个请求的高延迟），我们可以看到大部分请求再100ms内都可以完成，但也有少数几个请求的延迟达到了接近200，而这可能是用户接受不了的延迟，在短短两百的请求里就有几个用户可能接受不了的延迟，所以我能需要进行改进。

实现对冲请求的步骤二：根据步骤一得到的结果，我们选取时间限为70ms，所以附加线程会在70ms的延迟后发送第二个请求，得到的结果如下图所示（忽略最开始几个请求的高延迟），我们看到几乎全部的请求都可以在100ms的延迟里完成，仅有的一个也只是比100ms稍高，这有效的解决了尾延迟问题。

**总结：**通过这三个实验，我初步了解了对象存储系统的搭建过程，了解对象存储的优缺点，同时切身实地的感受尾延迟问题，并设计实验用对冲请求的方式来试图解决尾延迟问题，实际系统中对冲请求的实现大概会涉及到非阻塞IO以及线程间通信等，我采用的简单模拟方法确也能够观察到尾延迟问题变得不再那么明显。