**数据取回任务调度**

# 背景

随着视频场景近年来的风靡全球，出于合规性、舆情溯源等诉求，归档存储在云存储中的重要程度日益增强。据IDC报告，2025年将会有超过60%的数据存储在公有云中，而企业数据中18%的数据将会存储7年以上，非常适合使用归档存储系统进行存储。在归档存储系统中，对比其他所有的存储介质,磁带存储的单位价格最低,并且在搭建大型存储系统时更为明显。据富士通测算，磁带存储系统相较于硬盘存储系统，五年将会节省三分之二的总体硬件成本。

基于磁带的归档存储系统，数据读取和写入均需要依赖磁带驱动器。成熟的商业磁带存储系统，为保证系统整体成本，磁带与磁带驱动器比例往往高达200:1以上。这使得请求调度成为了归档存储系统中的至关重要的问题。相较于私有存储系统，公有云存储系统是天然的多租户存储系统。不同租户的不同请求往往会有不同的SLA，而归档云存储服务提供商必须要保证这些请求的合规性，这无疑使得数据请求调度的重要性进一步凸显。

# 基本概念

本题目基于云存储中最为广泛应用的对象存储为背景。在对象存储中，每个数据单元存储为称为对象的离散单元。这些对象可以是任何类型和大小的文件或者是其他类型的数据。为了能更好地理解题目，我们需要对一些基本概念进行澄清。

## 2.1 取回请求

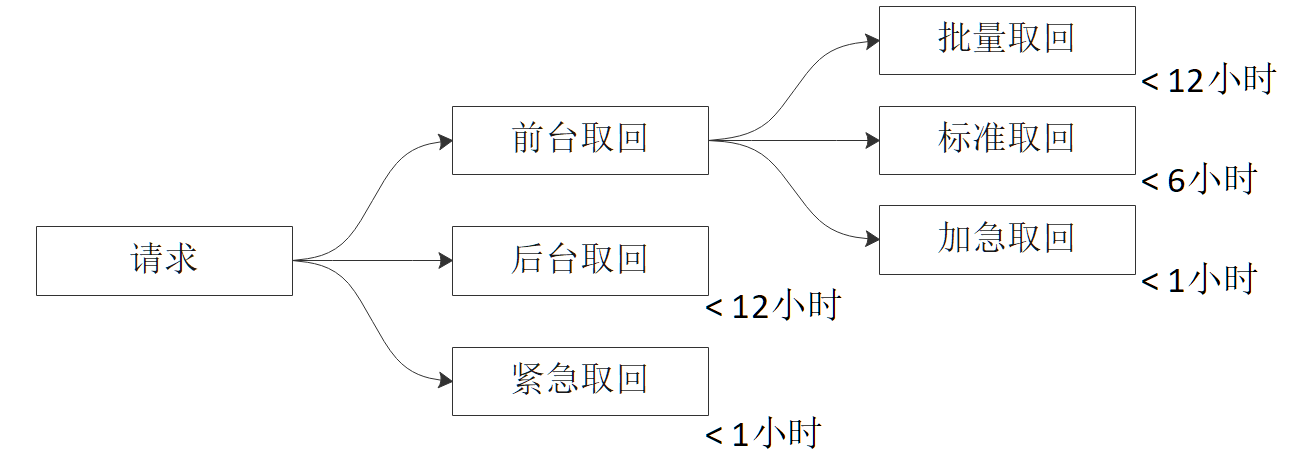
由于归档存储系统本身的特性，通常不会允许用户直接访问归档存储系统中的数据。用户在访问数据前需要将数据取回到标准存储中（业界也自称该动作为“解冻”）。商业上，云储存服务提供商与用户会就该数据取回过程签订一个协定以约束取回的时间，并据此收取数据取回费用并进行合规性检查。一份典型的数据取回协定如下所示：

* 批量取回：12小时内取回到标准存储中，并提供用户数据访问；
* 标准取回：6小时内取回到标准存储中，并提供用户数据访问；
* 加急取回：1小时内取回到标准存储中，并提供用户数据访问；

## 2.2 请求类型

对于云存储系统而言，数据请求不仅来着用户，也会来着系统本身（巡检、数据恢复、垃圾回收等）。而不同来源的请求通常会被附以不同的请求类型，场景的请求类型如下所示：

* 前台取回：即用户发起的取回请求，保证用户的数据取回协定即可；
* 后台取回：由巡检等后台服务发起，执行时间可被推迟，在12小时内执行即可；
* 紧急取回：通常由异常情况触发（数据恢复等），需要在1小时内尽快执行；

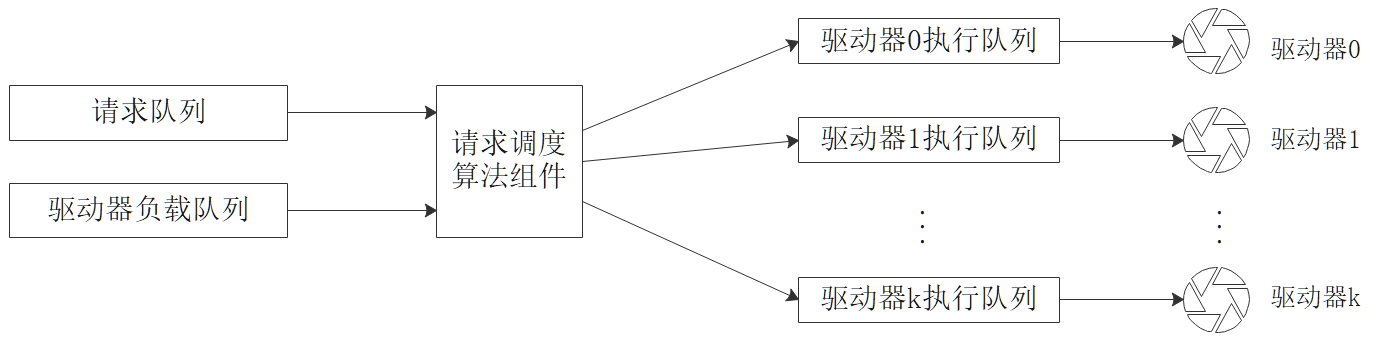


## 2.3 磁带驱动器负载

由于磁带驱动器维护、紧急请求、流量限制、软硬件降级等原因，磁带驱动器能够承受的负载并不是一成不变的。

# 任务描述

选手将会负责编写磁带归档存储系统中非常关键的数据请求调度算法组件，并管理数个磁带驱动器的执行队列。**该算法组件需要决策每个小时每个磁带驱动器需要执行的请求队列，并将请求在小时开始的时候一并发布到磁带驱动器执行队列中**。一旦请求发送到执行队列中，便无法再对队列中的请求进行调整。磁带驱动器会依次处理执行队列中的请求，直到队列中请求执行完毕或者该小时结束**。**



* 请求队列：每个小时开始的时候会发布当前收到的请求，请求示例如下所示：

{

RequestID: 994871,

RequestType: FE,

SLA: 12,

Driver: [1, 2, 4],

RequestSize: 53,

LogicalClock: 342

}

该消息表示，请求ID为994871，前台请求FE（后台为BE，紧急为EM），SLA为12小时内取回，该请求可以被驱动器 1,2或4执行，请求数据量为53MB, 请求发起逻辑时钟为第342小时

* 驱动器负载队列：每个小时开始的时候将会发布该小时各驱动器能处理请求的数据量，示例如下；

{

DriverID: 4,

Capacity: 950,

LogicalClock: 342

}

该消息表示，第4号驱动器在第342小时能取回950MB的数据；

* 请求调度算法组件：算法组件需要在每个小时开始的时候，基于将各个驱动器需要执行的请求发送到驱动器执行序列中。请求执行消息如下所示：

{

DriverID: 4,

RequestList: [994871, 993812, 8376, 736],

LogicalClock: 342

}

该消息表示，当前逻辑时钟为第342小时，调度算法希望第4号驱动器顺序执行请求994871, 993812, 8376, 736。

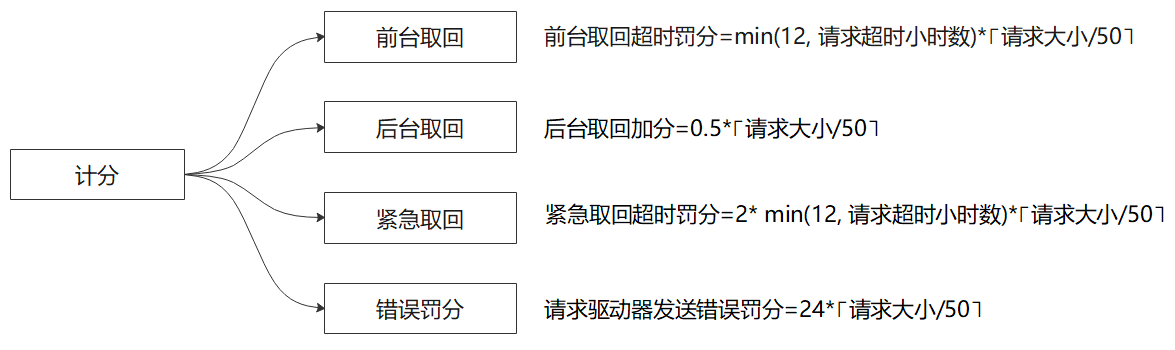
* 驱动器执行队列：驱动器将会依次执行队列中的请求执行消息。

# 评分标准

评分程序会记录驱动器执行的情况，并对前后台以及紧急请求合规性进行校验。积分规则如下：

* + 前台取回超时罚分=min(12, 请求超时小时数)\*⌈请求大小/50⌉
  + 后台取回加分=0.5\*⌈请求大小/50⌉
  + 紧急取回超时罚分=2\* min(12, 请求超时小时数)\*⌈请求大小/50⌉
  + 请求驱动器发送错误罚分=24\*⌈请求大小/50⌉
  + 最终总分 = 后台取回加分 - 前台取回超时罚分 - 紧急取回超时罚分 - 请求驱动器发送错误罚分

**如果发送超过驱动器执行能力的请求，超出能力部分的请求将会被丢弃不进行处理。**最终总分最高者获胜，分数相同调度用时短者获胜。



# 接口样例

**选手需要基于抽象类Scheduler实现调度器类FinalScheduler**，该类包含两个方法 init 和 schedule。如下所示：

class Scheduler(metaclass=ABCMeta):

@abstractmethod

def init(self, driver\_num: int) -> None:

pass

@abstractmethod

def schedule(self, logical\_clock: int, request\_list: list, driver\_statues: list) -> list:

pass

其中：

**def init(self, driver\_num: int) -> None**

启动时会调用该接口告诉选手驱动器数目，例如驱动器数目为3，那么驱动器编号即为0,1和2

**def schedule(self, logical\_clock: int, request\_list: list, driver\_statues: list) -> list**

* + logical\_clock 当前逻辑时钟
  + request\_list当前逻辑时钟的请求JSON数组；
  + driver\_statues当前逻辑时钟的驱动器负载JSON数组；
  + 返回：当前逻辑时钟算法决策需要执行的请求执行消息JSON数组；

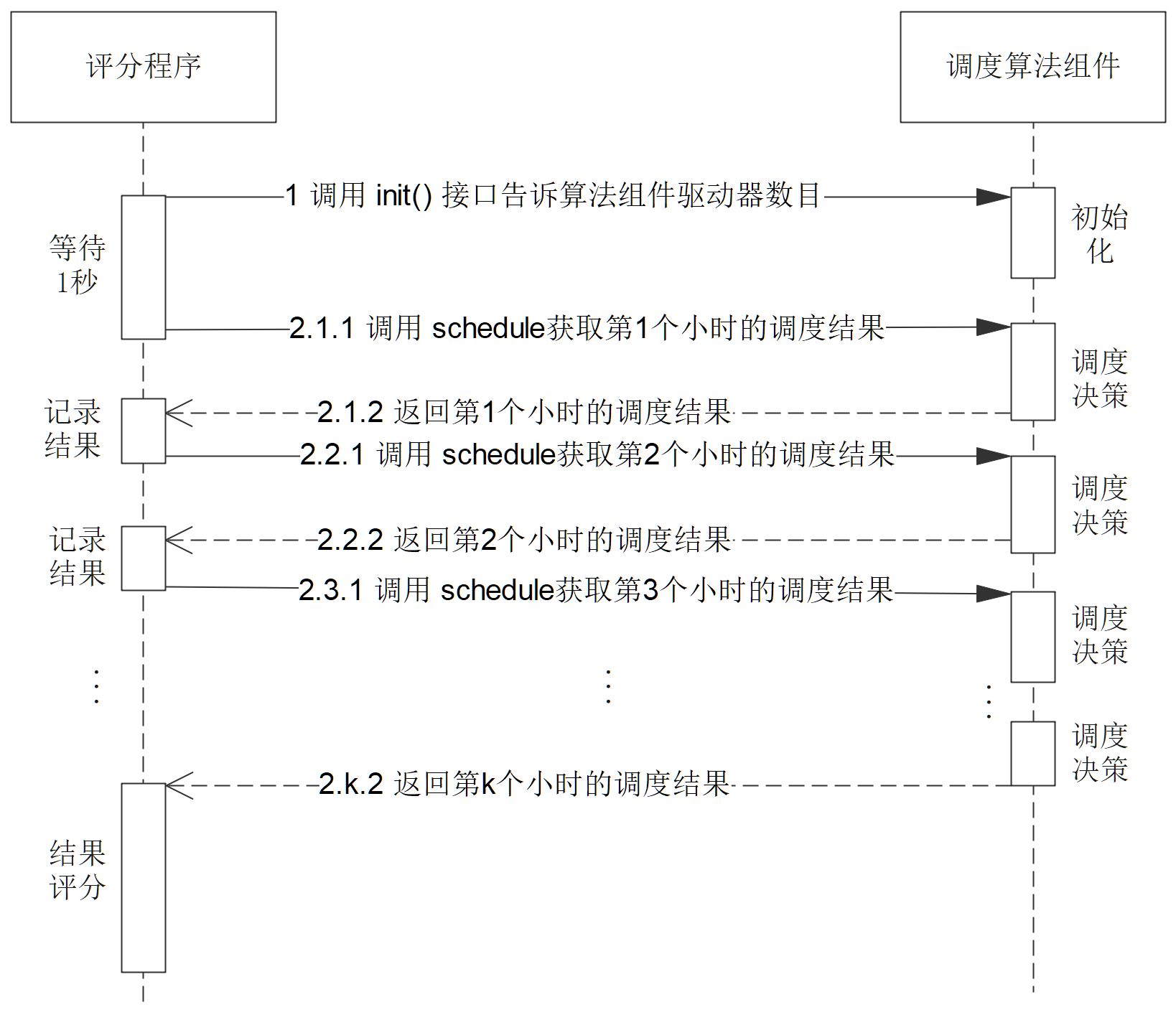
# 附录1 FAQ

1. **应当采用哪种程序设计语言完成比赛？**

答：我们会提供 Python 3模板，选手需要实现模板类中的接口。

1. **评分的具体过程是什么样的？**

答：选手设计的数据请求调度算法组件会以动态链接的形式链接到我们的评分程序中，评分程序将会根据接口调用返回情况进行评分。具体流程如下所示：



1. **未请求的结果如何积分？**

答：最后一小时调度请求发出后，评分程序将会进行结果评分。所有未超时的未调度请求将不进行评分，所有超时的未调度指令将会按超时情况计入总分。

1. **调度返回超时如何处罚？**

答：我们鼓励轻量级的调度算法，并严禁限制Schedule函数调用时间。每小时调度将会有5秒钟进行调度决策，结果记录阶段将会严格记录调度决策时间，并将总调度决策时间作为评分依据。若超时，评分程序会停止计分，并将该提交判罚出局。

1. **是否有提交次数限制？**

答：是的，每组选手有3次提交机会。评分程序会以最后一次提交结果作为最终结果。