2021华为软件精英挑战赛 上合赛区32强二等奖

题目大概意思是我们是公有云提供商，有很多类型服务器（不同的core和memory，价格，每天能耗）可以购买，给用户提供不同类型的虚拟机服务。而且虚拟机还分单双节点。我们如何在用户虚拟机请求给定的情况下，购买充足服务器满足用户所有需求，但是成本最低。这个成本包含服务器购买成本和服务器运行能耗成本。并且每天还有一定次数可以将部署好的虚拟机迁移到其他服务器上去，以便能将服务器尽可能满，空出一些服务器，减少能耗成本。所以本题的关键点在于服务器的购买，虚拟机在服务器上的部署，虚拟机的迁移。

服务器的选择：

在这个比赛中，服务器的选择是比较重要的。我们首先对服务器数据进行了分析，发现服务器的购买价格以及每天的能耗都是core和memory的二元函数，即购买价格和core、memory基本处于一个平面上，所以我们进行了最小二乘拟合，得到表达式a1\*core+b1\*memory=serverCost, a2\*core+b2\*memory=powerCost。而我们讨论中想到当服务器用的天数多的时候，每天能耗的价格占主导因素，而服务器用的天数少的时候，服务器的购买价格占主导因素。我们开始设定了一个剩余天数的阈值，当小于250天的时候，利用serverCost选服务器，否则用powerCost选阈值，发现少了几千万，证明是有效果的。但是参数越少越好，最好弄成自适应的。所以我们用拟合得到的表达式T\_days-(a1+b1)/(a2+b2)作为分界线，之前的用powerCost，之后的用serverCost。

虚拟机的分配：

这其实可以看成一个背包问题，或者装箱问题。每个服务器可以看成一个背包，最大容量就是最大的core和memory，如果能够将每个背包尽可能装满，则能耗肯定最低，因为就可以有空闲的服务器不增加能耗。所以我们写了01背包的动态规划代码，这里有两个限定条件，所以状态为3个，core、memory、服务器数量。最简答的01背包的只有两个状态量，dp[i][j]=max(dp[i-1][j],dp[i-1][j-w[i-1]]+val[i-1])表示前i个物品在背包容量为j的情况下能装的最大价值。而这里dp[i][v][u]=max(dp[i-1][v][u],dp[i-1][v-core[i-1]][u-memory[i-1]]+core[i-]+memory[i-1])表示前i个虚拟机在服务器core容量为v，memory容量为u的情况下能装下的最大的core和memory和。我们对三维状态量优化成了两维（威少补充：咋做的）

但是效果不太好，因为虚拟机的请求是一定的，则肯定是前面的背包装得满，后面就没法装得满（威少再补充一下原因），并且这种方法还很耗时，所以最终我们摒弃了这种算法。转而使用比较朴素的first fit，best fit这类方法。最开始我们使用best fit算法耗时太长，但是后面我们更改了服务器的存储结构，将其用set存储，排序方式按照资源剩余率排序，剩的少的在前面，则虚拟机部署时从前往后找到的第一个服务器就是剩余容量差不多能装上的。不过set其实设计初衷就不能直接修改其中的值，只有删除和插入节点才会触发重排，并且只重排插入删除节点相关的节点，这样才能达到logn的时间复杂度。最终我们对于需要修改的服务器先将存在set中的指针删除，然后更改容量，然后再插入。

虚拟机的迁移：

我主要负责迁移部分代码的编写，后面对于服务器的购买优化我也做了一些工作，提升了好几个名次。迁移的目的就是将每个服务器尽可能装满，然后空出一些服务器，这样就可以减少能耗成本。最开始我们是将所有的服务器按照部署的虚拟机的数量排序，多的在前，然后将后面服务器上的虚拟机往前面迁移。但是因为O(n^2)的复杂度，所以速度比较慢。后来我们在迁移之前将剩余率<0.04的服务器排除，这样虽然还是O(n^2)的复杂度，但是时间常数可以降低不少。后面我们又发现可以将当天需要删除的虚拟机先迁移到后面，将当天不删除的迁移到前面，这样每天之后删除的vm部署的服务器就很可能空出来，这样又提升了一些分数。

最终我们初赛22名，复赛12名，夺得上合赛区32强二等奖。