基于DRL的动态租用实例任务调度

刘肇泽

控制与计算机工程学院

2024 年 1 月 28 日

目录

摘要

与 Cost-Aware 的不同之处：

1. 引入了 On Demand 和 Spot 两种计费规则的实例
2. 引入了 Moldable 和 Rigid 两种任务类型
3. 引入了任务的挂起、恢复和跨区域调度

使用 T 表示时间段, t 表示时刻.

实例 (Instance) 的数学模型

固有属性：

- I_c 实例的计算核心数
- I_m 实例的内存大小
- I_r 实例所在区域
- I_b 实例的计费类型 (On Demand/Spot)
- v_l 将任务加载到实例内存中的速度 (loading speed)
- v_w 将任务数据从实力内存中写回外存的速度 (writing speed)

计费类型：

- On Demand: 每小时价格固定
- Spot: 每小时价格随市场波动

状态属性：

- $I_p(t)$ 实例在时刻 t 的价格
- t_i 实例空闲的时刻 (idle time)
- T_r 实例的剩余租期 (remain time)

任务 (Job) 的数学模型

固有属性：

- J_c 任务需要的计算核心数
- J_m 任务需要的内存大小
- J_t 任务的类型 (Moldable/Rigid)

状态属性：

- J_l 任务长度（小时）
- t_a 任务到达时刻 (arrival time)
- J_r 任务上一次执行所在的区域

Downey 加速模型

并行度 (parallelism) 具有均值 A 和标准差 σ 两个参数.
将 $0 \leqslant \sigma \leqslant 1$ 的模型称为Low variance model, 将 $\sigma > 1$ 的模型称为High variance model.

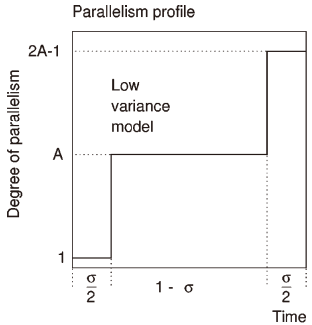


图: Low variance model

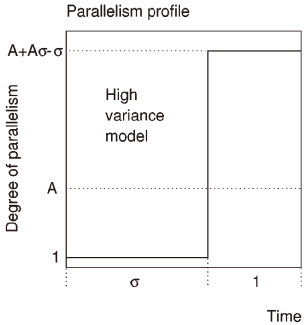


图: High variance model

Downey 加速模型

加速系数 $SU(n)$ 的计算

当 $0 \leqslant \sigma \leqslant 1$ 时：

$$SU(n) = \begin{cases} \frac{An}{A+\sigma(n-1)/2}, & 1 \leqslant n \leqslant A, \\ \frac{An}{\sigma(A-1/2)+n(1-\sigma/2)}, & A < n \leqslant 2A-1, \\ A, & n > 2A-1. \end{cases}$$

当 $\sigma > 1$ 时：

$$SU(n) = \begin{cases} \frac{nA(\sigma+1)}{A+A\sigma-\sigma+n\sigma}, & 1 \leqslant n \leqslant A+A\sigma-\sigma, \\ A, & n > A+A\sigma-\sigma. \end{cases}$$

其中 n 为计算核心数.

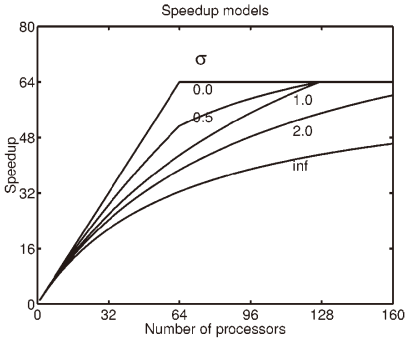


图: Speedup curves for a range of values of σ when $A = 64$

任务类型 J_t 与实际执行时间 T^e

Moldable/Rigid

任务类型 J_t 影响任务实际执行时间 T^e 的计算.

Algorithm 1: 计算任务实际执行时间 T^e

if $J_t = Moldable$ **then**
  $T^e = J_l \cdot \frac{SU(J_c)}{SU(I_c)}$;
end
if $J_t = Rigid$ **then**
  if $J_c \leqslant I_c$ **then**
    $T^e = J_l$;
  else
    任务调度失败;
  end
end

挂起时间 T_w 与恢复时间 T_l

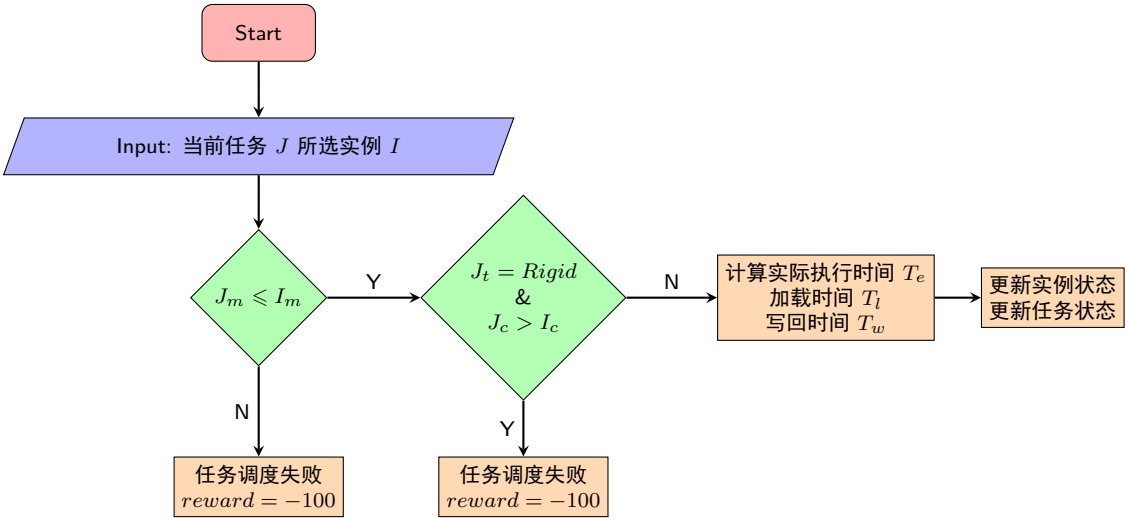
当正在执行任务的实例到期时，任务需要挂起到硬盘，该操作需要的时间为：

$$T_w = \frac{J_m}{v_w}$$

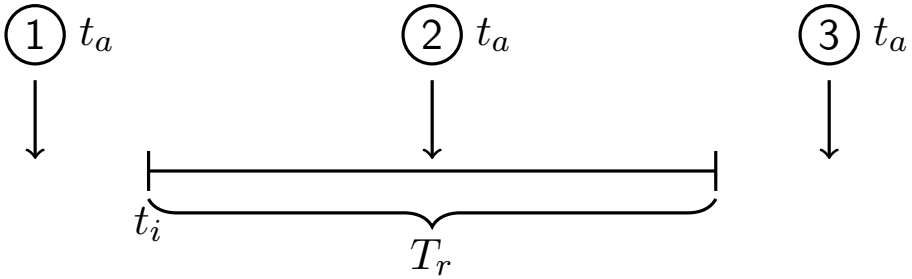
当任务首次在实例中运行或从挂起状态恢复时，任务需要加载到内存，该操作需要的时间为：

$$T_l = \begin{cases} \dfrac{J_m}{v_l}, & J_r = None \text{ or } J_r = I_r, \\ 2 \times \dfrac{J_m}{v_l}, & \text{otherwise.} \end{cases}$$

任务调度流程


Start
Input: 当前任务 J 所选实例 I
$J_m \leqslant I_m$
Y
$J_t = Rigid$
&
$J_c > I_c$
N
计算实际执行时间 T_e
加载时间 T_l
写回时间 T_w
更新实例状态
更新任务状态
N
任务调度失败
$reward = -100$
Y
任务调度失败
$reward = -100$

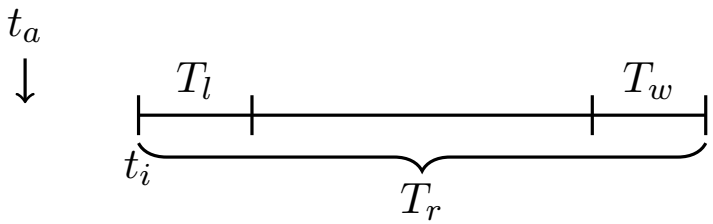
任务提交的三种情况



1. 任务在实例空闲之前提交 $t_a \leqslant t_i$
2. 任务在实例空闲之后、到期之前提交 $t_i < t_a < t_i + T_r$
3. 任务在实例到期之后提交 $t_a \geqslant t_i + T_r$

情况一: 任务在实例空闲之前提交 $t_a \leqslant t_i$

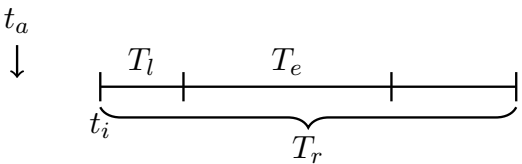
判断是否续租




```
1 cost ← 0;
2 while T_r − T_l − T_w ⩽ 0 do
3 |   T_r ← T_r + 1;                    /* 续租一小时 */
4 |   cost ← cost + t_i时刻的实例价格;
5 end
```

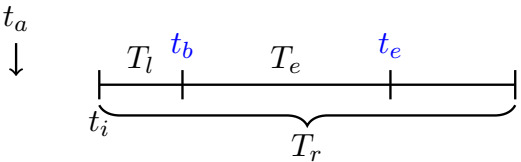
情况一: 任务在实例空闲之前提交 $t_a \leqslant t_i$

任务能够全部执行 $T_e \leqslant T_r - T_l$



情况一: 任务在实例空闲之前提交 $t_a \leqslant t_i$

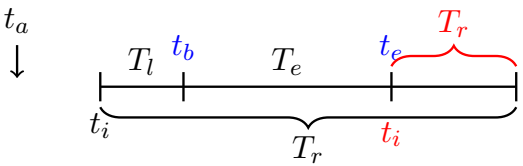
任务能够全部执行 $T_e \leqslant T_r - T_l$



任务在 $t_b = t_i + T_l$ 时刻开始执行, 在 $t_e = t_b + T_e$ 时刻结束执行.

情况一: 任务在实例空闲之前提交 $t_a \leqslant t_i$

任务能够全部执行 $T_e \leqslant T_r - T_l$



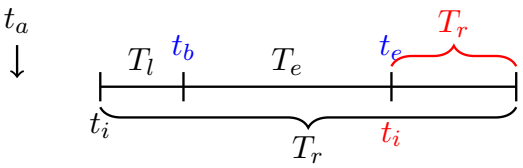
任务在 $t_b = t_i + T_l$ 时刻开始执行, 在 $t_e = t_b + T_e$ 时刻结束执行.

更新实例状态:

- 实例剩余租期 $T_r \leftarrow T_r - T_l - T_e$
- 实例空闲的时刻 $t_i \leftarrow t_e$

情况一: 任务在实例空闲之前提交 $t_a \leqslant t_i$

任务能够全部执行 $T_e \leqslant T_r - T_l$



任务在 $t_b = t_i + T_l$ 时刻开始执行, 在 $t_e = t_b + T_e$ 时刻结束执行.

更新实例状态:

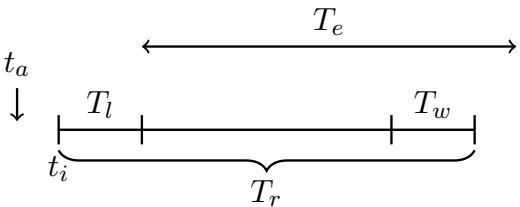
- 实例剩余租期 $T_r \leftarrow T_r - T_l - T_e$
- 实例空闲的时刻 $t_i \leftarrow t_e$

更新任务状态:

- 任务长度 $J_l \leftarrow 0$
- 任务上一次执行所在的区域 $J_r \leftarrow I_r$

情况一: 任务在实例空闲之前提交 $t_a \leqslant t_i$

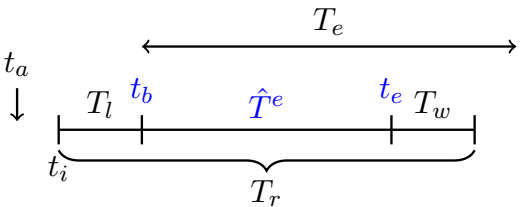
任务只能部分执行 $T_e > T_r - T_l$



情况一: 任务在实例空闲之前提交 $t_a \leqslant t_i$

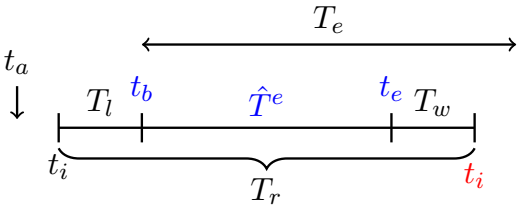
任务只能部分执行 $T_e > T_r - T_l$

任务在 $t_b = t_i + T_l$ 时刻开始执行, 在 $t_e = t_i + T_r - T_w$ 时刻结束执行.
任务实际执行时长 $\hat{T}^e = t_e - t_b$.



情况一: 任务在实例空闲之前提交 $t_a \leqslant t_i$

任务只能部分执行 $T_e > T_r - T_l$

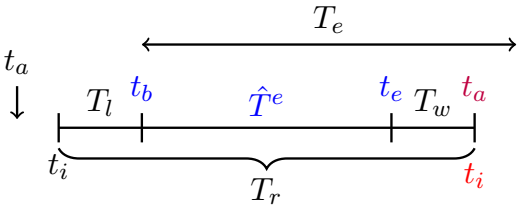


任务在 $t_b = t_i + T_l$ 时刻开始执行, 在 $t_e = t_i + T_r - T_w$ 时刻结束执行.
任务实际执行时长 $\hat{T}^e = t_e - t_b$.
更新实例状态:

- 实例剩余租期 $T_r \leftarrow 0$
- 实例空闲的时刻 $t_i \leftarrow t_i + T_r$

情况一: 任务在实例空闲之前提交 $t_a \leqslant t_i$

任务只能部分执行 $T_e > T_r - T_l$



任务在 $t_b = t_i + T_l$ 时刻开始执行, 在 $t_e = t_i + T_r - T_w$ 时刻结束执行.
任务实际执行时长 $\hat{T}^e = t_e - t_b$.
更新实例状态:

- 实例剩余租期 $T_r \leftarrow 0$
- 实例空闲的时刻 $t_i \leftarrow t_i + T_r$

更新任务状态:

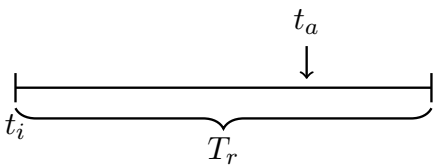
- 任务长度

$$J_l \leftarrow \begin{cases} J_l - \hat{T}^e, & J_t = \text{Rigid}, \\ J_l - \hat{T}^e / \frac{SU(J_c)}{SU(I_c)}, & J_t = \text{Moldable}. \end{cases}$$

- 任务上一次执行所在的区域 $J_r \leftarrow I_r$
- 任务提交时刻 $t_a \leftarrow t_i$

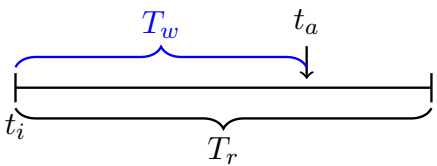
情况二: 任务在实例空闲之后、到期之前提交

$$t_i < t_a < t_i + T_r$$



情况二: 任务在实例空闲之后、到期之前提交

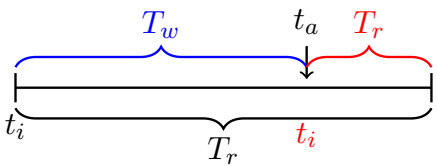
$$t_i < t_a < t_i + T_r$$



浪费时间 $T_w = t_a - t_i$.

情况二: 任务在实例空闲之后、到期之前提交

$$t_i < t_a < t_i + T_r$$

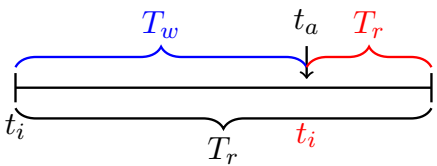


浪费时间 $T_w = t_a - t_i$.
更新实例状态:

- 实例剩余租期 $T_r \leftarrow T_r - T_w$
- 实例空闲的时刻 $t_i \leftarrow t_a$

情况二: 任务在实例空闲之后、到期之前提交

$$t_i < t_a < t_i + T_r$$

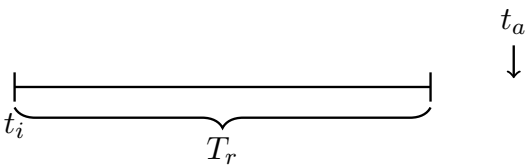


浪费时间 $T_w = t_a - t_i$.
更新实例状态:

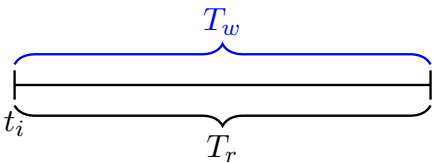
- 实例剩余租期 $T_r \leftarrow T_r - T_w$
- 实例空闲的时刻 $t_i \leftarrow t_a$

转换为情况一.

情况三：任务在实例到期之后提交 $t_a \geqslant t_i + T_r$

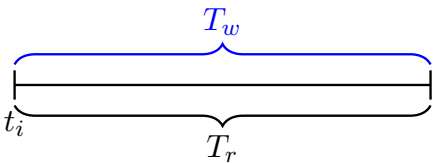


情况三：任务在实例到期之后提交 $t_a \geqslant t_i + T_r$



浪费时间 $T_w = T_r$.

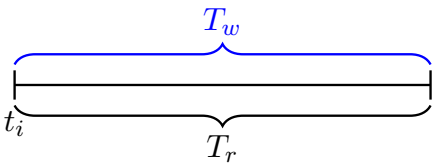
情况三：任务在实例到期之后提交 $t_a \geqslant t_i + T_r$



浪费时间 $T_w = T_r$.
更新实例状态:

- 实例剩余租期 $T_r \leftarrow 0$
- 实例空闲的时刻 $t_i \leftarrow t_a$

情况三：任务在实例到期之后提交 $t_a \geqslant t_i + T_r$



浪费时间 $T_w = T_r$.
更新实例状态:

- 实例剩余租期 $T_r \leftarrow 0$
- 实例空闲的时刻 $t_i \leftarrow t_a$

转换为情况一.

任务调度成功时的奖励

$$reward = \max\left\{-(\xi \cdot \text{cost} + \eta \cdot T_w) \cdot \frac{t_e - t_a}{t_e - t_b}, -50\right\}$$

将租金cost和浪费的时间 T_w 作为惩罚项, 任务响应比 $(t_e - t_a)/(t_e - t_b)$ 作为系数. 限制任务调度成功时的奖励不低于 -50, 与任务调度失败的奖励 -100 拉开差距.
其中, ξ 和 η 是超参数.

状态向量的构成

DQN输入的状态向量为:

$$\left[J_c \quad J_m \quad J_t \quad J_r \quad T_{wait}^{(1)} \quad T_r^{(1)} \quad \dots \quad T_{wait}^{(M)}, T_r^{(M)}\right]$$

即: 任务需要的计算核心数、任务需要的内存大小、任务的类型、任务上一次执行所在的区域; 任务提交给每个实例后预计的等待时间 $T_{wait}^{(j)} = \max\left\{t_i^{(j)} - t_a^{(i)}, 0\right\}$ 和实例剩余的租期.

训练过程

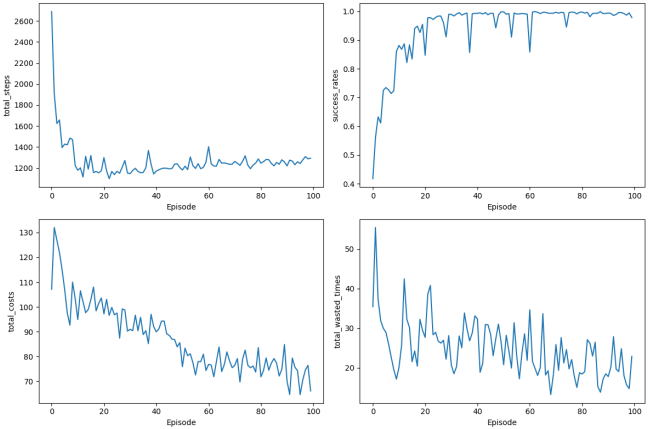


图: 训练过程中调度总数、调度成功率、总开销、总浪费时间的变化

调度算法评估

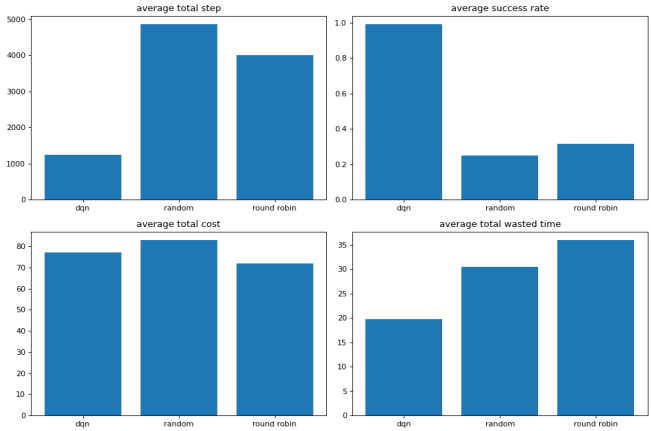


图: DQN、Random、Round-Robin三种调度算法的表现

Earliest算法无法完成该数学模型下的调度任务.

问题一: 时间指标

在任务调度流程中, 记录了每次调度任务的提交时刻 t_s, 开始时刻 t^b 和结束时刻 t^e. 应当如何使用这些时间数据更好地比较不同调度算法的性能?
目前想到的方法:

1. 比较平均每次任务调度的响应时间 $t^e - t_s$
2. 比较平均每次任务调度的相应比 $t^e - t_s/t^e - t^b$
3. 比较平均每个任务从第一次提交到最终全部完成的时间

问题二: 调度成功率

目前DRL调度算法的成功率不能达到 100%. 为了保证调度流程能够正常进行, 在DRL调度失败时使用Round-Robin调度算法, 将任务分配到可以执行的实例上.
是否有可行的办法使DRL调度算法的成功率稳定在 100%?

1. 应用DQN的全部高级技巧: 优先经验回放、双Q学习、对决网络和噪声网络
2. 增加训练轮数
3. ...

参考文献


Allen B. Downey.
A parallel workload model and its implications for processor allocation.
Cluster Computing, pages 133–145, May 1998.


Luke M. Leslie, Young Choon Lee, Peng Lu, and Albert Y. Zomaya.
Exploiting Performance and Cost Diversity in the Cloud.
In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 107–114, Santa Clara, CA, June 2013. IEEE.