**计算任务调度**

# 背景

随着大模型时代的到来，AI Infra的重要性大大凸显，学术与工业界对于大型智算中心的计算任务编排调度问题展开了系列研究。相较于私有计算中心，公有云计算系统是天然的多租户计算系统。不同租户的不同计算任务请求往往会有不同的SLA，而智算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

本题目以云计算中心计算任务调度为背景，在智算中心中，每个计算单元为称为任务的离散单元。这些计算任务可以是任何类型和大小的进程。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 计算请求

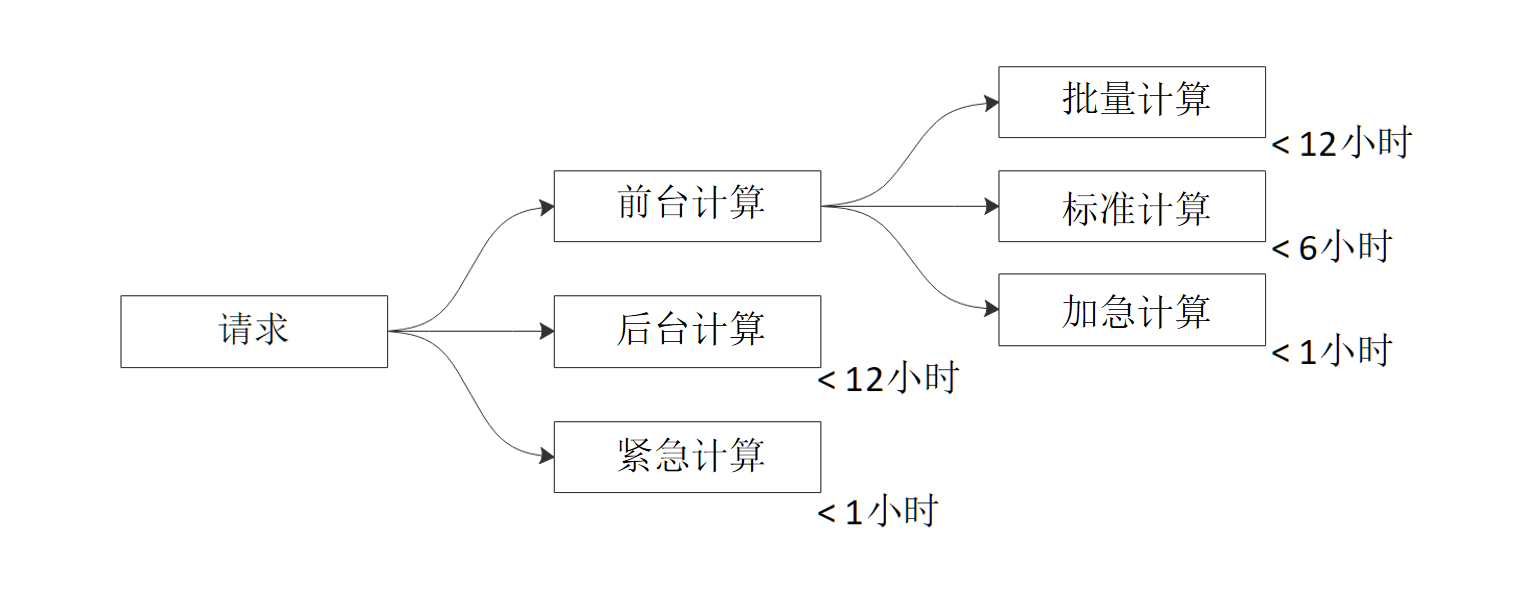
商业上，智算中心与用户会就该计算任务过程签订一个协定以约束计算耗时，并据此收取计算费用并进行合规性检查。一份典型的数据取回协定如下所示：

* 批量计算：12小时内完成计算任务，并提供最终结果；
* 标准计算：6小时内完成计算任务，并提供最终结果；
* 加急计算：1小时内完成计算任务，并提供最终结果；

## 2.2 请求类型

对于云计算系统而言，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：

* 前台计算：即用户发起的计算请求，保证用户的计算协定即可；
* 后台计算：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；
* 紧急计算：通常由异常情况触发，需要在1小时内尽快执行；

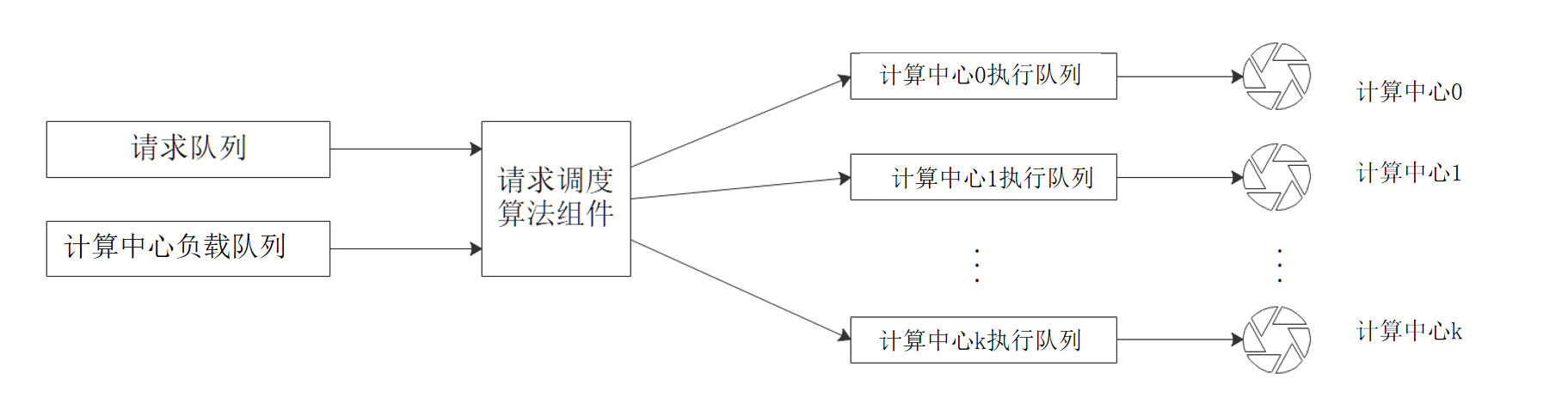


## 2.3 计算负载

由于系统维护、紧急请求、流量限制、软硬件降级等原因，计算中心能够承受的负载并不是一成不变的。

# 任务描述

选手将会负责编写云计算系统中非常关键的计算任务请求调度算法组件，并管理数个计算中心的任务执行队列。**该算法组件需要决策每个小时每个计算中心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计算中心执行队列中**。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。计算中心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小时内执行完成，该请求可以被计算中心 1,2或4执行，计算负载为53, 请求发起逻辑时钟为第342小时。

* 计算中心负载队列：每个小时开始的时候将会发布该小时各计算中心能处理的计算量，示例如下；

{

DriverID: 4,

Capacity: 950,

LogicalClock: 342

}

该消息表示，第4号计算中心在第342小时能执行950的计算量；

* 请求调度算法组件：算法组件需要在每个小时开始的时候，基于将各个计算中心需要执行的请求发送到计算中心执行序列中。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号计算中心顺序执行请求994871, 993812, 8376, 736。

注意，算法返回的结果是一个json字段数组，如下所示：

['{"DriverID": 0, "RequestList": [2499, 2510, 2543, 2529, 2521], "LogicalClock": 20}',

'{"DriverID": 1, "RequestList": [2546, 2456, 2545, 2484], "LogicalClock": 20}',

'{"DriverID": 2, "RequestList": [2498, 2525, 2537, 2532, 2501, 2454], "LogicalClock": 20}',

'{"DriverID": 3, "RequestList": [2509, 2486, 2462, 2482], "LogicalClock": 20}']

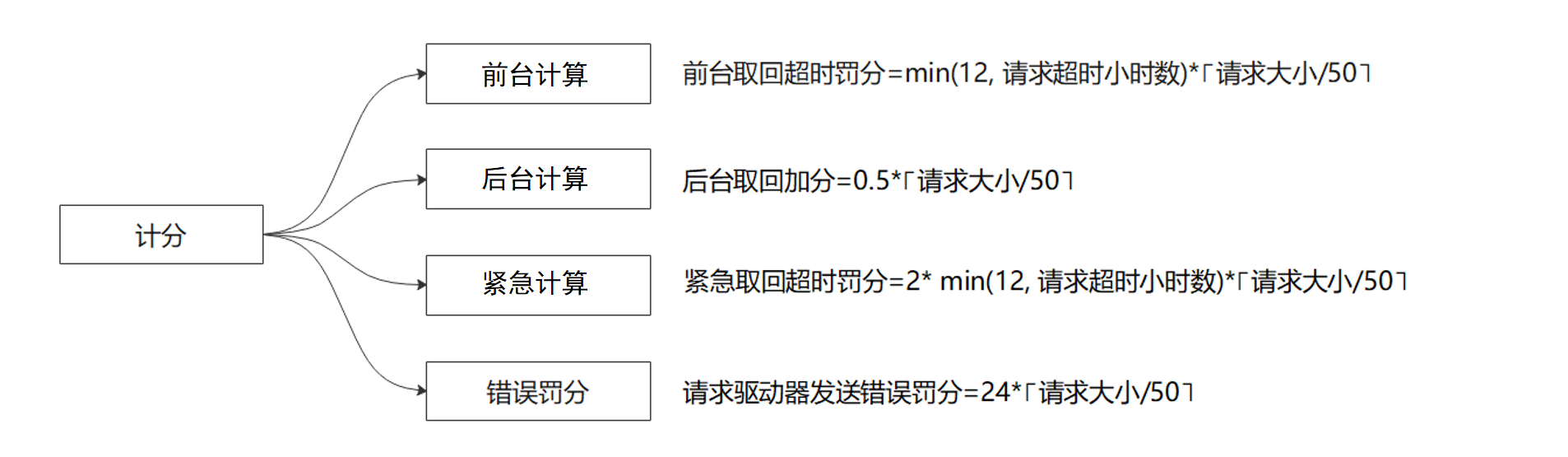
* 计算中心执行队列：计算中心将会依次执行队列中的请求执行消息。

# 评分标准

评分程序会记录计算中心任务执行的情况，并对前后台以及紧急请求合规性进行校验。积分规则如下：

* + 前台计算超时罚分=min(12, 请求超时小时数)\*⌈请求大小/50⌉
  + 后台计算加分=0.5\*⌈请求大小/50⌉
  + 紧急计算超时罚分=2\* min(12, 请求超时小时数)\*⌈请求大小/50⌉
  + 请求计算中心发送错误罚分=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 - 请求计算中心发送错误罚分

**如果发送超过计算中心执行能力的请求，超出能力部分的请求将会被丢弃不进行处理。**



# 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 FinalScheduler(Scheduler):

    def \_\_init\_\_(self):

        pass

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

        pass

    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计算中心数目，例如计算中心数目为3，那么计算中心编号即为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当前逻辑时钟的计算中心负载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；

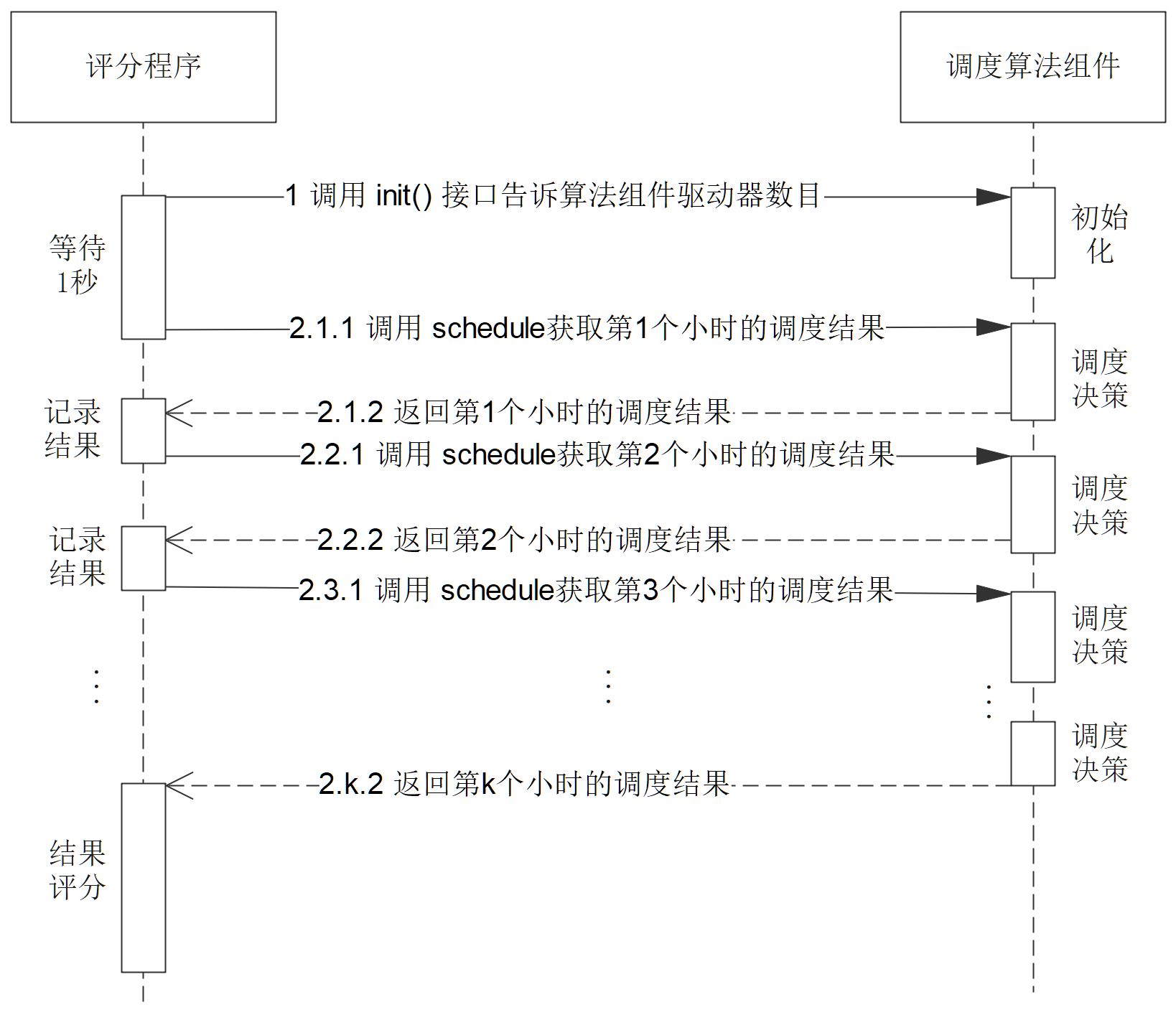
# 附录1 FAQ

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，你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：我们会提供相应的评分程序，你设计的数据请求调度算法组件会被评分程序调用，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：我们要求每小时schedule函数调用时间不得超过5秒，结果记录阶段将会严格记录调度决策时间，若超时，评分程序会停止计分，并将该提交判罚出局。

1. **如何获取给定的代码模板以及评分程序等？**

答：题目文档，代码模板，评分程序以及数据集均已由实验室上传到Github平台，同学可以自行下载，如果存在疑问可以在该项目的issues中提问，或者发送邮件。

1. **考核的提交要求是什么？**

答：需要通过邮件发送文档以及完整项目代码，文档需要给出自测分数以及算法架构图。我们将会用评分程序对代码分数进行二次验证，并进行代码查重，希望同学们遵守学术诚信，不要出现大量代码和算法细节高度雷同的情况。此外，我们鼓励提升代码的整洁性与可读性，并希望同学在文档中充分反映自身的思考以及创新过程。