# m云集群成本优化

限制

单个测试样例时间：5s

总时间：2min

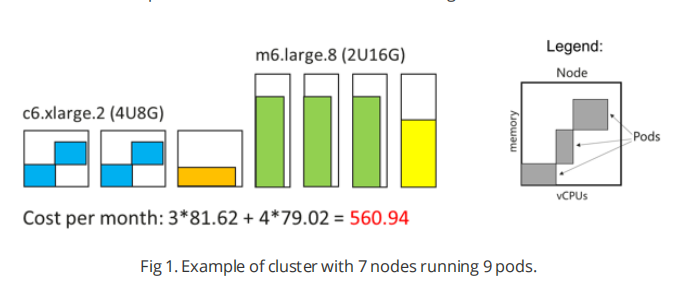
每个测试用例的内存限制：512M

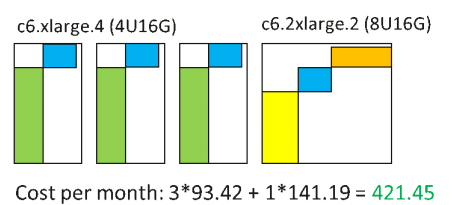
输入：标准输入

输出：标准输出

## 背景

在云服务提供商中，使用容器技术管理应用程序正在成为主流。此类服务的客户创建一个集群**cluster**，该集群由一个或多个节点**nodes**，用于运行客户的容器。容器提交用于由客户以**pod**的形式执行，并由集群分配给集群节点调度器**cluster scheduler**。集群节点通常对应于虚拟机（VM），因此可能的节点类型（例如。每个节点的可用CPU和内存数量）受到一组VM实例类型（称为风味**flavor**）。每种口味都有每秒的价格，集群成本**cluster cost**为基于所使用的节点的价格及其运行时间来计算。





前后对比两种解决方案，可以按照legend中横纵坐标为CPU和内存消耗查看集群中每个pod中的资源占用情况。

目前，客户必须手动决定使用哪些口味和多少口味作为集群节点。集群节点的选择直接影响客户的成本，而且是次优的决策可能导致支付未使用的资源和大量的成本超支（即只有一个最优解）。在示例中在集群之下，使用不同的pod可以将运行同一组pod的成本降低25%节点风味。

不过，这个例子只考虑了一组固定的 pod。实际上，正在运行的 pod 的集合是不断变化的。根据当前客户的需求创建新 pod 并删除旧 pod。例如，高峰时段会创建更多的 pod，而夜间运行的 pod 则会减少。有些 pod 可能只提交一次又一次，例如运行一些数据处理工作。在这种情况下，某个时间点具有成本效益的集群配置在其他时间点可能并不高效。例如，不同时间段可能需要不同数量的节点，甚至是不同的节点风味（节点类型）。

为了节省给客户带来的过多负担，可以手动调整集群配置。这项挑战的目标是开发一个**自动集群管理解决方案automated cluster management solution**，它将消除这一负担，代表客户决定使用哪种节点风味（节点类型）、何时使用以及使用多少。更具体地说，我们的目标是设计出以下算法，在满足所有客户要求的同时，使客户集群在一定时间段内的**运行总成本最小化minimize the total cost**。解决方案的第一部分是 **pod 调度算法pod scheduling algorithm**，用于决定将 pod 放在哪个节点上。解决方案的第二部分是**节点扩展算法node scaling algorithm**，用于决定将哪些节点添加到集群中，以容纳未调度的 pod。主要挑战在于，在生产中，这些算法必须在在线环境中工作，逐个处理客户请求，而不知道下一个请求和何时会出现。

## 描述

您的任务是开发用于管理客户集群的服务的主要逻辑。最初的集群是空的，没有节点，也没有正在运行的 pod。服务将获得一组可能的每种节点的资源容量（CPU 和内存）和每秒价格。

该服务需要处理两种类型的客户请求：

1. **创建Create** pods请求，指定创建一些pod并且他们需要的CPU和内存请求。
2. **删除Delete** pods请求，指定之前创建的一组pod来删除。

请求都是在线的，每个请求都必须要处理完之后，下个请求才会过来。

**创建Create** pods需要立即调度所有pods，请求中的每个pod都必须被放置到一些集群节点上，并且消耗对应的资源。一个节点上所有 pod 消耗的资源总量不得超过节点的 CPU 和内存容量。如果某些 pod 因资源不足而无法调度资源不足，则必须创建新节点，直到所有 pod 都成功调度为止。节点的大小只能从指定的风味（节点类型）中选择。

**删除Delete** pods需要立刻释放在代理节点上特定pod消耗的资源。一旦某个节点闲置（不运行任何 pod），它就会被自动从群集中删除。。请确保您的解决方案不会在删除节点后重新使用这些节点。

您的解决方案必须通过与分级程序交互来实现所述服务的逻辑。交互的格式将在下面的单独章节中介绍。

解决方案根据计算得出的**集群总成本total cluster cost**分级如下。每个集群节点的成本是由节点味道的价格和节点运行时间（节点创建和删除之间的时间）相乘计算得出的。节点创建和删除之间的时间）。集群总成本就是的总和。集群成本越低就越好。

## 交互（输入输出）

与程序的交互是通过标准输入和输出流来组织的。

程序启动时，会收到以下格式的虚拟机风味flavor数据。

第一行的整数`（1<=F<=50）代表了风味数量.

后续的F行，每一行包含了两个整数和一个浮点数：对应风味的可用CPU，可用内存，每秒钟的价格。CPU为抽象的CPU单元数量，CPU 单元的确切性质与问题无关，内存为以M的整数。

确保CPU的值范围为100~51200，内存范围为128~2000000。每秒钟的价格为0.005~20，小数点后至多4位数字。

然后，程序开始接收并处理一系列请求，请求的数量为2~20001。

每个请求开始的一行包含了时间戳（0~10000000），请求类型（CREATE，DELETE，END），整数值S为请求的pod个数。严格确保了时间戳是递增的，并且创建CREATE请求的S<=100。

1. 如果请求类型为CREATE，则后续的S行对应的pods，每一行包含3个整数
2. 第一个整数pod id，确保在一个测试用例中，所有的pods从1开始按照连续整数编号。
3. 第二个整数需要的CPU个数，整数。
4. 第三个整数需要的内存以M为单位，整数。
5. 如果请求类型为DELETE，
6. 如果请求类型为END，

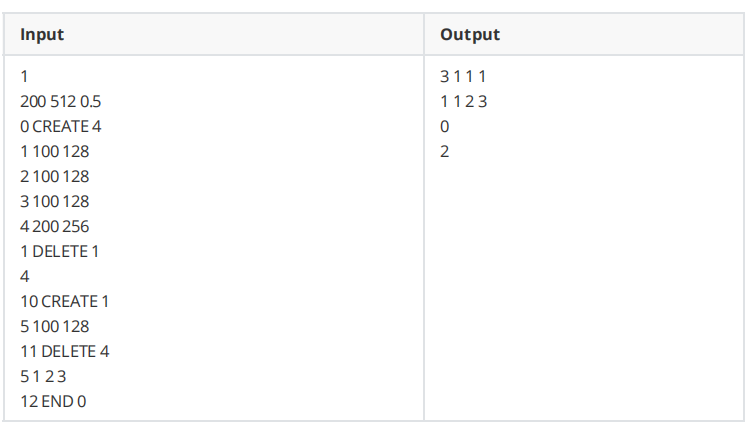
确保所有pod的删除都在end之前，一次交互不超过10000个pod创建。

对于CREATE请求，你的程序必须输出两行，否则不会接受后续的请求。

1. 第一行包含一个整数C(0<=C<=100)表示对于新节点的flavor个数，然后的C个整数id选择对应的flavor。
2. 第二行包含S个整数，表示用于容纳请求中相应 pod 的节点索引，顺序相同。请注意，这些数字的顺序必须与输入中 pod 的顺序一致，节点必须按照您的请求中出现的顺序编入索引。

打印之后，不要忘记刷新输出。fflush,cout.flush()在C++中，System.out.flush()在Java中，stdout.flush()在Python中。

## 示例



在这个例子中，有一个flavor和4个请求。第一个在0时间戳时请求创建了4个pod，第二个请求在1时间戳删除了4号pod，第三个请求在10时间戳创建了1个pod，最后一个请求删除了所有的pod。这里的解决方案中，对于第一个请求，创建了3个节点来满足该请求，1号和2号pod被放置到节点1上，3号pod被放置到节点2上，4号pod被放置到节点3上。对于第二个请求，删除4号pod可以释放节点3。对于第三个请求，把新创建的5号pod放置到节点2上。对于最后一个请求删除所有pods和节点。这个解决方案中用了3个节点，有两个节点用了11秒，第三个节点用了1秒，对应中的代价为2\*0.5\*11+0.5\*1=11.5。

## 评估

你的解决方案将会在多个测试用例上运行。对于每个测试用例会有一个独立的交互界面，解决方案分数将会计算为，cost为上述的集群cost，baseline为简单基础解决方案的得分。每个测试用例的分数将会相加为总得分，越高越好。

如果破坏了事件或内存限制，或者输出格式非法，该测试用例的得分为0.。