

**数据中心技术实验报告**

|  |  |
| --- | --- |
| **学院** | **武汉光电国家研究中心** |
| **班级** | **2021硕** |
| **小组** | **无** |
| **姓名** | **Sheng** |
| **学号** | **M202173484** |

2021 **年** 1 **月** 7 **日**

目录

[实验一 环境准备 1](#_Toc92110186)

[一、 实验环境 1](#_Toc92110187)

[二、 实验内容 1](#_Toc92110188)

[三、 实验步骤 2](#_Toc92110189)

[实验二 性能评估 3](#_Toc92110190)

[一、 实验环境 3](#_Toc92110191)

[二、 实验内容 3](#_Toc92110192)

[三、 实验步骤 4](#_Toc92110193)

[实验三 尾延迟 5](#_Toc92110194)

[一、 实验环境 5](#_Toc92110195)

[二、 实验内容 5](#_Toc92110196)

[三、 实验步骤 6](#_Toc92110197)

# 实验一 环境准备

### 实验环境

操作系统：Windows 11 22000.376 with Hyper-V

软件：Vagrant 2.2.19

### 实验内容

参照课程实验指导（<https://github.com/cs-course/obs-tutorial>），在本实验中需完成对象存储的服务端、客户端配置，并简单测试功能的正确性。在本实验系列中，我选择采用Ceph软件体系搭建对象存储服务器，其中radosgw用于提供类Amazon S3的对象存储接口。

整个实验在虚拟机集群中完成，采用Vagrant工具自动构建。根据Ceph组件的功能，我将服务器分为两类，master和worker，其中master节点负责逻辑控制与交互，worker节点负责数据存取。在本实验中，配置1个master节点和3个worker节点，所有来自客户端的数据交互仅与master进行。Vagrant配置文件参见我的GitHub repo

（<https://github.com/a5467021/ceph-vagrant-rocky8>）。



**图2.1 实验中的Ceph集群架构及服务组件部署情况**

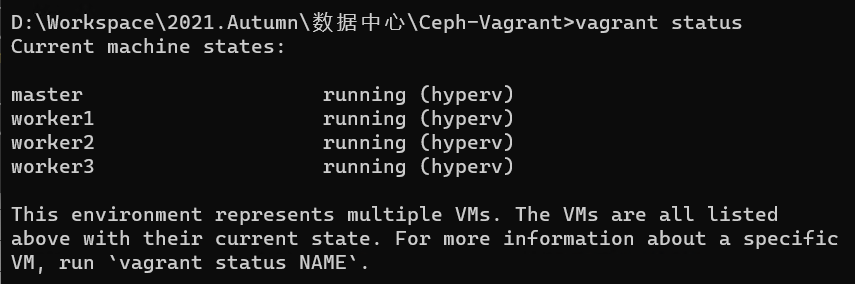
搭建Ceph服务端采用的方式是使用官方的Ansible自动化工具ceph-ansible（<https://github.com/ceph/ceph-ansible>），该工具可通过简单的hosts文件配置自动完成在指定节点上的指定组件配置，并配置网络互联。

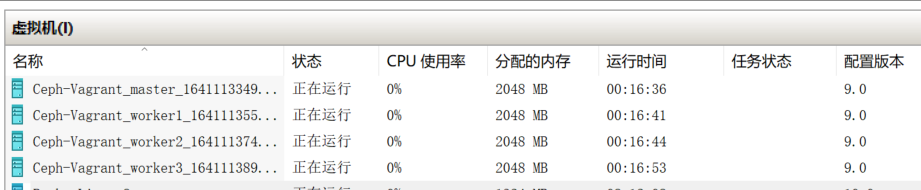
客户端选择使用osm作简单的登陆、创建/查询桶操作以验证环境。

### 实验步骤

1. 使用Vagrant创建虚拟机集群（需要管理员权限）；

|  |
| --- |
| > git clone <https://github.com/a5467021/ceph-vagrant-rocky8>  > cd ceph-vagrant-rocky8  > set VAGRANT\_EXPERIMENTAL=disks  > vagrant up |

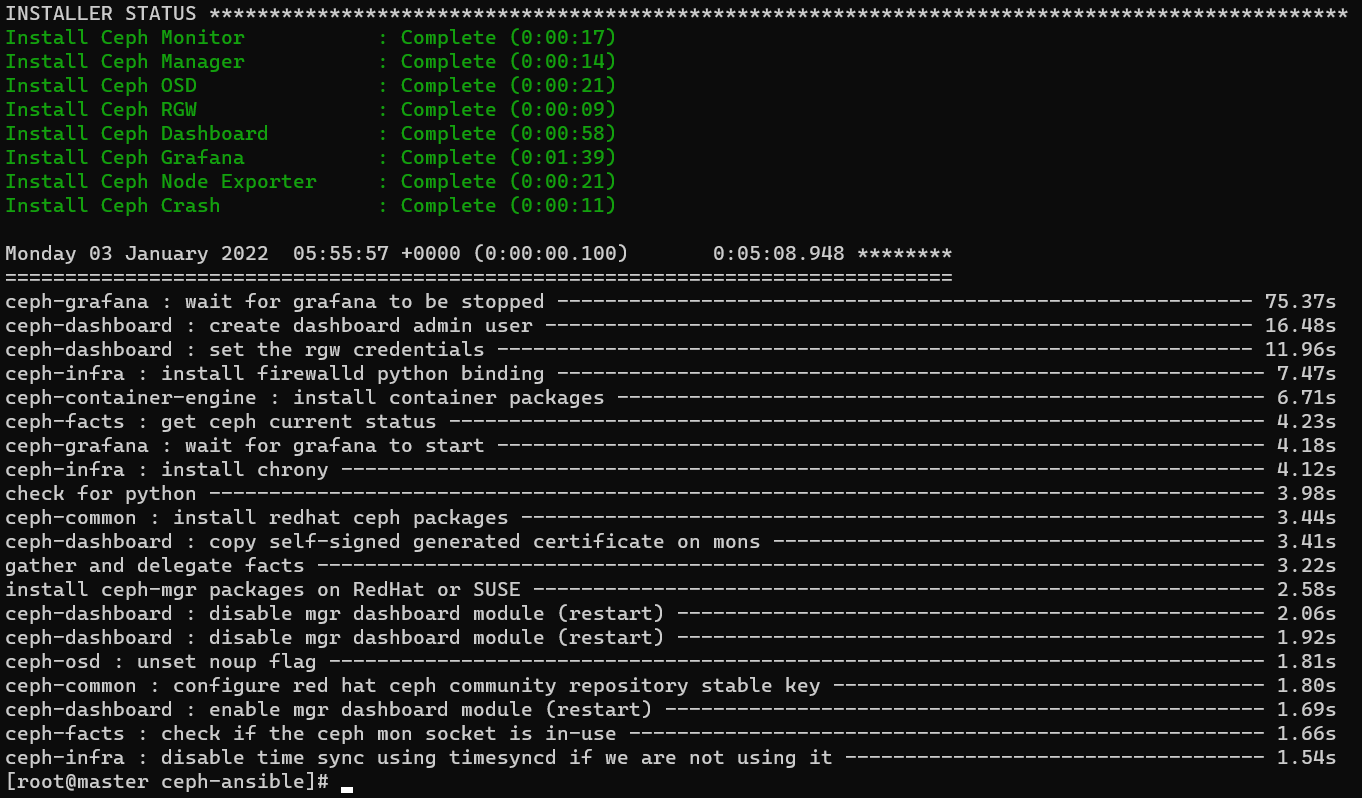




**图3.1 创建后的虚拟集群**

1. 使用ceph-ansible将Ceph部署至虚拟集群；

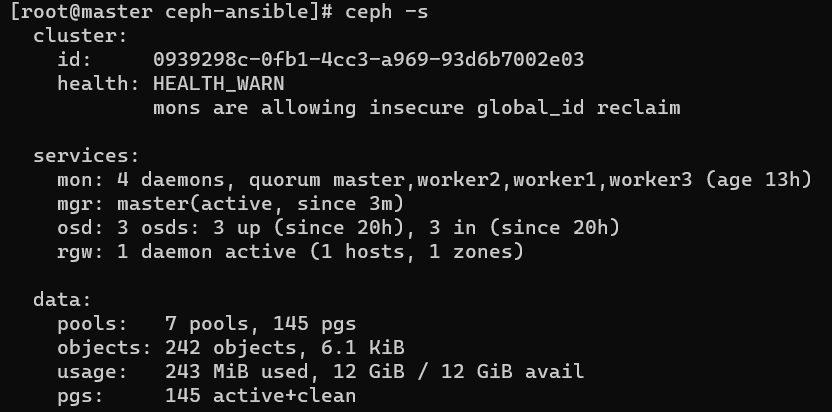
|  |
| --- |
| > vagrant ssh  $ sudo -i  # cd /opt/ceph-ansible  # ansible-playbook site.yml |

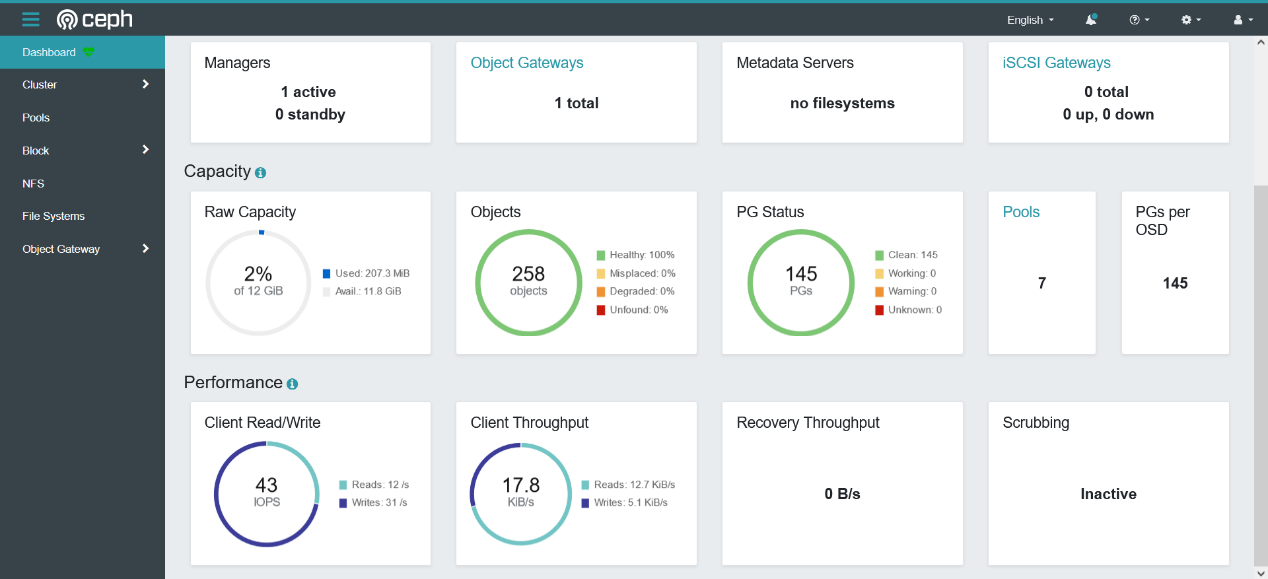


**图3.2 ceph-ansible部署完成**

1. 验证部署情况；

|  |
| --- |
| # ceph -s |

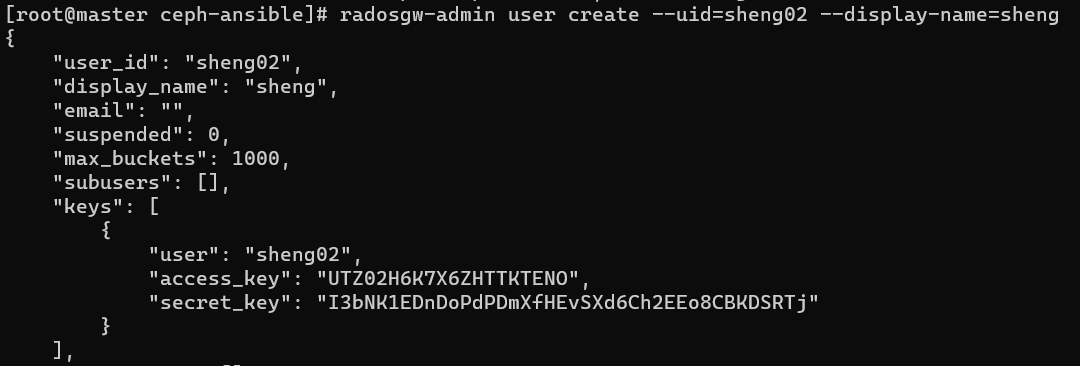




**图3.3 Ceph集群状态**

1. 创建radosgw用户，配置访问权限；

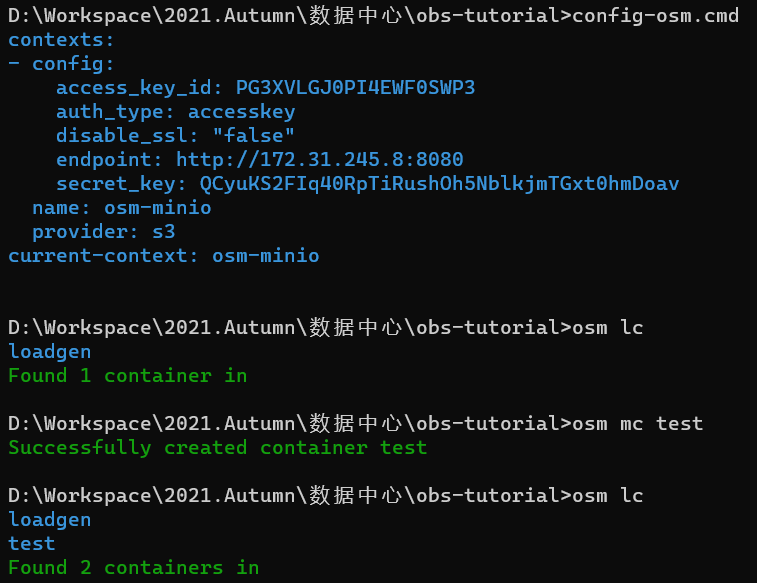
|  |
| --- |
| # radosgw-admin user create --uid=sheng --display-name=sheng |



**图3.4 使用radosgw-admin生成对象存储访问密钥**

1. 配置osm，验证桶创建和桶查询操作。

|  |
| --- |
| > config-osm.cmd  > osm mc test  > osm lc |



**图3.4 使用radosgw-admin生成对象存储访问密钥**

# 实验二 性能评估

### 实验环境

操作系统：Windows 11 22000.376 with Hyper-V

软件：Vagrant 2.2.19

### 实验内容

参照课程实验指导（<https://github.com/cs-course/obs-tutorial>），在本实验中需使用符合Amazon S3接口的工具，测试不同并发数、不同文件大小情况下的对象存储用户端性能。

### 实验步骤

1. 修改文件latency-collect.ipynb（参见code文件夹下文件），运行代码收集性能数据，并保存到csv格式的数据文件；



**图2.1 使用基于boto3库的python脚本访问Ceph对象存储**

1. 修改文件latency-plot.ipynb（参见code文件夹下文件），解析并可视化收集到的性能数据（图参见实验三）。

# 实验三 尾延迟

### 实验环境

操作系统：Windows 11 22000.376 with Hyper-V

软件：Vagrant 2.2.19

### 实验内容

参照课程实验指导（<https://github.com/cs-course/obs-tutorial>），依照常见的尾延迟处理方法，设计对冲请求和关联请求两种减小预期延迟的算法，并通过测试验证其性能提升。

### 实验步骤

1. 熟悉对冲请求和关联请求两种算法的基本思路：对冲请求对同一资源并发地发送多个请求，以完成最快的请求作为请求的结束；关联请求一次只对资源同时发送一个请求，但在请求耗时超过阈值（通常选取95%分位数）时，中断当前请求，重新发送新请求；
2. 编写代码。具体代码位于latency-collect.ipynb（参见code文件夹下文件）中；
3. 测试。修改实验二中测量延迟的方法，使用本实验中自己编写的请求算法替代，并记录访问延迟；
4. 借助latency-plot.ipynb（参见code文件夹下文件）可视化分析请求延迟信息，对算法性能做出评估。可视化结果参见图3.1。为了更好地展示低延迟下的延迟分布情况，概率密度函数在x轴采用了对数坐标轴。

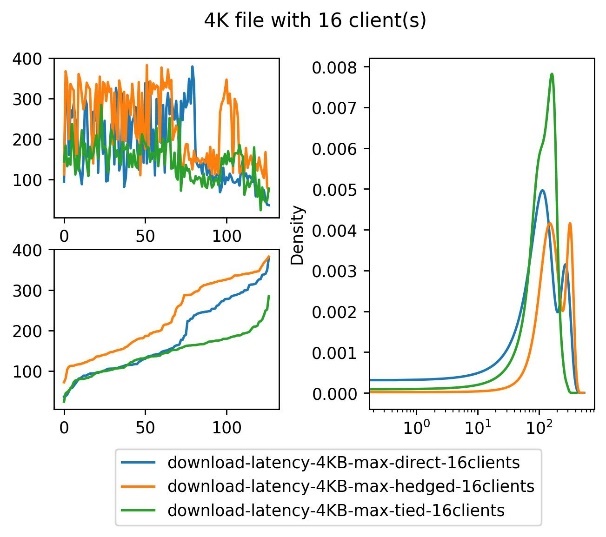
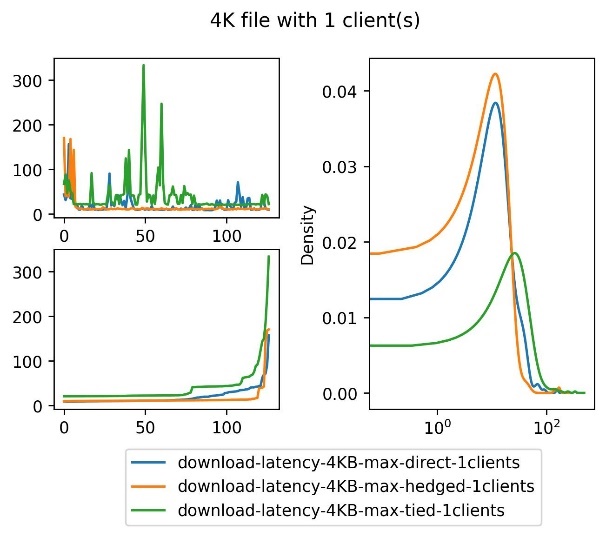
可以总结出如下规律：

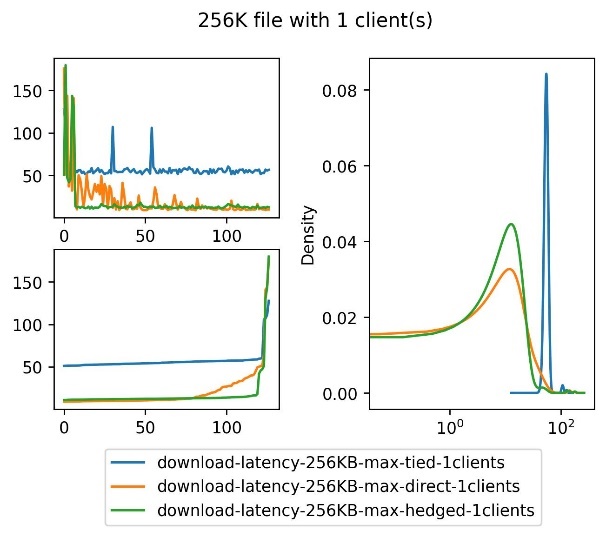
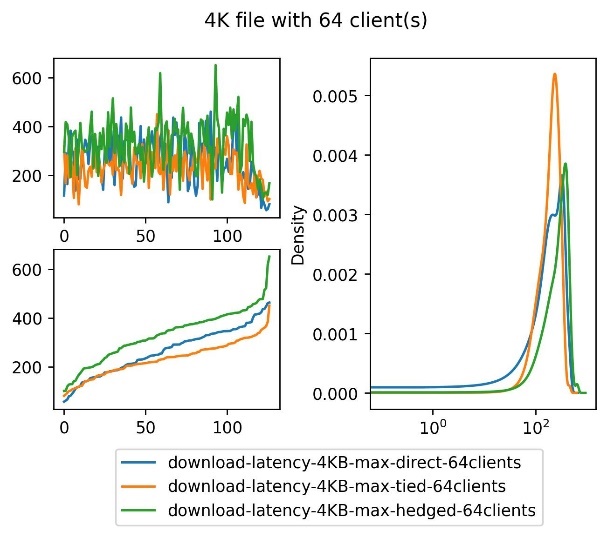
1. 对象的请求延迟与文件大小、并行客户端数量呈正相关；
2. 在文件小、连接数少的情况下，对冲请求具有最低的请求时延；
3. 在文件大、连接数多的情况下，关联请求具有最低的请求时延。

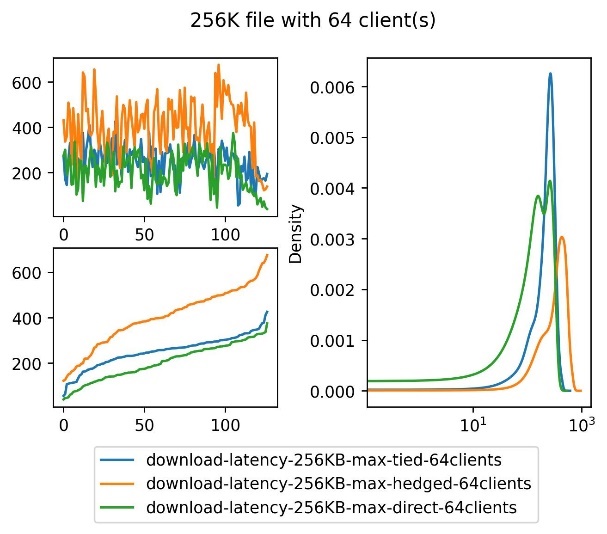
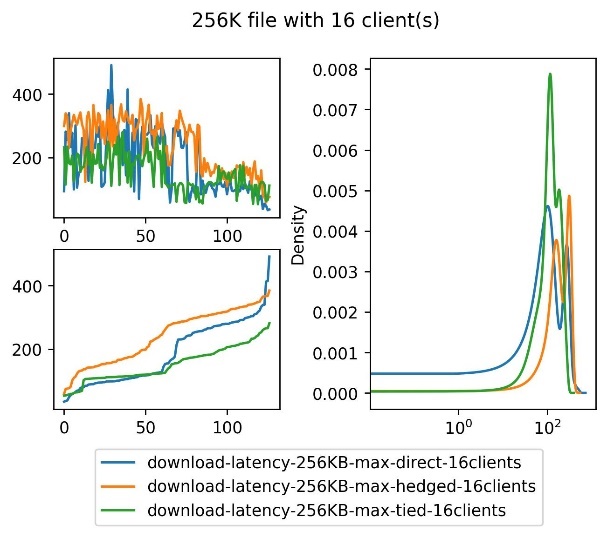
根据以上规律，我们可以得出结论：对冲请求和关联请求两种策略在各自合适的场景下具有优于直接请求的性能。而为何两种策略需要一定的条件才能发挥出更好的性能，我认为有以下原因。

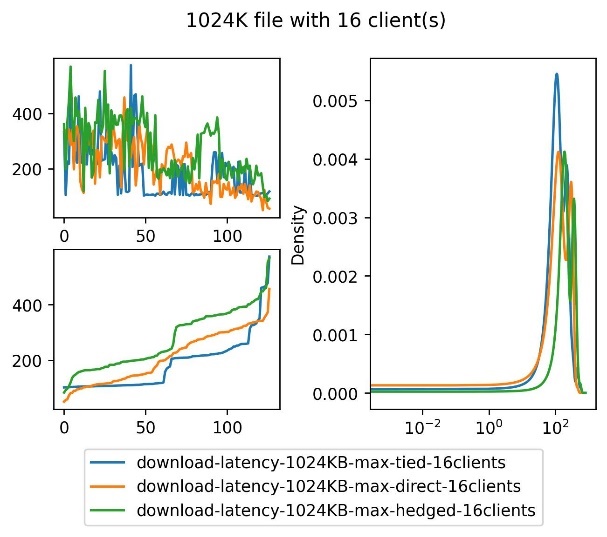
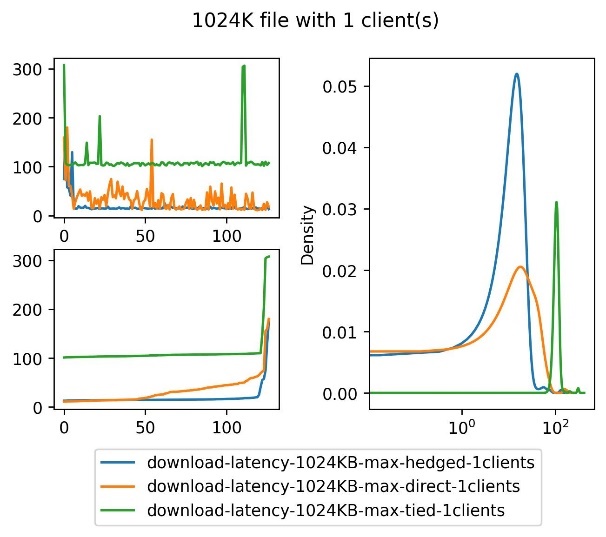
首先是性能原因。服务器的物理资源有限，就意味着不论是数据处理能力还是带宽都是有限的，对冲请求策略通过牺牲吞吐量的方式来换取更低的延迟，而在吞吐量已经接近上限的情况下显然无法再增加，因而在已有较高负载的系统下性能没有提升，甚至因为冗余请求消耗带宽的原因，反而降进一步加剧了现有请求的延迟。在低负载情况下，服务器资源处于空闲状态，此时增加的额外请求就不但不会拖慢原有请求，还可避免随机性遇到的服务器高延迟回复，因而可以达到降低延迟的效果。

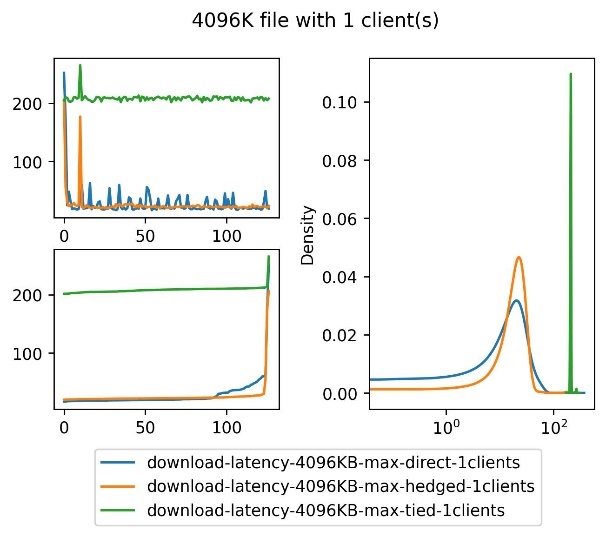
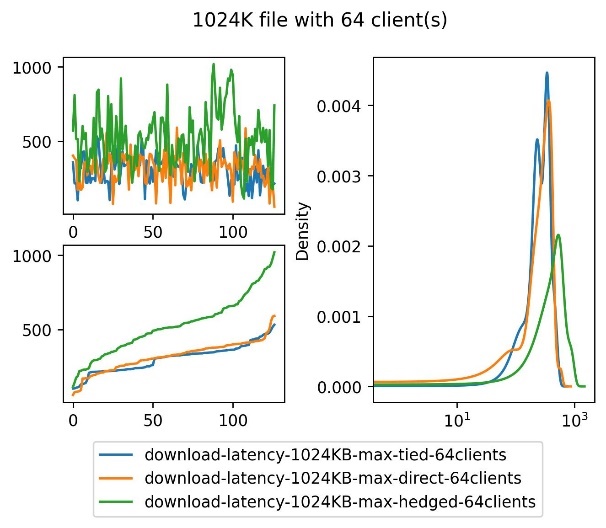
其次是网络请求的本身的问题。一个请求要完成，必须要经历发送-接收-处理-回复的过程，而延迟正是由这个逻辑中的数据传输产生的，这导致任何一个非并发的访问方式都与直接访问具有相同的访问时延分布，即关联请求策略达不到比直接访问更小的访问延迟（而执行访问策略本身的逻辑代码还可能使性能下降）。因而，关联请求减小延迟的原理是直接地利用请求时延的概率分布来“剪断”超过阈值部分的延迟，通过图形也可看出，关联请求将延迟降低、聚集在了设置的阈值附近，因而在大文件、高并发等概率密度分布较为松散、更容易遇到高时延的场景下有更好的效果。

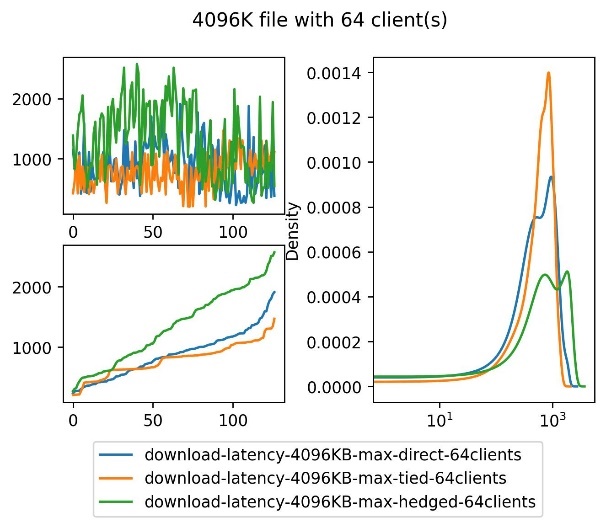
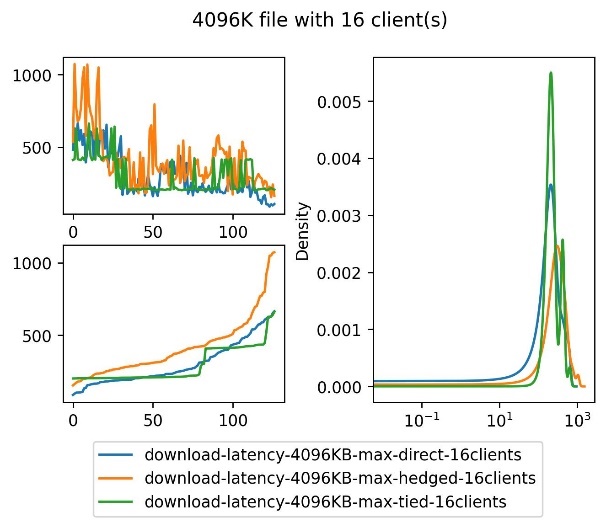












**图3.1测量的尾延迟及其对应的时间-概率密度分布**