华北电力大学
NORTH CHINA ELECTRIC POWER UNIVERSITY

CostAware论文汇报

刘肇泽

控制与计算机工程学院

2023 年 10 月 16 日

目录

云服务器任务调度模型

任务调度流程:

1. 用户提交任务
2. 任务调度器根据任务属性和虚拟机状态为任务分配虚拟机

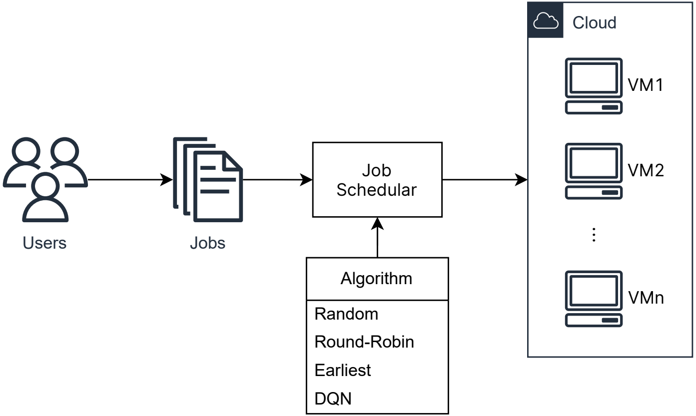


图: 云服务器任务调度模型

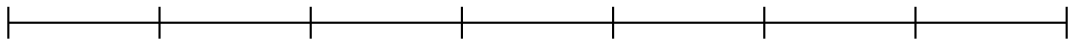
任务属性

对于用户提交的每个任务都具有如下属性：

- ID：任务编号
- $reqCom$：总计算量
- T_{submit}：提交时刻
- QoS：响应时间指标
- $Type$：任务类型
  - 计算敏感型
  - I/O敏感型

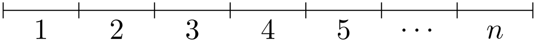
对任务提交时刻的建模

假设单位时间内用户**平均**提交的任务数量为 λ



对任务提交时刻的建模

假设单位时间内用户**平均**提交的任务数量为 λ，那么在单位时间内均匀观察 n 次



对任务提交时刻的建模

假设单位时间内用户平均提交的任务数量为 λ，那么在单位时间内均匀观察 n 次，每次观察时用户提交任务的概率为 $p = \dfrac{\lambda}{n}$。

$$\begin{array}{|c|c|c|c|c|c|c|}
\frac{\lambda}{n} & \frac{\lambda}{n} & \frac{\lambda}{n} & \frac{\lambda}{n} & \frac{\lambda}{n} & \cdots & \frac{\lambda}{n} \\
\hline
1 & 2 & 3 & 4 & 5 & \cdots & n
\end{array}$$

对任务提交时刻的建模

假设单位时间内用户平均提交的任务数量为 λ，那么在单位时间内均匀观察 n 次，每次观察时用户提交任务的概率为 $p = \dfrac{\lambda}{n}$。

$\frac{\lambda}{n}$	$\frac{\lambda}{n}$	$\frac{\lambda}{n}$	$\frac{\lambda}{n}$	$\frac{\lambda}{n}$	$\cdots$	$\frac{\lambda}{n}$
1	2	3	4	5	$\cdots$	n

单位时间内实际提交的任务数量 X 服从二项分布 $X \sim B(n, p)$：

$$P\{X = k\} = C_n^k\, p^k (1 - p)^{n-k}$$

对任务提交时刻的建模

当观察次数 $n \to \infty$ 时，表示在单位时间内持续观察用户提交任务的过程：

$$
\begin{aligned}
P\{X=k\} &= \lim_{n\to\infty} C_n^k p^k (1-p)^{n-k} \\
&= \lim_{n\to\infty} \frac{n(n-1)\ldots(n-(k-1))}{k!} \left(\frac{\lambda}{n}\right)^k \left(1-\frac{\lambda}{n}\right)^{n-k} \\
&= \frac{\lambda^k}{k!} \lim_{n\to\infty} \frac{n(n-1)\ldots(n-(k-1))}{n^k} \left(1-\frac{\lambda}{n}\right)^{-k} \left(1-\frac{\lambda}{n}\right)^n \\
&= \frac{\lambda^k}{k!} e^{-\lambda}
\end{aligned}
$$

对任务提交时刻的建模

当观察次数 $n \to \infty$ 时，表示在单位时间内持续观察用户提交任务的过程：

$$
\begin{aligned}
P\{X=k\} &= \lim_{n\to\infty} C_n^k p^k (1-p)^{n-k} \\
&= \lim_{n\to\infty} \frac{n(n-1)\ldots(n-(k-1))}{k!} \left(\frac{\lambda}{n}\right)^k \left(1-\frac{\lambda}{n}\right)^{n-k} \\
&= \frac{\lambda^k}{k!} \lim_{n\to\infty} \frac{n(n-1)\ldots(n-(k-1))}{n^k} \left(1-\frac{\lambda}{n}\right)^{-k} \left(1-\frac{\lambda}{n}\right)^n \\
&= \frac{\lambda^k}{k!} e^{-\lambda}
\end{aligned}
$$

此时单位时间内实际提交的任务数量 X 服从泊松分布 $X \sim P(\lambda)$ 。

对任务提交时刻的建模

记 $[0, t]$ 时间段内用户提交的任务数量为 $N(t)$，$(s, t]$ 时间段内用户提交的任务数量为 $N(s, t] = N(t) - N(s)$。

对任务提交时刻的建模

记 $[0,t]$ 时间段内用户提交的任务数量为 $N(t)$ ，$(s,t]$ 时间段内用户提交的任务数量为 $N(s,t]=N(t)-N(s)$ 。

初始时刻没有用户提交任务

对任务提交时刻的建模

记 $[0,t]$ 时间段内用户提交的任务数量为 $N(t)$，$(s,t]$ 时间段内用户提交的任务数量为 $N(s,t] = N(t) - N(s)$ 。

- $N(0) = 0$：初始时刻没有用户提交任务

对任务提交时刻的建模

记 $[0,t]$ 时间段内用户提交的任务数量为 $N(t)$，$(s,t]$ 时间段内用户提交的任务数量为 $N(s,t] = N(t) - N(s)$。

- $N(0) = 0$：初始时刻没有用户提交任务

在互不相交的时间段内，用户提交任务的数量相互独立

对任务提交时刻的建模

记 $[0,t]$ 时间段内用户提交的任务数量为 $N(t)$，$(s,t]$ 时间段内用户提交的任务数量为 $N(s,t]=N(t)-N(s)$。

- $N(0)=0$：初始时刻没有用户提交任务
- 独立增量性：在互不相交的时间段内，用户提交任务的数量相互独立

NORTH CHINA ELECTRIC POWER UNIVERSITY 华北电力大学

对任务提交时刻的建模

记 $[0,t]$ 时间段内用户提交的任务数量为 $N(t)$，$(s,t]$ 时间段内用户提交的任务数量为 $N(s,t]=N(t)-N(s)$。

- $N(0)=0$：初始时刻没有用户提交任务
- 独立增量性：在互不相交的时间段内，用户提交任务的数量相互独立
  在长度相等的时间段 t 内，任务提交数量服从相同的概率分布 $P(\lambda t)$

对任务提交时刻的建模

记 $[0,t]$ 时间段内用户提交的任务数量为 $N(t)$，$(s,t]$ 时间段内用户提交的任务数量为 $N(s,t]=N(t)-N(s)$ 。

- $N(0)=0$：初始时刻没有用户提交任务
- 独立增量性：在互不相交的时间段内，用户提交任务的数量相互独立
- 平稳增量性：在长度相等的时间段 t 内，任务提交数量服从相同的概率分布 $P(\lambda t)$

对任务提交时刻的建模

记 $[0, t]$ 时间段内用户提交的任务数量为 $N(t)$，$(s, t]$ 时间段内用户提交的任务数量为 $N(s, t] = N(t) - N(s)$。

泊松过程

- $N(0) = 0$：初始时刻没有用户提交任务
- 独立增量性：在互不相交的时间段内，用户提交任务的数量相互独立
- 平稳增量性：在长度相等的时间段 t 内，任务提交数量服从相同的概率分布 $P(\lambda t)$

对任务提交时刻的建模

记 $[0,t]$ 时间段内用户提交的任务数量为 $N(t)$，$(s,t]$ 时间段内用户提交的任务数量为 $N(s,t]=N(t)-N(s)$ 。

泊松过程

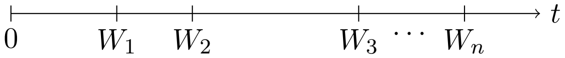
- $N(0)=0$：初始时刻没有用户提交任务
- 独立增量性：在互不相交的时间段内，用户提交任务的数量相互独立
- 平稳增量性：在长度相等的时间段 t 内，任务提交数量服从相同的概率分布 $P(\lambda t)$

因此 $\forall s, N(s,s+t] \sim P(\lambda t)$ ：

$$P\{N(s,s+t]=k\} = \frac{(\lambda t)^k}{k!}e^{-\lambda t}$$

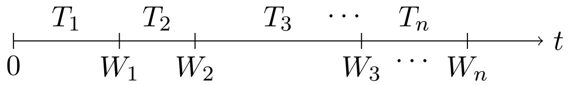
对任务提交时刻的建模

设 W_n 为第 n 个任务提交的时刻



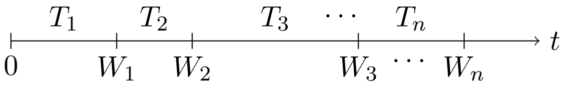
对任务提交时刻的建模

设 W_n 为第 n 个任务提交的时刻，T_n 为第 $n-1$ 个任务与第 n 个任务提交的时间间隔



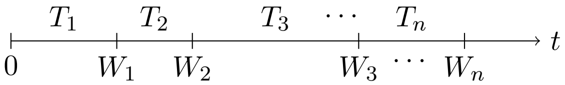
对任务提交时刻的建模

设 W_n 为第 n 个任务提交的时刻，T_n 为第 $n-1$ 个任务与第 n 个任务提交的时间间隔，则 $W_n = \sum_{i=1}^{n} T_i$ 。



对任务提交时刻的建模

设 W_n 为第 n 个任务提交的时刻，T_n 为第 $n-1$ 个任务与第 n 个任务提交的时间间隔，则 $W_n = \sum_{i=1}^{n} T_i$ 。



为了得到用户提交任务的时刻 T_{submit} ，只需要明确 T_n 服从的分布。

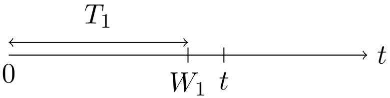
T_n 服从的分布

T_1 服从的分布

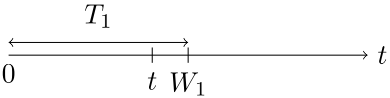
求 T_1 的分布函数：

$$F_{T_1}(t) = P\{T_1 \leqslant t\} = 1 - P\{T_1 > t\} = 1 - P\{N(0,t] = 0\} = 1 - e^{-\lambda t}$$

$$f_{T_1}(t) = F'_{T_1}(t) = \lambda e^{-\lambda t}$$



(a) $T_1 \leqslant t$



(b) $T_1 > t$

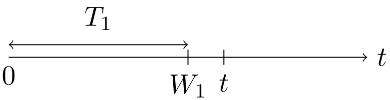
T_n 服从的分布

T_1 服从的分布

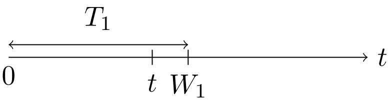
求 T_1 的分布函数：

$$F_{T_1}(t) = P\{T_1 \leqslant t\} = 1 - P\{T_1 > t\} = 1 - P\{N(0,t] = 0\} = 1 - e^{-\lambda t}$$

$$f_{T_1}(t) = F'_{T_1}(t) = \lambda e^{-\lambda t}$$



(a) $T_1 \leqslant t$



(b) $T_1 > t$

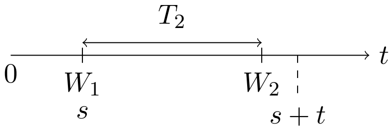
T_1 服从指数分布：$T_1 \sim E(\lambda)$ 。

T_n 服从的分布

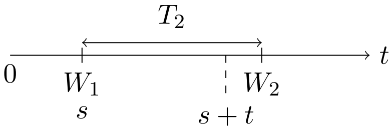
T_2 服从的分布

求 T_2 的分布函数（假设在任意的 s 时刻，第一个任务已经提交）：

$$
\begin{aligned}
F_{T_2}(t) &= P\{T_2 \leqslant t\} = 1 - P\{T_2 > t\} \\
&= 1 - P\{N(s, s+t] = 0 | N(0, s] = 1\} \\
&= 1 - P\{N(s, s+t] = 0\} \qquad \text{(0, s]与(s, s+t]两时间段互不相交，相互独立} \\
&= 1 - e^{-\lambda t}
\end{aligned}
$$



(a) $T_2 \leqslant t$



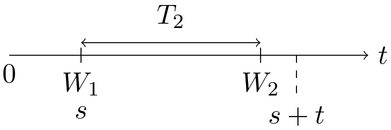
(b) $T_2 > t$

T_n 服从的分布

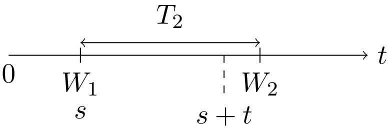
T_2 服从的分布

求 T_2 的分布函数（假设在任意的 s 时刻，第一个任务已经提交）：

$$\begin{aligned} F_{T_2}(t) &= P\{T_2 \leqslant t\} = 1 - P\{T_2 > t\} \\ &= 1 - P\{N(s, s+t] = 0 | N(0, s] = 1\} \\ &= 1 - P\{N(s, s+t] = 0\} \qquad (0, s]\text{与}(s, s+t]\text{两时间段互不相交，相互独立} \\ &= 1 - e^{-\lambda t} \end{aligned}$$



(a) $T_2 \leqslant t$



(b) $T_2 > t$

根据无记忆性，$T_2, \ldots, T_n$ 均服从参数为 λ 的指数分布。

任务提交时间序列和总计算量

因为 $T_n \sim E(\lambda)$ ，所以代码中对参数为 λ 的指数分布进行采样即可得到任务提交的时间间隔。将时间间隔累加，即可得到任务提交的时间序列，对应 T_{submit} 。

```
# 生成时间间隔
intervalT = stats.expon.rvs(scale=1 / lamda, size=self.jobNum)
# 对时间间隔累加得到提交时间
self.arrival_Times = np.around(intervalT.cumsum(), decimals=3)
```

总计算量 $reqCom$ 通过正态分布采样。

```
self.jobsMI = np.random.normal(self.jobMI, self.jobMI_std, self.jobNum)
self.jobsMI = self.jobsMI.astype(int)
```

虚拟机属性

每个虚拟机具有如下属性：

- vID：虚拟机编号
- $vCom$：单核计算速度
- $vAcc$：多核加速系数
- T_{idle}：虚拟机处理完最后一个任务的时刻
- $vType$：虚拟机类型
  - 高性能计算
  - 高性能I/O
- vSC：虚拟机启动开销
- vEC：虚拟机运行开销

任务在虚拟机中的运行过程

当一个任务在 T_{submit} 时刻提交给一个虚拟机时，可以得到该任务的等待时间：

$$T_{wait} = \max\{T_{idle} - T_{submit}, 0\}$$

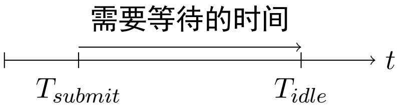
(a) $T_{idle} > T_{submit}$

(b) $T_{idle} \leqslant T_{submit}$

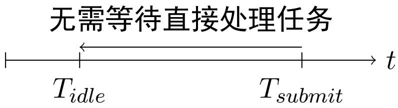
任务在虚拟机中的运行过程

当一个任务在 T_{submit} 时刻提交给一个虚拟机时，可以得到该任务的等待时间：

$$T_{wait} = \max\{T_{idle} - T_{submit}, 0\}$$



(a) $T_{idle} > T_{submit}$



(b) $T_{idle} \leqslant T_{submit}$

当虚拟机开始执行任务时，任务的执行时间：

$$T_{exe} = \frac{Type \oplus vType + 1}{2} \cdot \frac{reqCom}{vCom \cdot vAcc}$$

运行结果指标

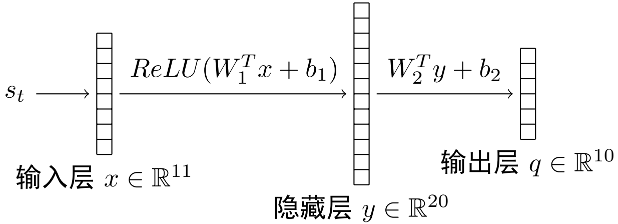
- 任务响应时间：$T_{rep} = T_{wait} + T_{exe}$
- 是否满足 QoS 要求：$success = \begin{cases} 1, & T_{rep} \leqslant QoS \\ 0, & \text{otherwise} \end{cases}$
- 虚拟机费用：$cost = vSC + vEC \cdot T_{exe}$

Deep Q-Network 结构

代码默认存在 10 台虚拟机，输入状态 s_t 为当前提交任务的类型和 10 台虚拟机的 T_{idle}：

$$s_t = \left[Type,\ T_{idle}^{(1)},\ T_{idle}^{(2)},\ \ldots,\ T_{idle}^{(10)}\right]^T$$

DQN输出在当前状态 s_t 下，对分配到每台虚拟机的总共 10 个动作的评分。



其中两层全连接层的参数：$W_1 \in \mathbb{R}^{11\times 20}$, $b_1 \in \mathbb{R}^{20}$, $W_2 \in \mathbb{R}^{20\times 10}$, $b_2 \in \mathbb{R}^{10}$ 。

Deep Q-Network 训练方法

奖励计算函数：$reward = (1 + e^{\xi - cost}) \cdot \frac{T_{exe}}{T_{rep}}$，其中 ξ 为超参数。

Deep Q-Network 训练方法

奖励计算函数：$reward = (1 + e^{\xi - cost}) \cdot \frac{T_{exe}}{T_{rep}}$，其中 ξ 为超参数。

- ϵ-greedy：以 ϵ 的概率随机决策，以 $1-\epsilon$ 的概率使用DQN决策。随机决策可以探索DQN没有学到的状态，每次学习后减小 ϵ 。

Deep Q-Network 训练方法

奖励计算函数：$reward = (1 + e^{\xi - cost}) \cdot \dfrac{T_{exe}}{T_{rep}}$，其中 ξ 为超参数。

- ϵ-greedy：以 ϵ 的概率随机决策，以 $1 - \epsilon$ 的概率使用DQN决策。随机决策可以探索DQN没有学到的状态，每次学习后减小 ϵ 。
- experience replay：将过去的决策轨迹 (s_t, a_t, r_t, s_{t+1}) 存入replay memory中。DQN每次学习时从replay memory中随机选取一组样本用来更新参数，以消除学习连续样本所带来的相关性。

Deep Q-Network 训练方法

奖励计算函数：$reward = (1 + e^{\xi - cost}) \cdot \frac{T_{exe}}{T_{rep}}$，其中 ξ 为超参数。

- ϵ-greedy：以 ϵ 的概率随机决策，以 $1-\epsilon$ 的概率使用DQN决策。随机决策可以探索DQN没有学到的状态，每次学习后减小 ϵ 。
- experience replay：将过去的决策轨迹 (s_t, a_t, r_t, s_{t+1}) 存入replay memory中。DQN每次学习时从replay memory中随机选取一组样本用来更新参数，以消除学习连续样本所带来的相关性。
- fixed Q-target：DQN训练时需要将 s_t 和 s_{t+1} 都输入网络中。如果仅使用一个网络，更新网络参数的操作会使 s_t 和 s_{t+1} 的输出向相同的方向移动。通过引入参数相对固定的 target 网络用来接收 s_{t+1} 的输入后，固定了参数更新的目标，加快了收敛速度。

Deep Q-Network 学习流程

确定环境参数和DRL的超参数（代码默认值）：

- 任务：
  - 任务提交速度 $\lambda = 20$
  - 任务类型比例 CPU : I/O $= 9 : 1$
  - 任务平均总计算量 $\mu = 200$
  - 任务总计算量的标准差 $\sigma = 20$
  - 任务提交总数 8000
  - 响应时间指标 $QoS = 0.25$
- 虚拟机：
  - 虚拟机类型（5台计算型，5台I/O型）$[0, 0, 0, 0, 0, 1, 1, 1, 1, 1]$
  - 虚拟机费用 $cost = [1, 1, 2, 2, 4, 1, 1, 2, 2, 4]$
  - 单核计算速度 $vCom = 1000$
  - 多核加速系数 $vAcc = [1, 1, 1.1, 1.1, 1.2, 1, 1, 1.1, 1.1, 1.2]$

Deep Q-Network 学习流程

确定环境参数和DRL的超参数（代码默认值）：

- DRL超参数：
  - ϵ-greedy $\epsilon = 0.9$，每次学习后减小 0.006
  - replay memory size $N = 800$
  - minibatch size $S = 30$
  - Q-target 网络参数更新间隔 50 步
  - 奖励函数超参数 $\xi = 1.5$
  - 学习率 $\gamma = 0.01$

Deep Q-Network 学习流程

设定环境参数、DRL超参数，随机初始化DQN参数，赋予target网络相同的参数;
foreach *Episode* **do**
  重置环境;
  foreach *Step* **do**
    对于状态 s_t 根据 ϵ-greedy 策略得到动作 a_t;
    执行动作 a_t 后环境变为 s_{t+1} 并得到奖励 r_t;
    将轨迹 (s_t, a_t, r_t, s_{t+1}) 存入replay memory;
    if *Step* $>$ 开始学习步数 **then**
      从replay memory中随机抽取 30 个样本作为minibatch;
      foreach *sample in minibatch* **do**
        将 s_t 传入Q-network得到 a_t 对应的 Q_{value};
        将 s_{t+1} 传入target-network得到输出的最大值 Q_{target};
        根据损失函数 $Loss = (Q_{value} - (r_t + \gamma \cdot Q_{target}))^2$ 使用梯度下降法更新Q-network;
      end
      if *Step % 50 = 0* **then**
        使用Q-network的参数更新target-network;
      end
      减小 ϵ;
    end
  end
end

作为Baseline的3种算法

- Random：将任务随机分配给任意一台虚拟机
- Round-Robin：将任务轮流分配给每台虚拟机
- Earliest：将任务分配给最先完成任务（即 T_{idle} 最小）的虚拟机

Fin.