Towards Efficient Resource Management in MEC: A Hybrid TRPO-SAC Solution for Computing, Caching, and Pushing

Deep Learning

Group name: PredictX

Indian Institute of Technology Guwahati

Contents

Introduction

Mobile Edge Computing (MEC) Networks

Mobile edge computing (MEC) networks bring computing and storage capabilities closer to edge devices, which:

- Reduces latency
- Improves network performance

Problem Statement

MEC systems need to optimize multiple things to be efficient:

- **Computing:** Performing tasks (e.g., processing requests) quickly and accurately.
- **Pushing:** Sending the right data to users before they ask for it (proactive).
- **Caching:** Storing frequently requested data, so it's immediately available when users request it.
  To make the system efficient, these three aspects (computing, pushing, caching) must be jointly optimized.

Goal: Minimize costs (e.g., transmission and computation) while maintaining a good user experience (e.g., low latency).

MDP Formulation for MEC System Optimization

To optimize this the problem is formulated as **Markov Decision Process (MDP)**:

- The system makes decisions based on its current state.
- The objective is to maximize a long-term reward (in this case, minimizing costs over time).

Since it's a complex, ongoing problem, it's framed as an **infinite-horizon discounted-cost MDP**:

- **Infinite horizon:** The system plans over an indefinite period.
- **Discounted cost:** Future rewards or costs are weighted less than immediate ones.

Deep Reinforcement Learning Framework for MEC

We use a **Deep Reinforcement Learning (DRL)**-based framework that enables the dynamic orchestration of:

- Computing
- Pushing
- Caching

Through the deep networks embedded in the DRL structure, our framework can:

- Implicitly predict user future requests
- Push or cache the appropriate content to effectively enhance system performance

One issue encountered when considering these three functions collectively is the **curse of dimensionality** for the action space.

The Curse of Dimensionality Explained - Part 1

Basic Concept:
The "curse of dimensionality" refers to the phenomenon where the complexity of a problem increases dramatically as the number of dimensions (or variables) increases. In action spaces, as you add more options for each function (computing, caching, pushing), the total combinations grow exponentially, complicating exploration and optimization.

Example Breakdown:

- **Defining Actions:**
  - Computing: 3 actions (Low, Medium, High)
  - Caching: 4 actions (No Cache, Cache 25%, Cache 50%, Cache 100%)
  - Pushing: 5 actions (No Push, Push 20%, Push 40%, Push 60%, Push 80%)

The Curse of Dimensionality Explained - Part 2

- **Total Combinations:**

$$\text{Total} = 3 \times 4 \times 5 = 60$$

- **Increasing Complexity:** Adding:
  - 4 actions for computing,
  - 5 actions for caching,
  - 6 actions for pushing,

  Results in:

$$4 \times 5 \times 6 = 120$$

- **Exponential Growth:** Adding more action will further increase the complexity and exponential grow the action space

Solution Approach

To solve this, they:

- Relaxed the discrete action space into a **continuous space:**
  Instead of having a finite number of specific actions, the system can take actions on a continuous scale.
- Used **hybrid approach that combines Trust Region Policy Optimization (TRPO) with Soft Actor-Critic (SAC)** learning to solve the optimization problem:
- Used **vector quantization** to convert the continuous actions back into discrete actions:
  This step is necessary for the actual system implementation.
- Added an **action correction method** to further compress the action space and speed up learning.

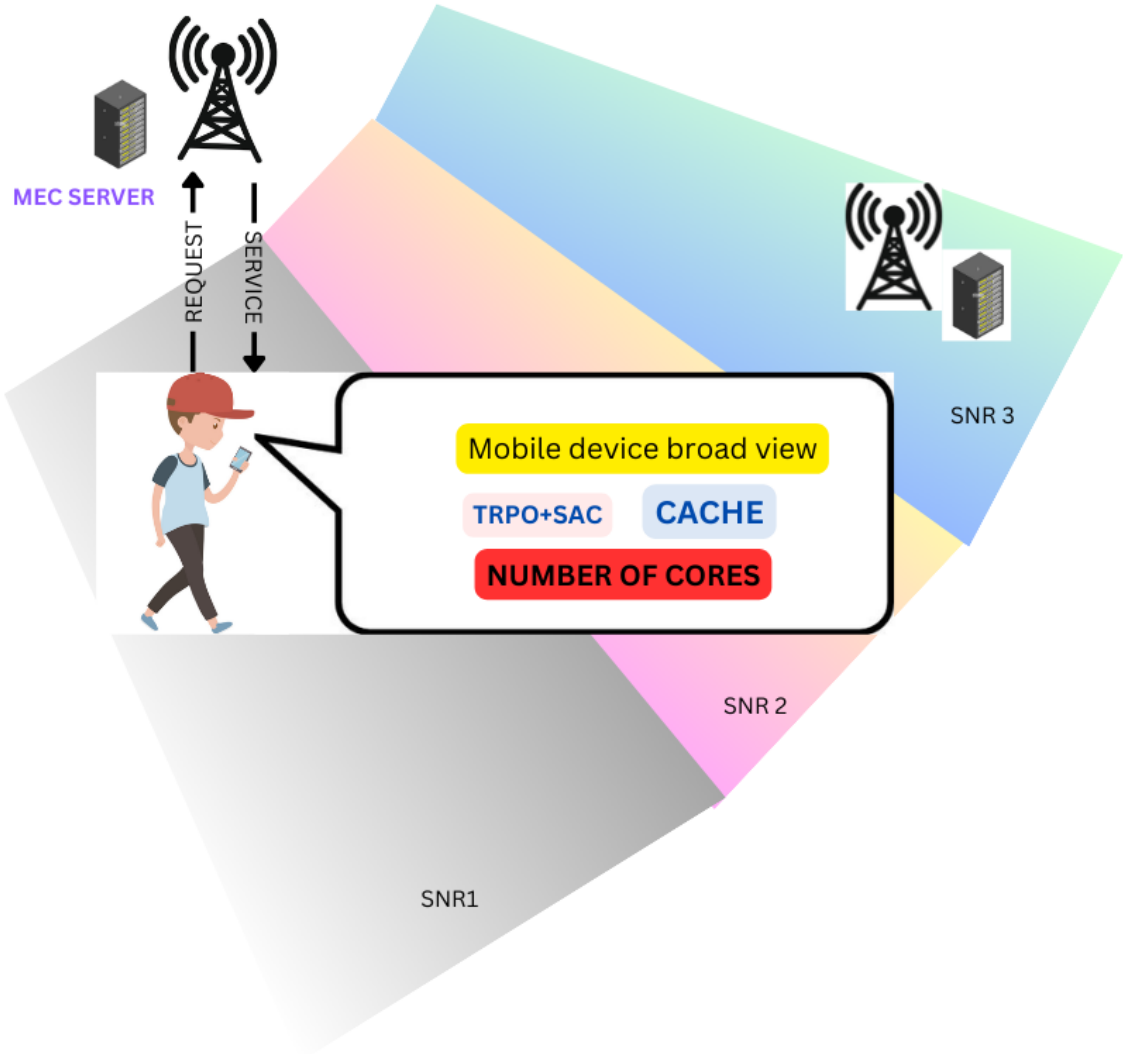
Contribution

- **Hybrid TRPO-SAC Framework for Enhanced Stability and Exploration:**
  - We develop a hybrid reinforcement learning (RL) model that integrates TRPO's stable policy updates with SAC's entropy-driven exploration. This synergy addresses exploration inefficiencies and policy instability in dynamic, high-dimensional MEC environments, enabling more accurate adaptation to fluctuating user demands.
- **Optimization for Computing, Caching, and Communication:**
  - Our model provides a unified approach to optimize caching, computation, and data pushing resources, improving latency and bandwidth efficiency for modern mobile applications. This joint optimization directly enhances user-perceived quality of experience (QoE) by better managing MEC resources.

Contribution cont

- **Performance Gains in Latency and Resource Utilization:**
  - Through extensive experiments, we demonstrate that the TRPO-SAC hybrid significantly reduces latency and enhances cache hit rates and computational efficiency over both standalone SAC and traditional heuristic approaches, underscoring its effectiveness in meeting MEC network performance targets.

Basic Outline



System Outline

Key Components of the MEC Network Model

Network Structure:

- The model starts with a simple setup: one MEC server and one mobile device.
- This simplicity allows for a focus on the core aspects of the optimization problem without the complexity of multiple users initially.
- The model can be extended to accommodate multiple users by aggregating their objectives and resource constraints.

Cache Sizes

Cache Sizes:

- **MEC Server:** It has a large cache capable of storing all the data necessary for the tasks requested by the mobile device.
- **Mobile Device:** It has a limited cache size C (in bits), which restricts how much data it can hold.

Computational Resources

Computational Resources:

- The mobile device is equipped with multi-core computing capabilities:
  - Let M represent the number of computing cores.
  - Let f_D denote the computation frequency (in cycles/second) of each core.
- This indicates that the device can handle tasks concurrently, improving its efficiency.

Time Framework

Time Framework:

- The system operates over an infinite time horizon, divided into time slots indexed by $t = 0, 1, 2, \ldots$ with a fixed duration of τ seconds for each slot.
- This structure is essential for scheduling tasks and optimizing actions over time.

Task Characteristics

Task Characteristics:

- At the beginning of each time slot, the mobile device submits a task request that is delay-intolerant.
- This means that the task must be completed within the time slot to ensure optimal user experience, especially for applications sensitive to delays like augmented reality (AR), virtual reality (VR), real-time communication, and streaming.

Dynamic Transmission Rate

Dynamic Transmission Rate:

- The data transmission rate between the mobile device and the MEC server is dynamic and can vary over time due to the mobility of the device.
- This is modeled using the signal-to-noise ratio (SNR) based on Shannon's theory, which assesses the quality of the communication link.

Task Set Definition

Task Set Definition:

- Assuming that the mobile device will request a total of F tasks, the task set F is defined as:

$$F\Delta = \{1, 2, \ldots, f, \ldots, F\}$$

- Each task $n_f \in F$ is characterized by a 4-item tuple:

$$(I_f, O_f, w_f, \tau)$$

  - I_f (in bits): Size of the input data generated from the Internet which can be cached.
  - O_f (in bits): Size of the output data after the computation is completed.
  - w_f (in cycles/bit): Required computation cycles per bit.
  - τ (in seconds): Maximum service latency.

System State

Request State

1) Request State:

- At each time slot t, the mobile device submits a single task request.
- The request state at time t is denoted by $A(t) \in F$, where $A(t) = f$ signifies that task f in set F is being requested.
- The size of F is F.
- The evolution of requested tasks is modeled using a first-order F-state Markov chain:

$$A(t) : t = 0, 1, 2, \ldots$$

- Each state corresponds to a distinct task, and the probability of transitioning to state $j \in F$ at time $t+1$, given that the request state at time t is $i \in F$, is:

$$\Pr[A(t+1) = j | A(t) = i]$$

Request State Con't

- The transition probability matrix is denoted by Q with elements $q_{i,j} \equiv \Pr[A(t+1) = j | A(t) = i]$.
- The system is represented by an irreducible Markov chain, ensuring that any state can be reached from any other with a non-zero probability.
- The limiting distribution is denoted by $p \equiv (p_f)_{f \in F}$, where:

$$p_f = \lim_{t \to \infty} \Pr[A(t) = f]$$

Cache State

Cache State:

- Let $S_I^f(t) \in \{0, 1\}$ denote the indicator of the cache state of the input data for task f:
  - $S_I^f(t) = 1$ means the input data for task f is cached; $S_I^f(t) = 0$ means it is not cached.
- Similarly, let $S_O^f(t) \in \{0, 1\}$ denote the indicator for the output data:
  - $S_O^f(t) = 1$ means the output data for task f is cached; $S_O^f(t) = 0$ means it is not cached.
- The cache size of the mobile device is denoted by C (in bits), with the cache size constraint given by:

$$\sum_{f=1}^{F} \left(I_f S_I^f(t) + O_f S_O^f(t) \right) \leq C$$

Cache State Con't

- This constraint ensures that the sum of the sizes of input and output data cached for all tasks does not exceed the cache size.
- The cache state of the mobile device at time slot t is denoted by:

$$S(t) \equiv (S_I^f(t), S_O^f(t))_{f \in F} \in S$$

- The cache state space S is defined as:

$$S \equiv \{(S_I^f, S_O^f)_{f \in F} \in \{0,1\}^F \times \{0,1\}^F : \sum_{f \in F} I_f S_I^f + O_f S_O^f \leq C\}$$

Cache State Con't

- The cardinality of S is bounded by:

$$N_{\min} \equiv \frac{C}{\max_{f \in F}(I_f, O_f)}, \quad N_{\max} \equiv \frac{C}{\min_{f \in F}(I_f, O_f)}$$

- Thus, the cardinality of S is bounded by $F^{N_{\min}}$ and $F^{N_{\max}}$.

System State

3) System State:

- At time slot t, the system state consists of both the request state and the cache state:

$$X(t) \equiv (A(t), S(t)) \in F \times S$$

System Action

Overview of Reactive Computation Action

Reactive Computation Action:

- At each time slot t, the reactive transmission bandwidth cost $B_R(t)$ and the reactive computation energy cost $E_R(t)$ are defined.
- The task request $A(t)$ is served based on the current system state $X(t) = (A(t), S(t))$.

Cached Output Data

Case 1: Output Cached

- If the cache state $S_O^{A(t)}(t) = 1$:
  - The output of task $A(t)$ is already cached locally.
  - Retrieval can occur without any transmission or computation.
  - **Result:**
    - Delay is negligible.
    - Reactive computation energy cost $E_R(t) = 0$.
    - Reactive transmission cost $B_R(t) = 0$.

Cached Input and Computation Action

Scenario: Cached Input, Missing Output

- Assume $S_I^{A(t)}(t) = 1$ and $S_O^{A(t)}(t) = 0$:
  - Task $A(t)$ is computed directly using locally cached input data.
  - Number of computation cores allocated: $c_{R,f}(t) \in \{1, \ldots, M\}$.
  - For non-requested tasks: $c_{R,f}(t) = 0$.
- Ensure task completion within latency τ:

$$\frac{I_{A(t)} w_{A(t)}}{\tau} \leq c_{R,A(t)}(t) f_D$$

Reactive Computation Costs

Energy and Transmission Costs

- Energy consumed for one cycle:

$$E_{cycle} = \mu c_{R,f}^2(t) f_D^2$$

- Reactive computation energy cost $E_R(t)$:

$$E_R(t) = \mu c_{R,A(t)}^2(t) f_D^2 I_{A(t)} w_{A(t)}$$

- Reactive transmission cost $B_R(t)$:
  - $B_R(t) = 0$ when input data is cached.

Downloading Input Data from MEC Server

Scenario: No Cached Data

- If $S_I^{A(t)}(t) = 0$ and $S_O^{A(t)}(t) = 0$:
  - The mobile device must download the input data for task $A(t)$ from the MEC server.
  - Let SNR(t) be the SNR value of the data transmission link at time t.

Latency Expression and Constraints

Latency Calculation

- The required latency can be expressed as:

$$\frac{I_{A(t)}}{B_R(t)\log_2(1+\mathrm{SNR}(t))}+\frac{I_{A(t)}w_{A(t)}}{c_{R,A(t)}(t)f_D}$$

- To satisfy the latency constraint:

$$\frac{I_{A(t)}}{B_R(t)\log_2(1+\mathrm{SNR}(t))}+\frac{I_{A(t)}w_{A(t)}}{c_{R,A(t)}(t)f_D}\leq\tau$$

Reactive Computation and Cost Expressions

Reactive Transmission and Computation Costs

- Minimum reactive transmission cost $B_R(t)$:

$$B_R(t) = \left(1 - S_I^{A(t)}(t)\right)\left(1 - S_O^{A(t)}(t)\right) \cdot \frac{I_{A(t)}}{\tau - \frac{I_{A(t)}\, w_{A(t)}}{c_{R,A(t)}(t) f_D}} \cdot \frac{1}{\log_2(1 + \mathrm{SNR}(t))}$$

- Reactive computation energy cost $E_R(t)$:

$$E_R(t) = \left(1 - S_O^{A(t)}(t)\right) \mu c_{R,A(t)}^2(t) f_D^2 I_{A(t)}\, w_{A(t)}$$

- The reactive computation action $c_{R,f}(t)$ must satisfy:

$$c_{R,f}(t) \leq 1(A(t) = f) \cdot \left(1 - S_O^f(t)\right) M$$

Proactive Transmission and Cache Update Actions

Let $b^f(t) \in \{0, 1\}$ denote the binary decision variable for task $f \in F$, where $b^f(t) = 1$ indicates that the remote input data of task f is pushed to the mobile device, and $b^f(t) = 0$ otherwise.

$$\sum_{f=1}^{F} I^f b^f(t)\tau \leq B_P(t) \log_2(1 + \mathrm{SNR}(t)),$$

where $B_P(t)$ denotes the proactive transmission bandwidth cost. Thus, the minimum proactive transmission cost can be expressed as:

$$B_P(t) = \sum_{f=1}^{F} I^f b^f(t) \frac{\tau}{\log_2(1 + \mathrm{SNR}(t))}.$$

In summary, the system pushing action under system state $b \equiv (b^f)_{f \in F} \in \{0, 1\}^F$. The size of the system pushing action space under system state X is 2^F.

Cache updation in proactive pushing

The cache state of each task $f \in F$ is updated according to:

$$S_I^f(t+1) = S_I^f(t) + \Delta s_I^f(t), \quad S_O^f(t+1) = S_O^f(t) + \Delta s_O^f(t),$$

where $\Delta s_I^f(t) \in \{-1, 0, 1\}$ and $\Delta s_O^f(t) \in \{-1, 0, 1\}$ denote the update action for the cache state of the input and output data of task f, respectively. Then, we have $\forall f \in F$:

$$-S_I^f(t) \leq \Delta s_I^f(t) \leq \min\left(b^f(t) + c_R^f(t), 1 - S_I^f(t)\right),$$

$$-S_O^f(t) \leq \Delta s_O^f(t) \leq \min\left(c_R^f(t), 1 - S_O^f(t)\right),$$

$$\sum_{f=1}^{F} I^f\left(S_I^f(t) + \Delta s_I^f(t)\right) + O^f\left(S_O^f(t) + \Delta s_O^f(t)\right) \leq C.$$

In summary, let $\Delta s \equiv (\Delta s_I^f, \Delta s_O^f)_{f \in F} \in \Pi_{\Delta s}(X)$ denote the system cache update action, where:

$$\Pi_{\Delta s}(X) \equiv \left\{(\Delta s_I^f, \Delta s_O^f)_{f \in F} \in \{-1, 0, 1\}^F \times \{-1, 0, 1\}^F\right\}.$$

System Action

The system action at each time slot is a combination of three distinct actions: reactive computation, pushing, and cache update. This combination is represented as:

$$(c_R, b, \Delta s) \in \Pi(X),$$

where $\Pi(X)$ is the system action space under the current system state X, defined as:

$$\Pi(X) \equiv \Pi_R^C(X) \times \{0,1\}^F \times \Pi_{\Delta s}(X).$$

Here,

- c_R represents the reactive computation action,
- b denotes the pushing action for tasks,
- Δs indicates the cache update actions.

Overall System Cost

The overall system cost at each time slot t is composed of two main components:

- **Transmission Bandwidth Cost:**

$$B(t) = B_R(t) + B_P(t)$$

where:

  - $B_R(t)$: Reactive transmission cost based on the need to transmit data reactively.
  - $B_P(t)$: Proactive transmission cost for data pushed to devices before needed.

- **Computation Energy Cost:**

$$E(t) = E_R(t)$$

where $E_R(t)$ indicates the reactive computation energy cost incurred during local task execution.

Overall Cost Cont

- **System Cost Calculation:**

$$C(t) = B(t) + \lambda E(t)$$

The term $\lambda E(t)$ introduces a weighting factor λ that determines the relative importance of computation versus transmission costs, emphasizing energy costs with higher λ values and bandwidth costs with lower values.

Problem Formulation

Joint Computing, Pushing, and Caching Policy

Given an observed system state X, the joint reactive computing, transmission, and caching action, denoted as $(c_R, b, \Delta s)$, is determined by the following policy:

Definition 1: *Stationary Joint Computing, Pushing, and Caching Policy.* A stationary joint computing, pushing, and caching policy π is a mapping from system state X to system action $(c_R, b, \Delta s)$:

$$(c_R, b, \Delta s) = \pi(X) \in \Pi(X)$$

From properties of $\{A(t)\}$ and $\{S(t)\}$, the induced system state process $\{X(t)\}$ under policy π is a controlled Markov chain.

The expected total discounted cost $\phi(\pi)$ is given by:

$$\phi(\pi) \stackrel{\Delta}{=} \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \gamma^t E[B(t) + \lambda E(t)]$$

Joint Computing, Pushing, and Caching Policy Cont

where:

- T is the length of the request process,
- γ is the discount factor,
- $B(t)$ is the transmission bandwidth cost,
- $E(t)$ is the computation energy cost,
- λ is the weight balancing the two costs.

Problem 1: *Joint Computing, Pushing and Caching Policy Optimization.*

- The goal is to find the optimal policy $\phi(\pi)$ that minimizes the expected total discounted cost:

$$\phi^* \triangleq \min_{\pi} \phi(\pi) \quad \text{s.t.} \quad \pi(X) \in \Pi(X), \quad \forall X \in F \times S.$$

TRPO-SAC System State and Action

TRPO-SAC System State and Action

The system state x of the TRPO-SAC algorithm is defined to align with the system state X in the formulated problem:

$$x = X = (A(t), S(t))$$

with vector size $2F + 1$.

The action space $(c_R, b, \Delta s)$ is adjusted for continuous-action settings:

$$a = (\bar{c}_R, \bar{b}, \bar{\Delta} s) \in \Pi(\bar{X}) \triangleq \Pi_{\bar{R}_C}(X) \times [0, 1]^F \times \Pi_{\bar{\Delta} s}(X).$$

Here:

$$\Pi_{\bar{R}_C}(X) \triangleq \{(c_{R,f})_{f \in F} \in [0, M]^F : (2)\}$$

and

$$\Pi_{\bar{\Delta} s}(X) \triangleq \left\{ \left(\Delta s_f^I, \Delta s_f^O\right)_{f \in F} \in [-1, 1]^F \times [-1, 1]^F : (9), (10), (11) \right\}.$$

SAC System State and Action

Continuous Version of Actions: To adapt TRPO-SAC, we define a continuous action:

$$a = (\bar{c}_R, \bar{b}, \bar{\Delta}s) \in \Pi(\bar{X})$$

with:

- $\bar{c}_R \in [0, M]$: Continuous range for computation actions.
- $\bar{b} \in [0, 1]$: Binary pushing decisions.
- $\bar{\Delta}s \in [-1, 1]$: Cache state changes.

TRPO-SAC Learning

TRPO-SAC combines TRPO's trust region policy optimization** with SAC's entropy regularization. The two work together to provide stable learning with exploration.

- **TRPO**: Ensures stable policy updates by constraining the KL-divergence between the new and old policies, preventing large, unstable policy updates. - **SAC**: Uses entropy maximization to encourage exploration and improve the quality of learned policies.

The TRPO objective is:

$$J_{\text{TRPO}}(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(x_t, a_t) \sim \rho_\pi} \left[r(x_t, a_t) \right],$$

subject to the KL-divergence constraint:

$$D_{\text{KL}}(\pi_{\text{old}} \| \pi) < \delta.$$

TRPO Policy Update

TRPO updates the policy by maximizing the objective while maintaining stability.

The key idea in TRPO is to ensure that the new policy π does not deviate too far from the old policy π_{old}. This is done using a KL-divergence constraint. The update step is determined by solving the following optimization problem:

$$\pi_{\text{new}} = \arg\max_{\pi} \mathbb{E}_{(x_t, a_t)\sim\rho_\pi} \left[r(x_t, a_t) \right]$$

subject to:

$$D_{\text{KL}}(\pi_{\text{old}} \| \pi) < \delta.$$

- The **KL-divergence** between the old and new policy is used to ensure gradual updates. - The **Fisher Information Matrix (FIM)** is used to approximate the natural gradient, which helps update the policy in a way that respects the KL constraint.

TRPO ensures stable learning by limiting how much the policy can change at each step.

TRPO step Size

TRPO computes the step size based on the Fisher Information Matrix (FIM) and the KL-divergence constraint:

The step size is calculated as:

$$\text{step size} = \sqrt{\frac{2\delta}{\mathbf{g}^T\mathbf{F}^{-1}\mathbf{g}}}\mathbf{F}^{-1}\mathbf{g},$$

where:

$$\mathbf{g} = \nabla_\theta J_{\text{TRPO}}(\pi) \quad \text{and} \quad \mathbf{F} = \mathbb{E}_{x_t, a_t \sim \rho_\pi}\left[\nabla_\theta \log \pi_\theta(a_t|x_t) \nabla_\theta \log \pi_\theta(a_t|x_t)^T\right].$$

This ensures that the policy update stays within the trust region defined by δ, maintaining stability and avoiding drastic changes in policy.

SAC Objective

SAC's objective is entropy-regularized, meaning it maximizes both the reward and the entropy of the policy:

$$J_{\text{SAC}}(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(x_t, a_t) \sim \rho_\pi} \left[r(x_t, a_t) + \alpha H(\pi(\cdot|x_t)) \right]$$

where the entropy term $H(\pi(\cdot|x_t))$ encourages exploration of the action space.

Policy and Value Function Updates

The Q-function $Q_\theta(x_t, a_t)$ and policy $\pi_\phi(a_t|x_t)$ are both learned through neural networks.

The Q-function update involves minimizing the Bellman residual:

$$J_Q(\theta) = \mathbb{E}_{(x_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(x_t, a_t) - \left(r(x_t, a_t) + \gamma\mathbb{E}_{x_{t+1}\sim p}[\,V_{\bar{\theta}}(x_{t+1})]\right)\right)^2\right].$$

The soft value function is defined as:

$$V_{\bar{\theta}}(x_t) = \mathbb{E}_{a_t\sim\pi}\left[Q_{\bar{\theta}}(x_t, a_t) - \alpha\log\pi(a_t|x_t)\right].$$

Policy Update

The policy parameters ϕ are updated by minimizing:

$$J_\pi(\phi) = \mathbb{E}_{x_t \sim \mathcal{D}} \mathbb{E}_{a_t \sim \pi_\phi} \left[\alpha \log \pi_\phi(a_t | x_t) - Q_\theta(x_t, a_t) \right].$$

Using a Gaussian distribution, the policy π_ϕ is parameterized as:

$$a_t = f_\phi(\epsilon_t; x_t)$$

where ϵ_t is sampled from a Gaussian distribution.

Gradient of the Policy Objective

The gradient of $J_\pi(\phi)$ is approximated by:

$$\begin{aligned}\nabla_\phi J_\pi(\phi) =& \nabla_\phi \alpha \log(\pi_\phi(a_t|x_t)) \\ &+ \nabla_{a_t}\left[\alpha \log(\pi_\phi(a_t|x_t)) - Q_\theta(x_t, a_t)\right] \nabla_\phi f_\phi(\epsilon_t; x_t).\end{aligned} \tag{1}$$

Auto Correction

Action Correction Overview

Why Action Correction is Necessary:

- High dimensionality and sparsity in the valid action space due to system constraints.
- Action correction ensures that actions conform to system requirements, improving performance and stability.

Rule 1: Service Latency Constraint

Condition: If $S_{(}^{OA}t) = 0$.

- If the number of computation cores $c_A(t)$ is less than the minimum value $\lceil \frac{I_A(t)}{w_A(t)\tau f_D} \rceil$, update $c_A(t)$ to the minimum.
- If $S_{(}^{OA}t) = 1$, set $c_A(t) = 0$.

Purpose: Ensures latency requirements are met while preventing unnecessary computation.

Rule 2: Proactive Pushing Condition

Condition: When $S_{If} + S_{Of} \geq 1$.

- Set $b_f = 0$.

Purpose: Prevents redundant data pushing when data for a task is already cached.

Rule 3: Minimizing Pushing Costs

Condition: When multiple tasks are un-pushed.

- Proactively transmit only one task, specifically the task with the largest $\overline{b}_f$.

Purpose: Minimizes costs by pushing only the most critical task to the mobile device.

Rule 4: Pushing Data Requirement

Condition: If $b_f = 1$.

- Set $\Delta s_{If} = 1$.

Purpose: Ensures that proactively pushed data is cached for future use.

Rule 5: Cache Capacity Constraint

Condition: If the sum of cache sizes exceeds capacity (Eq. (11)).

- Drop input or output cache based on ascending order of their corresponding $\bar{s}$ values until capacity is met.

Purpose: Prevents cache overflow by systematically removing less important data.

Rule 6: Adding Cache Space

Condition: If the sum of caches is below capacity (Eq. (11)).

- Add reactive input or output cache in descending order of continuous variables $\Delta\bar{s}_{IA}(t)$ and $\Delta\bar{s}_{OA}(t)$.

Purpose: Efficiently utilizes available cache space by adding the most relevant data.

Rule 7: Clipping Cache Actions

Condition: Applies to all actions.

- Clip Δs to fit within the minimum and maximum limits as per Eq. (9) and Eq. (10).

Purpose: Prevents invalid cache states by ensuring updates are within legal bounds.

Benefits of Action Correction

Valid Actions:

- Ensures the output action remains valid, preventing errors.

Efficiency:

- Compresses the action space, improving the efficiency of the training process.

Reduced Costs:

- Minimizes redundant actions, reducing overall system costs.

Adaptability:

- Dynamically adjusts actions based on system states and constraints.

Baseline

Baseline 1: MRU-LRU (Most-Recently-Used and Least-Recently-Used)

Description:

- Heuristic algorithm serving tasks reactively while caching the input data of the most recently used task proactively.

Cache Management:

- When the cache reaches capacity, it replaces the input data of the least recently used task.

Rationale:

- Output data is larger, so caching only input data reduces overall costs.

Computing Resources:

- Fixed at $0.75M$ computing cores.

Baseline 2: MFU-LFU (Most-Frequently-Used and Least-Frequently-Used)

Description:

- Focuses on frequency rather than recency, caching the most frequently used tasks while replacing the least frequently used ones.

Cache Management:

- Similar to MRU-LRU, prioritizes frequently accessed data for caching.

Baseline 3: DFNC (Dynamic-Computing-Frequency Reactive Service with No Cache)

Description:

- Mobile device reacts to task requests by downloading input data from the MEC server and computing the output without caching.

Rationale:

- Provides a baseline to compare caching versus non-caching performance.

Baseline 4: DFC (Dynamic-Computing-Frequency Reactive Service with Cache)

Description:

- Combines reactive service with the ability to cache both input and output data.

Cache Management:

- Uses a strategy that balances efficiency and capacity for resource management.

Baseline 5: Comparison with SOTA

Description:

- Uses a SAC-based implementation for proactive transmission techniques and incorporates dynamic-computing-frequency reactive service.

Comparison Baseline:

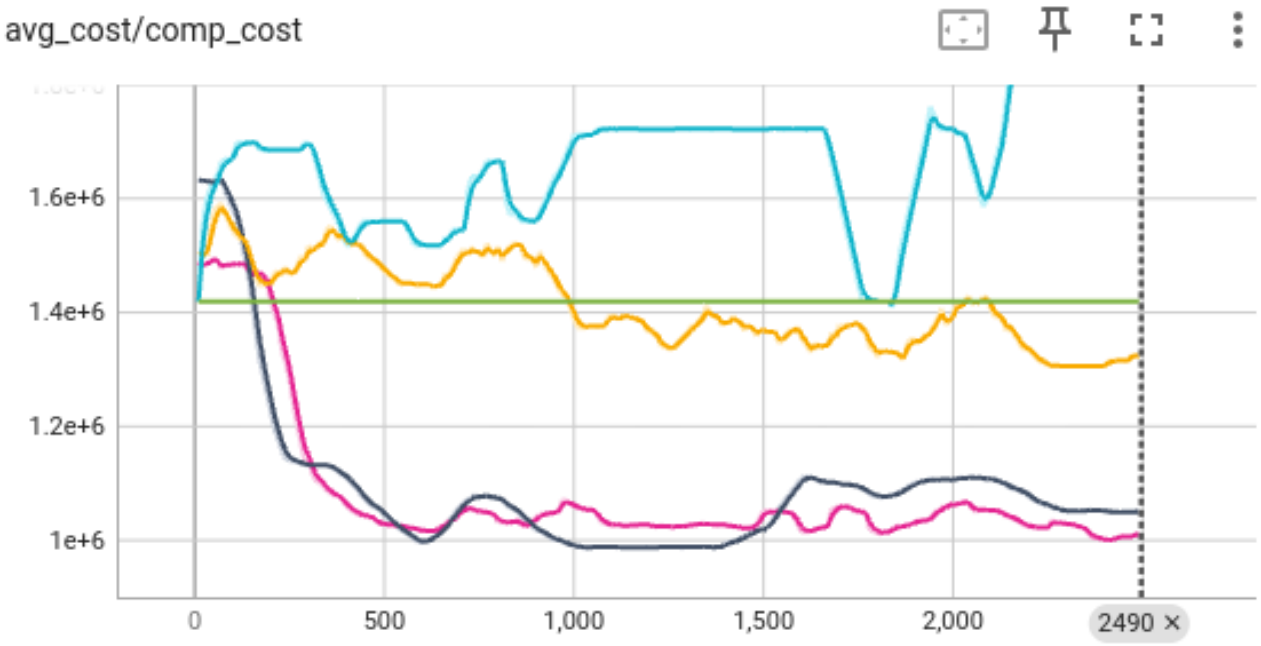
- Serves as a benchmark to assess the performance of the new TRPO-SAC-based approach.

Evaluation Metrics

To compare these algorithms, the following key metrics are used:

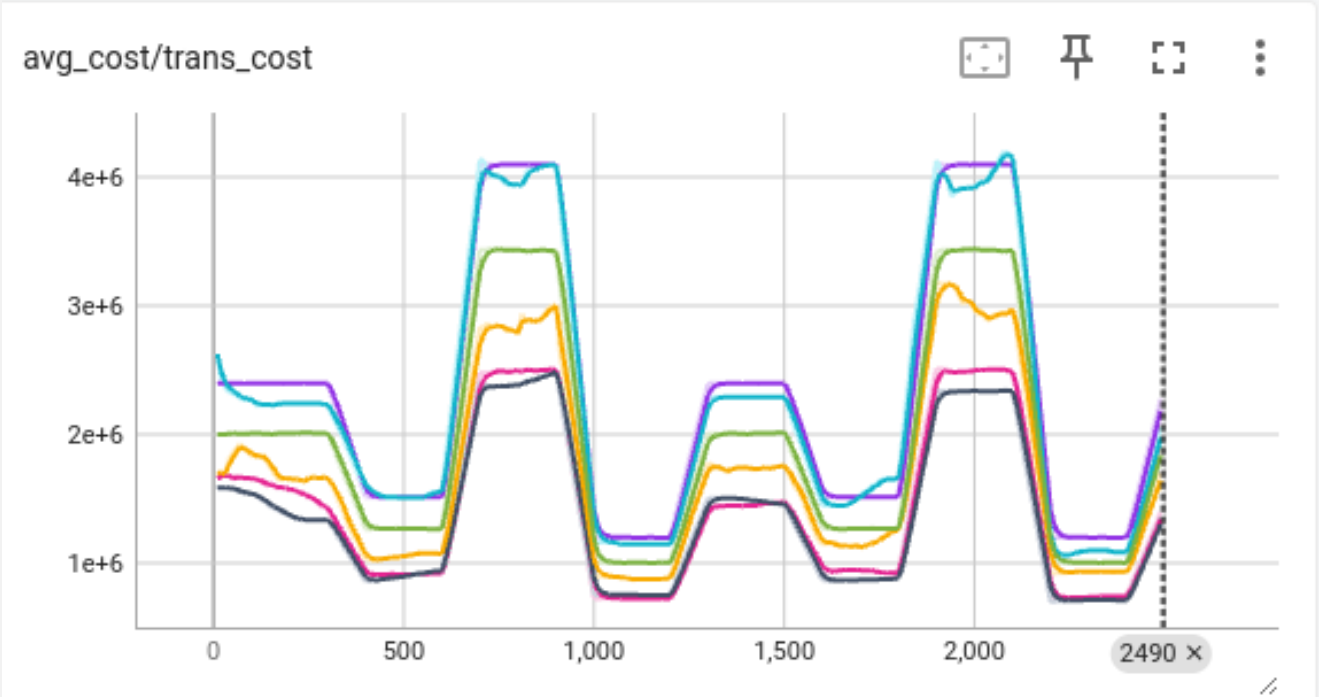
- **Total Cost:** The transmission and computation costs over the evaluation period.
- **Cache Hit Rate:** The proportion of requests served from the cache versus those fetched from the server.
- **Latency:** The time taken to process requests, important for real-time applications.
- **Resource Utilization:** Effectiveness of computing resource usage across algorithms.

Comparison of Average Cost and Computational Costs for TRPO-SAC and SAC



Run	Smoothed	Value	Step	Relative
DFC	1,324,281.3815	1,324,770.125	2,490	13.94 hr
DFNC	1,830,105.0647	1,830,243.75	2,490	14.5 hr
MFU-LRU	1,419,119.3893	1,419,128	2,490	15.06 hr
MRU-LRU	1,419,119.3893	1,419,128	2,490	14.81 hr
SAC_PTDFC	1,011,048.1959	1,012,914.5	2,490	14.16 hr
TRPO_SAC_PTDFC	1,050,326.3394	1,050,577.25	2,490	15.49 hr

Comparison of Average Cost and Transmission Costs for TRPO-SAC and SAC



Run	Smoothed	Value	Step	Relative
DFC	1,639,338.3488	1,702,153.125	2,490	13.94 hr
DFNC	1,975,592.2767	2,057,173	2,490	14.5 hr
MFU-LRU	2,185,683.591	2,275,681	2,490	15.06 hr
MRU-LRU	1,836,863.2729	1,911,952.125	2,490	14.81 hr
SAC_PTDFC	1,340,551.4347	1,394,826.75	2,490	14.16 hr
TRPO_SAC_PTDFC	1,302,818.7961	1,356,886.25	2,490	15.49 hr

Metric Analysis

- **Transmission Cost:** - *TRPO-SAC*: Shows a modest improvement over SAC in reducing transmission costs. By using the trust region constraint and entropy-based exploration of TRPO, the system achieves more efficient data transmission decisions, especially in high-load scenarios with many requests or complex tasks.
- **Computational Cost:** - *TRPO-SAC*: Yields a slight reduction in computational costs compared to SAC alone. TRPO's stable policy updates, combined with SAC's exploration strategies, help the system reach optimal solutions with fewer resources while maintaining high task accuracy.

Thank You