Математическое обоснование алгоритма PPO для распределения задач по CPU

Мы точно всё успеем!

19 мая 2025 г.

1 Постановка задачи

Рассмотрим ацикличный ориентированный граф вычислений $G = (V, E)$, где:

- $V = \{v_1, ..., v_n\}$ – множество вершин (операций)
- $E \subseteq V \times V$ – множество ориентированных ребер (зависимостей)
- Каждой вершине v_i соответствует среднее время выполнения $t_i \in \mathbb{R}^+$

Цель: найти оптимальную последовательность выполенения операций, минимизирующее общее время вычислений при условии:

1. Операция может быть выполнена, только если все ее предшественники завершены
2. Доступно $k \geq 1$ параллельных вычислительных единиц

2 Возможные проблемы и трудности задачи

- Задача поиска минимума является NP трудной, поэтому решение задачи нужно находить приближенное.
- Пространство возможных состояний также слишком велико. Решением может являтся хранение состояния как эмбединга, значит необходима апроксимизация.

- Мы обучаем модель на среднем времени вычисления операции в вершине, но в реальности время вычисления может меняться. В данной мат. модели мы это не учитываем.

- Из проблемы выше следует, что выдавать цепочку вычислений заранее может быть не верно, мы упускаем всю информацию о вычислении в данный момент.

- Длина запросов влияет на граф вычислений(меняет его). Из-за этого оптимальная цепочка может меняться.

3 Формализация как markov decision process (MDP)

3.1 State representation

Состояние представляется вектором прогресса операций:

$$s_t \in [0,1]^n, \quad s_t^{(i)} = \begin{cases} 1, & \text{операция } v_i \text{ завершена} \\ \frac{\text{пройденное время } v_i}{t(v_i)}, & \text{операция } v_i \text{ в процессе} \\ 0, & \text{иначе} \end{cases} \tag{1}$$

3.1.1 Action

На каждом шаге агент выбирает **одну** операцию для запуска:

$$a_t \in \{0, 1, \ldots, n\} \tag{2}$$

3.2 Restrictions on actions

Action $a_t = i$ допустимо только если:

1. Операция v_i еще не выполнена ($s_t^{(i)} \neq 1$)

2. Все предшественники выполнены ($\forall v_j \in \mathrm{pred}(v_i) : s_t^{(j)} = 1$)

3.3 Reward

Reward выдается в конце вычисления графа:

$$R = -time \tag{3}$$

Где time - время выполнения всего графа

3.4 Environment

Environment - наш граф $G = (V, E)$

3.5 Agent

В качестве алгоритма будем использовать PPO

4 PPO algorithm

4.1 Policy (Actor)

Для каждого ядра CPU агент предсказывает вероятность его использования как независимое событие Бернулли:

$$\pi_\theta(a|s) = \prod_{i=1}^{N} p_i^{a_i}(1-p_i)^{1-a_i} \tag{4}$$

где:

- N – количество ядер CPU (`num_cpu_cores`),
- $p_i = \sigma(f_\theta(s)_i)$ – вероятность использования ядра i,
- f_θ – нейронная сеть актора.

4.2 Critic

Value function:

$$V_\theta(s) = g_\theta(s) \tag{5}$$

где g_θ – нейронная сеть критика.

4.3 Loss PPO

Loss function с ограничением:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right] \tag{6}$$

где:

- $r_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$,
- $A_t = R_t - V_\theta(s_t)$,
- $\epsilon = 0.2$ (`eps_clip`).

4.4 Loss function in details

$$L(\theta) = L^{CLIP}(\theta) + c_1 L^{VF}(\theta) - c_2 H(\pi_\theta(s_t)) \tag{7}$$

где:

- Critic loss:
$$L^{VF}(\theta) = \frac{1}{2}\|V_\theta(s_t) - R_t\|^2 \tag{8}$$

- Entropy regularization(мне казалось у этого термина есть другое название?):

$$H(\pi_\theta(s_t)) = -\sum_{i=1}^{N} [p_i \log p_i + (1-p_i)\log(1-p_i)] \tag{9}$$

- $c_1 = 1$, $c_2 = 0.01$

4.5 Discounting

$$R_t = \sum_{k=t}^{T} \gamma^{k-t} r_k, \quad \gamma = 0.99 \tag{10}$$

4.6 Normalization

$$\hat{A}_t = \frac{A_t - \mu_A}{\sigma_A + 10^{-7}} \tag{11}$$

4.7 Update

$$\theta_{\text{actor}} \leftarrow \theta_{\text{actor}} - \alpha_{\text{actor}} \nabla_\theta L(\theta)$$
$$\theta_{\text{critic}} \leftarrow \theta_{\text{critic}} - \alpha_{\text{critic}} \nabla_\theta L(\theta)$$

где $\alpha_{\text{actor}} = 3 \times 10^{-4}$, $\alpha_{\text{critic}} = 1 \times 10^{-3}$.

5 Improvements and additions

5.1 Возможный выбор устройства

Введем:

- Множество вычислителей $M = \{m_1, ..., m_k\}$

- Матрицу времени выполнения: $T \in \mathbb{R}^{n \times k}$, где $T_{i,j}$ - время выполнения операции v_i на вычислителе m_j
- Ограничение: одна операция на вычислитель одновременно

5.2 Расширенное представление состояния

Состояние $s_t \in [0,1]^{n+k}$, где:

- Первые n элементов: прогресс операций как ранее
- Последние k элементов: загрузка вычислителей

$$s_t^{(n+j)} = \begin{cases} 0, & \text{вычислитель } m_j \text{ свободен} \\ \frac{\text{осталось времени}}{T_{i,j}}, & \text{если } m_j \text{ выполняет } v_i \end{cases}$$

5.3 Модифицированные действия

Действие $a_t = (i,j) \in \{0,1,...,n\} \times \{1,...,k\}$:

- запуск операции v_i на вычислителе m_j

Ограничения:

1. $s_t^{(j+n)} = 0$ (вычислитель свободен)
2. $\forall v_p \in \text{pred}(v_i) : s_t^{(p)} = 1$

NOTE: Так формулируется задача, когда мы имеем разные вычислители, однако зачастую будет 2-3 типа вычислителей(cpu, gpu, tpu), каждый вычислитель будет принадлежать одному из классов и при этом в рамках класса будут одинаковые характеристики(время выполнения). Также будет иметь смысл отделять и или обозначать принадлежность вычислителя классу в векторном представлении.