Online Policy Adaptation for Networked Systems using Rollout

Forough Shahab Samani†, Kim Hammar†, and Rolf Stadler†
† Dept. of Computer Science, KTH Royal Institute of Technology, Sweden
Email: {foro, kimham, stadler}@kth.se

NOMS 2024-2024 IEEE Network Operations and Management Symposium | 979-8-3503-2793-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/NOMS59830.2024.10575707

***Abstract*—Dynamic resource allocation in networked systems is needed to continuously achieve end-to-end management objectives. Recent research has shown that reinforcement learning can achieve near-optimal resource allocation policies for realistic system configurations. However, most current solutions require expensive retraining when changes in the system occur. We address this problem and introduce an efficient method to adapt a given base policy to system changes, e.g., to a change in the service offering. In our approach, we adapt a base control policy using a rollout mechanism, which transforms the base policy into an improved rollout policy. We perform extensive evaluations on a testbed where we run applications on a service mesh based on the Istio and Kubernetes platforms. The experiments provide insights into the performance of different rollout algorithms. We find that our approach produces policies that are equally effective as those obtained by offline retraining. On our testbed, effective policy adaptation takes seconds when using rollout, compared to minutes or hours when using retraining. Our work demonstrates that rollout, which has been applied successfully in other domains, is an effective approach for policy adaptation in networked systems.**

***Index Terms*—Performance management, reinforcement learning, service mesh, policy adaptation, rollout, Istio, Kubernetes**

I. Introduction

To continuously meet performance objectives for a service, such as a bound on end-to-end delay or maximizing throughput for service requests, the management system must dynamically re-allocate the resources of the infrastructure. Such control actions can be taken on the physical layer, the virtualization layer, or the service layer. They include horizontal and vertical scaling of compute resources, function placement, as well as request routing and request dropping.

A promising approach to automatically find effective control policies is reinforcement learning [1]. Following this approach, the problem is modeled as a Markov decision problem, and policies are learned through simulation. Though encouraging results have been obtained (e.g., [2]–[6]), key challenges remain. Chief among them is to efficiently adapt control policies to changes in the target system. Such adaptation is needed when changes occur in operational systems: load patterns shift, the available bandwidth fluctuates, components fail or are updated, etc. Most existing reinforcement learning methods do not allow for rapid policy adaptation but require slow and expensive retraining, which limits their practical use [2]–[6].

Policy adaptation in networked systems is an active area of research. Methods for policy adaptation proposed in prior work include meta-learning [7], transfer learning [8], offline dynamic programming [9], offline reinforcement learning [2], [10], and online model-free reinforcement learning [11]. Each of these methods has practical disadvantages: offline methods require expensive re-optimization whenever the target system changes, which does not scale to practical scenarios; meta-learning and transfer-learning methods are scalable but generally have worse performance than methods that re-optimize; and the model-free reinforcement learning methods are slow to adapt to system changes as they do not make use of a system model.

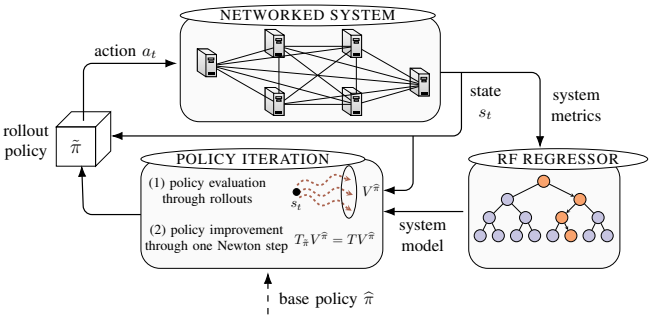


Fig. 1. Our approach for policy adaptation in networked systems; during each control cycle, the system model is estimated from system metrics using supervised learning; a given base policy $\hat{\pi}$ is adapted for the current state and the current system model through one step of policy iteration, which we call rollout; the output of this step is an improved rollout policy $\tilde{\pi}$ which is used to select the next control action.

In this paper, we advocate a new approach for policy adaptation in networked systems, without the above drawbacks. It is based on *rollout* as formulated by D. Bertsekas [12]. We combine a given *base policy* with one step of *policy iteration*. The iteration step makes use of the system model, which we estimate from system measurements (see Fig. 1). The base policy can be chosen freely but must satisfy certain consistency properties [12]. It can be obtained through offline reinforcement learning or dynamic programming. It can also be based on heuristics or be designed by a domain expert. At each time-step during online execution, the system model is used to estimate the value of the base policy and then one step of policy iteration is executed with the base policy as the start point [13, Eqs. 6.4.1-22]. This procedure transforms the base policy into a *rollout policy*, which is effective for the current

system state (see Fig. 1).

While rollout has been applied in other contexts including board games [14], [15] and video games [16], we find that this work is the first systematic study of rollout for policy adaptation in networked systems.

To study and evaluate our approach, we apply it to adapting a control policy on a service mesh based on the Kubernetes [17] and Istio [18] platforms. From the evaluation we learn that (*i*) compared with offline re-optimization, rollout provides a cost-effective way to rapidly adapt a given base policy to changes in the target system; (*ii*) having a reasonably accurate system model through estimation from system measurements, rollout can yield near-optimal performance; (*iii*) for the system changes we investigate, rollout with a lookahead horizon of length 1 is sufficient (increasing the horizon does not practically improve performance); (*iv*) multi-agent rollout algorithms are significantly faster than single-agent rollout algorithms for the multi-dimensional control problems we study in the evaluation scenarios; and (*v*) selecting actions uniformly at random leads to a policy that is not sequentially consistent, which causes the rollout algorithms to have poor performance.

This work fits in the broad context of policy-based management, a concept that was developed in the late 1990s and focused primarily on rules that guide the behavior of network elements to facilitate and automate their administration. Note that we use the term policy in this paper with the precise meaning of reinforcement learning, not with the broader meaning of policy-based management. Our work further lies in the scope of Intent-based networking, a concept proposed in the mid 2010s, which builds on the foundations of policy-based management and deals with network objectives from a business or operational perspective.

The main contributions of this paper are:

- We develop an approach for efficient policy adaptation in networked systems based on the rollout method. To our knowledge, we are the first to study and apply rollout in this context. Our implementation is available at [19].
- We describe the design space of rollout for policy adaptation in networked systems and provide a guide for selecting a specific rollout algorithm.
- We evaluate our policy adaptation approach using a service mesh running on a lab testbed and show that it outperforms state-of-the-art offline reinforcement learning methods for the policy adaptation task.

II. Theoretical Background

A. Discrete-Time Dynamical Systems

We model a networked system as a discrete-time dynamical system which is defined by the system model

$$s_{t+1} = f(s_t, a_t, w_t, t), \tag{1}$$

where $s_t \in \mathcal{S}$ is the system state at time t, $a_t \in \mathcal{A}$ is the action, and $w_t \in \mathcal{W}$ is a random disturbance which realizes the random variable W_t. Actions are decided by a control policy $\pi : \mathcal{S} \to \mathcal{A}$ and the disturbances are sampled from a probability distribution $P(\cdot \mid s_t, a_t)$ [12].

Each state transition is associated with a reward $r(s_t, a_t) \in \mathbb{R}$ and a policy is said to be optimal if it maximizes the expected cumulative discounted reward

$$\mathbb{E}_{(W_t)_{t=1,2,\ldots,N}} \left[\sum_{t=1}^{N} \gamma^{t-1} r(s_t, a_t) \right], \tag{2}$$

where $N > 1$ is the time horizon and $\gamma \in [0, 1]$ is a discount factor.

The Bellman equation [20, Eq. 1] relates a policy π to the value function V^π, which is defined as

$$V^\pi(s_t) \triangleq \mathbb{E}_W \left[r(s_t, \pi(s_t)) + \gamma V^\pi(f(s_t, \pi(s_t), w_t)) \right], \tag{3}$$

for all $s_t \in \mathcal{S}$.

As any policy that maximizes (2) also satisfies

$$\pi^\star(s_t) \in \arg\max_{a_t \in \mathcal{A}} \mathbb{E}_W \left[r(s_t, a_t) + \gamma V^{\pi^\star}(s_{t+1}) \mid s_t, a_t \right],$$

the Bellman equation (3) effectively provides an alternative optimality condition (sufficient and necessary) to (2). (Remarks: in this paper we restrict ourselves to finite state and action spaces which ensures that a maximizer of (2) always exists, which is why we write $\max$ instead of $\sup$; if the system model f or the reward function r are time-dependent, or if the time horizon N is finite, then $\pi^\star$ and $V^\star$ may be non-stationary, denoted as $\pi_t^\star$ and $V_t^\star$.)

B. Dynamic Programming

Dynamic programming algorithms, e.g., value iteration and policy iteration, use the Bellman equation (3) to obtain an optimal policy through successive approximations of $V^{\pi^\star}$.

Let T and T_π denote Bellman operators defined as

$$(TV)(s) \triangleq \max_{a \in \mathcal{A}} \mathbb{E}_W \left[r(s, a) + \gamma V(f(s, a, W)) \right] \tag{4}$$

$$(T_\pi V)(s) \triangleq \mathbb{E}_W \left[r(s, \pi(s)) + \gamma V(f(s, \pi(s), W)) \right] \tag{5}$$

for all $s \in \mathcal{S}$. Using these operators, value iteration can be defined through the recursion $V_{k+1} = TV_k$. Similarly, policy iteration can be defined through the following two equations

$$V^{\pi_k} = T_\pi V^{\pi_k} \tag{6}$$

$$\pi_{k+1}(s) \in \arg\max_{a \in \mathcal{A}} \mathbb{E}_W \left[r(s, a) + \gamma V^{\pi_k}(f(s, a, W)) \right] \tag{7}$$

for all $s \in \mathcal{S}$ and $k = 1, 2, \ldots$.

If $|\mathcal{S}| < \infty$, $|\mathcal{A}| < \infty$ and $\gamma \in [0, 1)$ or $N < \infty$, then the value functions produced by value iteration satisfy $\lim_{k\to\infty} V_k = V^{\pi^\star}$ [13, Thm. 6.3.1] and the policies produced by (6)–(7) satisfy $\lim_{k\to\infty} \pi_k = \pi^\star$ [13, Thm. 6.4.2].

C. Relation Between Policy Iteration and Newton's Method

Policy iteration can be viewed as a vector space version of Newton's method for solving the Bellman equation for the optimal policy $\pi^\star$ (3). To see this, note that the value functions produced by policy iteration satisfy $T_{\pi_k} V^{\pi_{k-1}} = TV^{\pi_{k-1}}$, where T_{π_k} is linear and T is non-linear. Hence, policy iteration

effectively linearizes the Bellman operator around $V^{\pi_{k-1}}$ and then solves for the fixed point, analogous to Newton's method. For further details on the relation between policy iteration and Newton's method see [13], [21].

D. The Rollout Method for Online Policy Improvement

The rollout method with a base policy $\widehat{\pi}$ can be viewed as a single step of policy iteration applied to a specific state [12]. That is, given a state s_t, we first compute $V^{\widehat{\pi}}(s_t)$ by rolling out the system model (i.e., running simulations of the model) and taking actions prescribed by the base policy $\widehat{\pi}$ (6). We then solve (7) to obtain an improved policy $\tilde{\pi}$, which we call the *rollout policy*. Lastly, we use the rollout policy to select the next action to apply to the system, i.e., $a_t = \tilde{\pi}(s_t)$. After applying the action a_t, the system transitions to a new state s_{t+1} and the same process is repeated.

The rollout framework includes several enhancements to the general method described above. In particular:

Approximation in value space. In practice, it is often computationally intractable to compute $V^{\widehat{\pi}}(s)$ as it requires enumerating the entire state space (6). For this reason, it is common to perform the policy iteration step with an approximation of $V^{\widehat{\pi}}(s)$. Typical ways to approximate $V^{\widehat{\pi}}(s)$ are offline neural network training and online Monte-Carlo methods.

Multi-step lookahead. The one-step lookahead maximization used in the policy iteration step (7) can be exchanged with an l-step lookahead maximization, where $l \geq 1$. The effect of using $l > 1$-step lookahead maximization is that the starting point of the Newton step is moved closer to the optimal value function through $l - 1$ value iterations [12].

Multi-agent rollout. When a control action is of the form $a = (a_1, \ldots, a_m)$, the computational complexity of one-step lookahead maximization grows exponentially with the number of dimensions m. To circumvent this challenge, *multi-agent rollout* replaces the maximization over the joint action space with m successive one-dimensional maximizations (i.e., m agents). These successive maximizations define a modified but equivalent control problem. In this modified control problem, each agent only has to consider one dimension of the action space, which means that the number of lookahead-values that have to be computed (i.e., the number of evaluations of (7)) is reduced from $O(n^m)$ to $O(nm)$ (given that $a_i \in \{1, \ldots, n\}$ for all i) [12]. The drawback of multi-agent rollout is that it introduces additional artificial states that represent the sequential selection of actions. While this means that the size of the state space is increased, it is a minor drawback in practice as it does not increase the computational requirements of the rollout algorithm.

Asynchronous multi-agent rollout. The multi-agent rollout algorithm can be executed asynchronously to reduce the computational time. In the asynchronous version, the controls are computed in parallel rather than sequentially. This algorithm may work well for some problems but does not have the same theoretical guarantees as the standard rollout and multi-agent rollout algorithms [12].

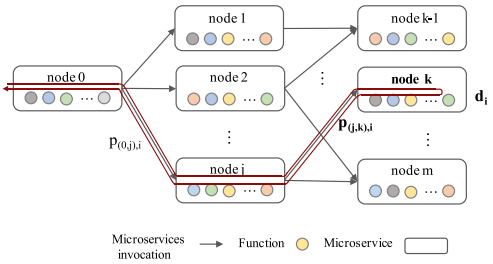


Fig. 2. Example of a service mesh. The nodes are microservices and the links represent microservice invocations. d_i is the end-to-end delay of processing a request of service i; $p_{(j,k),i}$ denotes the routing probability of microservice j invoking microservice k to process a request for service i.

III. System Model and Problem Formulation

We consider a service mesh that provides a set of services to a client population. The services are built from microservice components and can be understood as contiguous subgraphs on a directed graph that represents the service mesh (see Fig. 2). In the following, we formally model the service mesh as a discrete-time dynamical system with graph structure and formulate the problem of meeting management objectives as an optimal control problem.

A. Modeling the Service Mesh

Denote with $\mathcal{G} \triangleq (\mathcal{V}, \mathcal{E})$ the directed graph of the service mesh where $\mathcal{V}$ represents the set of nodes and $\mathcal{E}$ represents the set of edges. Each node $k \in \mathcal{V}$ offers a specific microservice and a directed edge $(j, k) \in \mathcal{E}$ represents the invocation of the microservice of node k by node j.

The nodes of the service mesh collectively provide a set of services $\mathscr{S} = \{S_1, \ldots, S_{|\mathscr{S}|}\}$, which we model as subgraphs of $\mathcal{G}$. Associated with each node $k \in \mathcal{V}$ is a CPU allocation $c_k \in \mathbb{R}^+$ and associated with each service S_i are performance metrics, such as end-to-end response time $d_i \in \mathbb{R}^+$ and service utility $u_i \in \mathbb{R}^+$.

The services are consumed by clients that generate service requests, each of which can be considered a traversal of the subgraph of the requested service. A service request can be fulfilled in several ways, corresponding to different graph traversals. The specific traversal path for a request is decided by a set of routing probabilities $p_{(j,k),i} \in [0, 1]$, where each probability is associated with an edge $(j, k) \in \mathcal{E}$. (For each node $j \in \mathcal{V}$ with at least one edge $(j, k) \in \mathcal{E}$, the routing probabilities satisfy $\sum_{(j,k)\in\mathcal{E}} p_{(j,k),i} = 1 \quad \forall S_i \in \mathscr{S}$).

The stream of service requests generated by the clients induces an offered load $l_i \in \mathbb{R}^+$ (in requests per second) for each service $S_i \in \mathscr{S}$, which we assume evolves according to a load function $l_{i,t} = \lambda_i(t, w_t)$. Here w_t is a random disturbance which realizes the random variable W_t. To balance the load and bound the response time d_i of a service S_i, the service mesh may block a fraction $b_i \in [0, 1]$ of the service requests for S_i, which results in the carried load $l_i^{(c)} \triangleq (1 - b_i) l_i$.

B. Resource Allocation to Meet Management Objectives

We consider the problem of dynamically controlling the routing probabilities $p_{(j,k),i}$, the blocking rates b_i, and the CPU allocations c_j to meet a given management objective. For the results reported in this paper, we focus on the management objective of minimizing the number of CPU cores c_j while bounding the response time $d_i < O_i$ of each service $S_i \in \mathscr{S}$. This objective can formally be expressed as

$$\text{minimize} \sum_j c_j \quad \text{subject to } d_i < O_i \text{ for all } S_i \in \mathscr{S}. \tag{8}$$

We model the problem of meeting the above objective as the problem of controlling a discrete-time dynamical system that evolves in time-steps $t = 1, 2, \ldots$. In this model, the management objective is encoded with the following reward function

$$r(s_t, a_t) \triangleq \begin{cases} 1 - \dfrac{\sum_{j=1}^{|\mathcal{V}|} c_j}{|\mathcal{V}| c_{max} \Delta_c} & \text{if } d_i \leq O_i, \forall S_i \in \mathscr{S} \\ \displaystyle\sum_{S_i \in \mathscr{S}} (O_i - d_i) & \text{if } \exists S_i \in \mathscr{S}, d_i > O_i. \end{cases} \tag{9}$$

This function gives increasing rewards as the delays are reduced to the specified thresholds, as defined by the term $\sum_i (O_i - d_i)$. When the delay thresholds are met, the reward increases when the number of allocated CPUs is decreased. (Δ_c and c_{max} denote the "discretization step for CPU allocation" and "maximum number of CPU units", respectively.) We obtain the delay d_i for each service $S_i \in \mathscr{S}$ through a delay function α_i that we estimate from system measurements (14).

The system state s_t is defined by the service loads, the routing probabilities, the blocking rates, and the CPU allocations:

$$s_t \triangleq (l_{i,t}, p_{(j,k),i,t}, b_{i,t}, c_{j,t})_{i \in \mathscr{S}, (j,k) \in \mathcal{E}, j \in \mathcal{V}}. \tag{10}$$

The evolution of the state depends on the load as well as the control actions, which we model as

$$s_{t+1} \triangleq f(s_t, a_t, w_t, t) \qquad t = 1, 2, \ldots, \tag{11}$$

where f is the system model (1), $w_t \in \mathcal{W}$ is a random disturbance, and a_t is the control action at time t, which is defined as

$$a_t \triangleq ((a^{(p)}_{(j,k),i,t}, a^{(b)}_{i,t}, a^{(c)}_{j,t}))_{i \in \mathscr{S}, (j,k) \in \mathcal{E}, k, j \in \mathcal{V}}, \tag{12}$$

where $a^{(p)}_{(j,k),i,t} \in \{-\Delta_p, 0, \Delta_p\}$ indicates the change in routing probability for edge (j,k) and service S_i, $a^{(b)}_{i,t} \in \{-\Delta_b, 0, \Delta_b\}$ indicates the change in blocking rate for service S_i, and $a^{(c)}_{j,t} \in \{-\Delta_c, 0, \Delta_c\}$ indicates the change in allocated CPU cores for node j.

Given (12), the system model (11) can be stated more explicitly as

$$w_{t+1} \sim P(\cdot \mid s_t, a_t) \tag{13a}$$
$$l_{i,t+1} = \lambda_i(t+1, w_{t+1}) \qquad i \in \mathscr{S} \tag{13b}$$
$$p_{(j,k),i,t+1} = p_{(j,k),i,t} + a^{(p)}_{(j,k),i,t} \qquad i \in \mathscr{S}, (j,k) \in \mathcal{E} \tag{13c}$$
$$b_{i,t+1} = b_{i,t} + a^{(p)}_{i,t} \qquad i \in \mathscr{S} \tag{13d}$$
$$c_{j,t+1} = c_{j,t} + a^{(c)}_{j,t} \qquad j \in \mathcal{V}, \tag{13e}$$

where $t = 1, 2, \ldots$ and $w_{t+1} \sim P(\cdot \mid s_t, a_t)$ denotes that w_{t+1} is sampled from P.

Based on (13), we compute the delay as

$$d_{i,t+1} = \alpha_i(s_t, a_t, w_{t+1}) \qquad i \in \mathscr{S}, \tag{14}$$

where α_i is a delay estimator for service i.

We restrict the set of allowable actions at time t to be $\mathcal{A}(s_t)$, where $\mathcal{A}(s_t)$ is the set of actions that keeps the state of the system within its *operating region* [22]. We say that a system state s_t is within the operating region $O_{p,t}$ if the variances of the response times $(d_{i,t})_{i \in \mathscr{S}}$ are small. In this work, we consider a variance to be small if it is smaller than 50% of the mean. Formally,

$$O_{p,t} = \left\{ s_t \mid s_t \in \mathcal{S},\ \sigma_{d_{i,t}, s_t} \leq \frac{d_{i,t}}{2}\ \forall i \in \mathscr{S} \right\}, \tag{15}$$

where $\sigma_{d_{i,t},s}$ is the standard deviation of $d_{i,t}$ in state s.

As the reward function (9) encodes the management objective (8), our goal is to find a policy $\pi_t : \mathcal{S} \to \mathcal{A}$ that maximizes the expected cumulative discounted reward. The control problem of finding such a policy can be stated as

$$\underset{\pi_t \in \Pi}{\text{maximize}} \qquad \mathbb{E}_{(W_t)_{t=1,2,\ldots}} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \right] \tag{16a}$$
$$\text{subject to} \qquad s_{t+1} = f(s_t, a_t, w_t, t) \in \mathcal{S} \tag{16b}$$
$$a_{t+1} = \pi_t(s_t) \in \mathcal{A}(s_t) \tag{16c}$$
$$w_{t+1} \sim P(\cdot \mid s_t, a_t) \in \mathcal{W}, \tag{16d}$$

where Π is the policy space, $\gamma \in (0,1)$ is a discount factor; (16b) captures the dynamics; (16c) captures the actions and the operating region constraint; and (16d) captures the disturbances.

Solving (16) yields an optimal control policy $\pi^\star_t$. From standard results in Markov decision theory we know that (16) is well-defined in the following sense.

Proposition 1. *There exists an optimal non-stationary and deterministic policy $\pi^\star_t : \mathcal{S} \to \mathcal{A}$ that solves (16). Further, if the load function $\lambda_i(t, w_t)$ for each service $S_i \in \mathscr{S}$ is constant, there exists an optimal stationary deterministic policy $\pi^\star : \mathcal{S} \to \mathcal{A}$.*

Proof. From (10)–(12) we know that solving (16) is equivalent to solving a finite discounted Markov decision process (MDP) with bounded rewards. If the load is constant, the dynamics of the MDP are stationary, otherwise they are time-dependent. The statement then follows from standard results in Markov decision theory, see e.g., Proposition 4.4.3 and Theorem 6.2.10 in [13]. For the sake of brevity we do not restate the proof. □

IV. Adaptive Control through Rollout

Our approach for policy adaptation in networked systems involves two parts: (*i*) construction of a base policy; and (*ii*), online adaptation of the base policy through rollout.

A. Base policies

We consider three base policies: $\widehat{\pi}_{\text{RANDOM}}$, $\widehat{\pi}_{\text{GREEDY}}$, and $\widehat{\pi}_{\text{PPO}}$. The base policy $\widehat{\pi}_{\text{RANDOM}}$ selects actions uniformly at random, i.e., $\widehat{\pi}_{\text{RANDOM}}(a \mid s) = \frac{1}{\mathcal{A}(s)}$ for all $a \in \mathcal{A}(s)$. Similarly, $\widehat{\pi}_{\text{GREEDY}}$ is a heuristic base policy that greedily selects the action that has the highest immediate reward in each state, i.e., $\widehat{\pi}_{\text{GREEDY}}(s) \in \arg\max_{a \in \mathcal{A}(s)} r(s,a) - 10^{-3}\|a\|$, where $10^{-3}\|a\|$ is a regularization term. Lastly, the deterministic base policy $\widehat{\pi}_{\text{PPO}}$ is obtained through offline reinforcement learning with the PPO algorithm [23, Alg. 1].

B. Online Adaptation of a Base Policy

Our approach for adapting a given base policy $\widehat{\pi}$ to changes in the target system is based on the rollout method described in §II and involves repeating five steps: (*i*) measure the system state; (*ii*) estimate $V_t^{\widehat{\pi}}$ (3); (In this work, we estimate $V^{\widehat{\pi}_{\text{RANDOM}}}$ and $V^{\widehat{\pi}_{\text{GREEDY}}}$ through online Monte-Carlo simulations and we estimate $V^{\widehat{\pi}_{\text{PPO}}}$ through offline neural network training.) (*iii*) perform one step of policy iteration (6)–(7) to transform $\widehat{\pi}$ into a new rollout policy $\tilde{\pi}$; (*iv*) use $\tilde{\pi}$ to select the control to apply to the system; and (*v*) update the system model f (16b) and the delay functions $(\alpha_i)_{i\in\mathscr{S}}$ (14) (the delay functions are required to compute the reward). The pseudocode of our rollout algorithm that implements these five steps is listed in Alg. 1. (Note that Alg. 1 implements the multi-agent rollout method, which reduces to the single-agent rollout method when the action space is of dimension 1.)

C. Theoretical Guarantees

The principal aim of the policy adaptation approach described above is *policy improvement*, i.e., that the adapted rollout policy is improved with respect to the base policy. In this section, we briefly discuss conditions under which this improvement is guaranteed to hold. For a more thorough treatment of the theory we refer the reader to Bertsekas [12].

Any base policy that is *sequentially consistent* has the policy improvement property [12, Prop. 2.3.1]. A policy is said to be sequentially consistent if when it generates the sequence $(s_k, a_k, s_{k+1}, a_{k+1}, \ldots)$ starting from state s_k it also generates the sequence $(s_{k+1}, a_{k+1}, \ldots)$ from state s_{k+1} [12, Def. 2.3.1]. Clearly, any Markovian deterministic policy is sequentially consistent whereas a stochastic policy may not be sequentially consistent. We thus have the following basic result.

Proposition 2. *The base policies $\widehat{\pi}_{\text{GREEDY}}$ and $\widehat{\pi}_{\text{PPO}}$ are sequentially consistent. The base policy $\widehat{\pi}_{\text{RANDOM}}$ is not sequentially consistent.*

Proof. The proof is immediate from the definitions of $\widehat{\pi}_{\text{GREEDY}}$, $\widehat{\pi}_{\text{PPO}}$, and $\widehat{\pi}_{\text{RANDOM}}$. In particular, $\widehat{\pi}_{\text{GREEDY}}$ and $\widehat{\pi}_{\text{PPO}}$ are deterministic Markovian policies whereas $\widehat{\pi}_{\text{RANDOM}}$ is a stochastic policy. □

Corollary 1. *The rollout policies $\tilde{\pi}_{\text{GREEDY}}$ and $\tilde{\pi}_{\text{PPO}}$ adapted from $\widehat{\pi}_{\text{GREEDY}}$ and $\widehat{\pi}_{\text{PPO}}$ through Alg. 1 satisfy $V^{\tilde{\pi}_{\text{GREEDY}}}(s) \geq V^{\widehat{\pi}_{\text{GREEDY}}}(s)$ and $V^{\tilde{\pi}_{\text{PPO}}}(s) \geq V^{\widehat{\pi}_{\text{PPO}}}(s)$ for all $s \in \mathcal{S}$ with respect to the system model f and the reward function r.*

Proof. The statement is a direct consequence of Prop. 2 and [12, Prop. 2.3.1]. In the interest of space we do not restate the proof here. □

Algorithm 1: Online policy adaptation.

Input : Base policy $\widehat{\pi}$, system model f, reward function r, delay functions $(\alpha_i)_{i\in\mathscr{S}}$, lookahead horizon l, discount factor γ, set of states $\mathcal{S}$, set of actions $\mathcal{A}$

Output: Control actions $a_1, a_2, \ldots$ prescribed by $\tilde{\pi}$

Algorithm
OnlinePolicyAdaptation$(\widehat{\pi}, f, l, \gamma, \mathcal{S}, \mathcal{A}, r, (\alpha_i)_{i\in\mathscr{S}})$
  Initialize $\mathbf{D}_0 \triangleq \emptyset$, $a_0 = \perp$, $s_0 = \perp$
  for $t = 1, 2, \ldots$ **do**
    Measure the system state s_t
    Update dataset $\mathbf{D}_t \leftarrow \mathbf{D}_{t-1} \cup \{(s_{t-1}, a_{t-1}, s_t)\}$
    $f, (\alpha_i)_{i\in\mathscr{S}} \leftarrow \text{SystemModel}(f, (\alpha_i)_{i\in\mathscr{S}}), \mathbf{D}_t)$
    $\tilde{\pi}_t \leftarrow \text{Rollout}(\widehat{\pi}, s_t, f, \gamma, \mathcal{S}, \mathcal{A}, r, (\alpha_i)_{i\in\mathscr{S}})$
    Apply control $a_t = \tilde{\pi}_t(s_t)$
  end
Procedure Rollout$(\widehat{\pi}, s_t, f, \gamma, \mathcal{S}, \mathcal{A}, r, (\alpha_i)_{i\in\mathscr{S}})$
  $N \leftarrow \dim(\mathcal{A})$
  for agent $i \in N$ **do**
    in parallel for $a_{i,t} \in \mathcal{A}_i(s_t)$ **do**
      $a_t \leftarrow (\tilde{a}_{1,t}, \ldots, a_{i,t}, \widehat{\pi}(s_t)_{i+1}, \ldots, \widehat{\pi}(s_t)_N)$
      $R_{a_{i,t}} \leftarrow r(s_t, a_t) + \arg\max_{\tilde{\pi}_{t+1}, \ldots, \tilde{\pi}_{t+l-1}}$
      $\mathbb{E}\left[\sum_{k=t+1}^{k+l-1} r(s_k, \tilde{\pi}_k(s_k)) + \gamma^l V_t^{\widehat{\pi}}(s_{k+l})\right]$
      where $V_t^{\widehat{\pi}}(s) = \text{StateValue}(s, \widehat{\pi}, f)$
    $\tilde{a}_{i,t} \leftarrow \arg\max_{a_{i,t}} R_{a_{i,t}}$
  end
  $\tilde{\pi}_t(s_t) \leftarrow (\tilde{a}_{1,t}, \ldots, \tilde{a}_{N,t})$
  $\tilde{\pi}_t(s) \leftarrow \widehat{\pi}(s) \quad \forall s \in \mathcal{S} \setminus \{s_t\}$
  return $\tilde{\pi}_t$
Procedure StateValue$(s, \widehat{\pi}, f)$
  Estimate $V^{\widehat{\pi}}(s)$ using f
  return $V^{\widehat{\pi}}(s)$
Procedure SystemModel$(f, (\alpha_i)_{i\in\mathscr{S}}), \mathbf{D}_t)$
  Update f and $(\alpha_i)_{i\in\mathscr{S}}$ based on $\mathbf{D}_t$
  (In this paper we use random forest regression.)
  return f

V. Target System and Evaluation Scenarios

In this section, we describe our setup to evaluate the rollout approach described above. We first detail the target system which is used for evaluation (§V-A). Then we describe our method for learning the system model based on measurements from the target system (§V-B). Lastly, we define two evaluation scenarios to assess the performance of our approach (§V-C).

A. Target System: The KTH Testbed

The target system in this work is a service mesh consisting of two services provided by five microservice components: an information service (S_1) and a compute service (S_2). The service mesh is realized using Kubernetes [17] and Istio [18], and runs on top of a server cluster with 10 servers connected through a Gigabit Ethernet switch. (See [4] for more details about the cluster and the services.)

B. Online Learning of the System Model

Using the measurements collected from the target system, we estimate the delay functions $(\alpha_i)_{i\in\mathscr{S}}$ (14) through random forest regressors that are re-trained periodically on data collected from the target system [10], [24], [25]. For the evaluation results reported in this paper, we re-fit the regressors every time-step. (A time-step on the testbed is defined to be 30 seconds.) To evaluate the performance of the fitted regressors, we use the Normalized Mean Absolute Error (NMAE) metric defined as $\frac{1}{\overline{y}}\left(\frac{1}{m}\sum_{i=1}^{m}|y_i-\widehat{y_i}|\right)$, where $\widehat{y_i}$ is the ith predicted response time, y_i is the ith measured response time, $\overline{y}$ is the average of the measured response times, and m is the number of samples.

C. Evaluation Scenarios

Following the problem formulation in §III, we define two scenarios to evaluate our approach for policy adaptation. In both scenarios, the goal is to adapt a given base policy to meet the management objective of minimizing CPU usage while meeting delay bounds on service requests under varying system configurations (see (8)–(9)). To meet this objective, we consider two control functions: routing $a^{(p)}_{(j,k),i,t}$ and scaling $a^{(c)}_{j,t}$, with the control steps $\Delta_p = 0.2$ and $\Delta_c = 1$ (12).

Our aim in evaluating the scenarios is to assess the following: a) the performance of the rollout method; b) the computational time required for policy adaptation; and c) the trade-offs between different rollout algorithms.

Scenario 1: Rollout with different base policies. In this scenario, we investigate the impact of different base policies on the rollout. In addition, we study the effect of different lookahead steps, as well as how multi-agent rollout compares to the single agent rollout. We evaluate rollout with the three base policies defined in §IV (i.e., $\widehat{\pi}_{\text{PPO}}$, $\widehat{\pi}_{\text{RANDOM}}$, and $\widehat{\pi}_{\text{GREEDY}}$).

The scenario includes two services on the service mesh: information service S_1 and compute service S_2. The service loads are kept constant with $l_1 = 4$ req/sec and $l_2 = 15$ req/sec.

Scenario 2: Rollout with changing services. In this scenario, we investigate the performance of the rollout method on the task of adapting a given base policy to a change in the number of services running on the target system.

The scenario is divided into two time intervals. In the first interval, which begins at t_0, we run the compute service S_2 and load it with a constant load of $l_2 = 1$ req/sec. In the second time interval, which begins at t_1, we start the information service S_1 in the background on processing node 2 and load it with $l_1 = 20$ req/sec.

As opposed to the first scenario, where we evaluate rollout with different base policies, in this scenario we focus on adapting the PPO base policy $\widehat{\pi}_{\text{PPO}}$ and compare the performance of the rollout method with the performance of offline retraining, which is the most popular method for policy adaptation among previous works (see §VII for a review of related work).

VI. Experimental Evaluation

This section describes the evaluation results for the two scenarios defined above. We evaluate the adapted policies both in simulation and on the target system described in §V-A. The source code of our implementation is available at [19]. Hyperparameters are listed in [4].

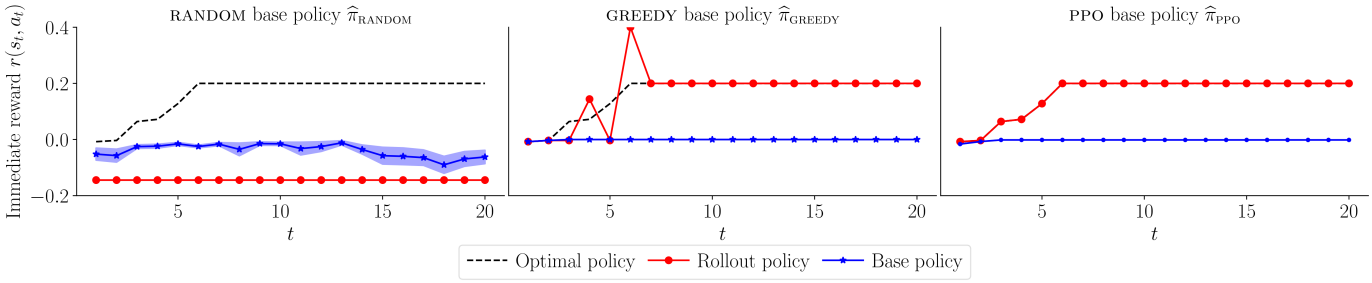
A. Evaluation Results for Scenario 1

As described in §V-C, for scenario 1 we run services S_1 and S_2 under constant load. To illustrate the different behavior of these services, we show the empirical distributions of the response times (histogram) as well as the predicted distributions (solid line) in Figure 3b. We observe that the distribution of service S_2 has a long tail while the distribution of service S_1 has a small variance.

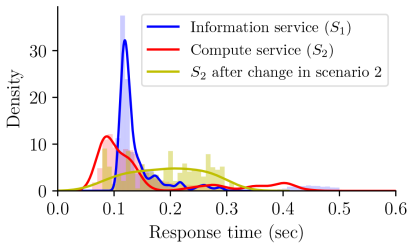
To illustrate the behavior of the base policies, we show the learning curve of the PPO policy $\widehat{\pi}_{\text{PPO}}$ and compare it with the performance of the greedy policy $\widehat{\pi}_{\text{GREEDY}}$, the random policy $\widehat{\pi}_{\text{RANDOM}}$, and the optimal policy $\pi^\star$. This is shown in Figure 3c. After convergence, the PPO policy $\widehat{\pi}_{\text{PPO}}$ performs close to optimal, the greedy policy $\widehat{\pi}_{\text{GREEDY}}$ performs worse, and the random policy $\widehat{\pi}_{\text{RANDOM}}$ is the least effective policy.

Figure 3a compares the performance of the three base policies (blue curves) with the respective rollout policies (red curves) with 1-step lookahead (i.e., $l = 1$) for 20 control cycles. The optimal policy is indicated with the dashed black line. We observe that the performances of the PPO and the greedy rollout policies (i.e., $\tilde{\pi}_{\text{PPO}}$ and $\tilde{\pi}_{\text{GREEDY}}$) are close to the performance of the optimal policy, while the performance of the random rollout policy $\tilde{\pi}_{\text{RANDOM}}$ is not. In fact, $\tilde{\pi}_{\text{RANDOM}}$ performs even worse than the base policy $\widehat{\pi}_{\text{RANDOM}}$. Recall that $\widehat{\pi}_{\text{PPO}}$ and $\widehat{\pi}_{\text{GREEDY}}$ are sequentially consistent and therefore, the rollout policies $\tilde{\pi}_{\text{PPO}}$ and $\tilde{\pi}_{\text{GREEDY}}$ are guaranteed to improve the base policies $\widehat{\pi}_{\text{PPO}}$ and $\widehat{\pi}_{\text{GREEDY}}$, which is consistent with our results (see Corollary 1). (In contrast, $\widehat{\pi}_{\text{RANDOM}}$ is not sequentially consistent.)

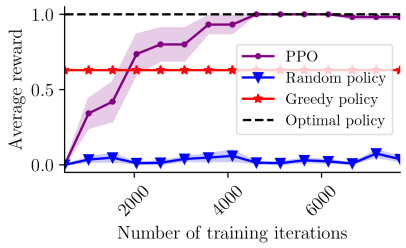
We investigate different lookahead horizons l. Since the greedy rollout policy $\tilde{\pi}_{\text{GREEDY}}$ and the PPO rollout policy $\tilde{\pi}_{\text{PPO}}$ achieve optimal performance with the lookahead horizon $l = 1$, increasing l does not give any improvement. Figure 3e shows measurements of the performance of the random rollout policy $\tilde{\pi}_{\text{RANDOM}}$ with different lookahead horizons. We observe that the performance increases significantly from $l = 1$ to $l = 2$ where the policy has optimal performance. It then decreases from $l = 2$ to $l = 3$ and slightly increases again from $l = 3$ to $l = 4$. While we expect an increasing performance



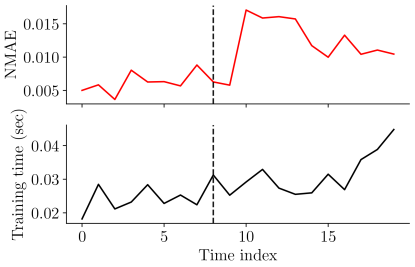
(a) Scenario 1: Policy adaptation of three base policies in simulation for $t = 1, 2, \ldots, 20$ with $l = 1$ for scenario 1; the curves indicate the mean and standard deviation ($\pm\frac{\sigma}{\sqrt{5}}$) over five evaluations with different random seeds.



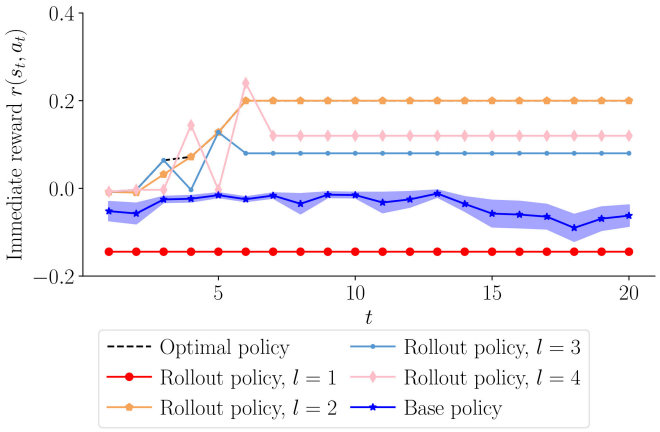
(b) Scenarios 1 and 2: The histograms show empirical distributions of response times d_1 and d_2; the overlaid curves depict the distribution of predictions generated by fitted random forest models.



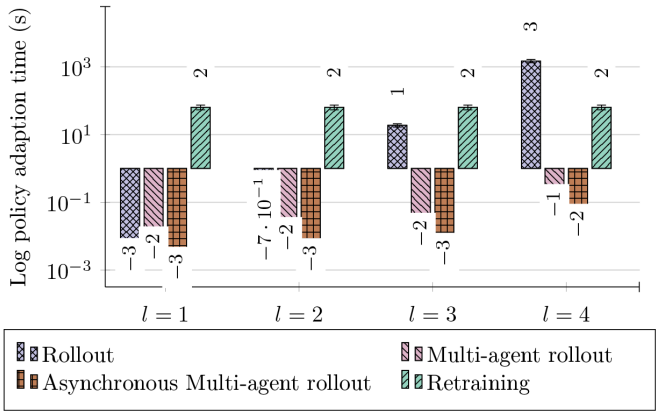
(c) Scenario 1: Learning curves from offline training of $\widehat{\pi}_{\text{PPO}}$ in simulation; the curves indicate the mean and standard deviation ($\pm\frac{\sigma}{\sqrt{5}}$) over five evaluations with different random seeds.



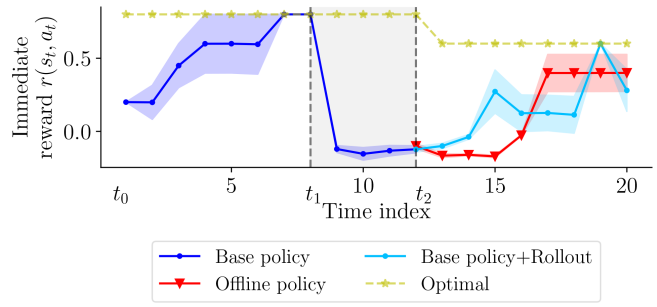
(d) Scenario 2: Performance of the estimated system model in terms of NMAE; the average NMAE before the change is 0.0079 and the average NMAE after the change is 0.0135.



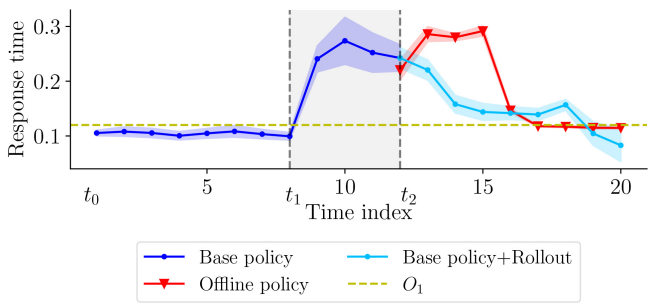
(e) Scenario 1: Online policy adaptation of the random base policy $\widehat{\pi}_{\text{RANDOM}}$ for varying lookahead horizons $l = 1, \ldots, 4$; the curves indicate the mean and standard deviation ($\pm\frac{\sigma}{\sqrt{5}}$) over five evaluations with different random seeds.



(f) Scenario 1: Time for adapting the PPO base policy $\widehat{\pi}_{\text{PPO}}$, using different methods: the single-agent rollout (purple), multi-agent rollout (pink), and asynchronous multi-agent rollout (beige); the green bars represent the offline retraining baseline; the error bars indicate standard deviations from 3 measurements.



(g) Scenario 2: Reward trajectories of the PPO base policy $\widehat{\pi}_{\text{PPO}}$ and the rollout policy on the target system for $l = 1$; the curves indicate the mean and standard deviation ($\pm\frac{\sigma}{\sqrt{5}}$) over five evaluations with different random seeds.



(h) Scenario 2: Response times of the simulated system when running the PPO base policy $\widehat{\pi}_{\text{PPO}}$ and the rollout policy on the target system for $l = 1$; the curves indicate the mean and standard deviation ($\pm\frac{\sigma}{\sqrt{5}}$) over five evaluations with different random seeds.

Fig. 3. Evaluation results for scenarios 1 and 2.

from $l = 1$ to $l = 2$, we do not fully understand the behavior of the system for $l > 2$.

We study the computing time for the PPO rollout policies $\tilde{\pi}_{\text{PPO}}$ for different configurations. Figure 3f shows the computing time for different lookahead horizons, as well as for single-agent and multi-agent configurations. With multi-agent, we measure synchronous versus asynchronous setups (see §II-D). We observe that multi-agent rollout has shorter computing times than single-agent rollout, which we can explain with the reduced size of the action space (see §II-D). As we expect, the computing time increases when l becomes larger. This increase is larger for single agents than for multi-agents due to the above reasons. If we compare single-agent rollout with offline retraining, we see that for horizon $l < 3$, rollout takes orders of magnitude less time. The computing time becomes comparable for $l \geq 3$.

B. Evaluation Results for Scenario 2

As described in §V-C, this scenario is divided into two time intervals: one before the change in the target system ($[t_0, t_1]$) and one after the change ($[t_1, \infty)$). Figures 3g and 3h show the reward trajectories and the response time trajectories obtained when running the PPO base policy $\widehat{\pi}_{\text{PPO}}$ and the rollout policy $\tilde{\pi}_{\text{PPO}}$ on the target system (i.e., the KTH testbed). We observe that the performance of $\widehat{\pi}_{\text{PPO}}$ is close to optimal before the change but drops drastically after the change at time t_1. After time t_2, the rollout policy $\tilde{\pi}_{\text{PPO}}$ quickly improves the performance of the system. When comparing the results from the simulator with those from the target system, we note that the figures from the target system show higher variance and slightly worse performance. In this scenario, the performance of rollout with $l > 1$ is the same as that of rollout with $l = 1$. The execution times for both standard rollout and multi-agent rollout exhibit similar trends as those observed in scenario 1.

Lastly, Fig. 3d shows the evolution of the predicted response time and the prediction error measured in NMAE of the estimated system model. We observe that after the change in the target system (indicated by the dashed vertical line), the error doubles. This is expected as the system model is not updated immediately when the change occurs. We note that as time progresses after the change, the system model is updated which causes the error to decrease gradually.

C. Discussion of the Evaluation Results

The key findings can be summarized as follows.

(i) Compared with reoptimization, rollout can provide a computationally effective way to adapt a base policy to near-optimal performance.

(ii) Using PPO or greedy base policies, rollout with a lookahead of $l = 1$ can significantly improve the performance of the base policy.

(iii) In case of a multi-dimensional control problem, as manifested in the scenarios we studied, the multi-agent rollout can drastically reduce the needed computing time for the rollout policy.

VII. Related work

Existing approaches for policy adaptation in networked systems include offline reinforcement learning [2]–[5], [10], dynamic programming [9], online reinforcement learning [11], [26]–[28], transfer learning [8], [29]–[31], and meta learning [7], [32]–[34].

The most common approach for policy adaptation in networked systems is offline retraining. Examples of previous work that use offline retraining in the context of networked systems include [2]–[6], [9], [10]. The main advantage of our approach compared to these references is that it is more efficient. In particular, we show that we are able to obtain the same performance as offline retraining by using one step of policy iteration online, which is significantly more efficient than retraining (see Fig. 3f).

Online model-free learning approaches adapt policies by updating them for each new measurement obtained from the target system. Examples of previous work that use online learning in the context of networked systems include [11], [26]–[28], [30]. The main benefit of our rollout approach compared to these works is that it has stronger theoretical guarantees (see §II).

Transfer learning and meta learning refer to reinforcement learning methods that aim to speed up the learning process by leveraging commonalities between different control tasks and domains. Examples of previous works that use transfer learning and meta learning for policy adaptation in networked systems include [7], [8], [29]–[36]. The main benefit of our rollout approach compared to these approaches is that it has stronger theoretical guarantees. In particular, our approach is guaranteed to improve the base policy (see Prop. 2 and Cor. 1). No such guarantee is available for the transfer learning and meta learning approaches.

VIII. Conclusion and future work

We present a novel approach to policy adaptation in networked systems based on the concept of rollout. In this approach, a given base policy is transformed it into an improved rollout policy. We perform an experimental evaluation on a testbed where we run information and computing services on a service mesh and study resource control policies. We find that rollout policies often achieve comparable performance to policies obtained through offline retraining but at much lower computing costs. We conclude that rollout, which has been successfully applied in other domains, is an effective method for policy adaptation in networked systems. Our future work includes studying scalable methods for online training of system models.

REFERENCES

[1] D. Bertsekas, *Reinforcement learning and optimal control*. Athena Scientific, 2019.

[2] F. Shahab Samani and R. Stadler, "Dynamically meeting performance objectives for multiple services on a service mesh," in *2022 18th International Conference on Network and Service Management (CNSM)*, 2022, pp. 219–225.

[3] F. S. Samani, K. Hammar, and R. Stadler, "Demonstrating a system for dynamically meeting management objectives on a service mesh," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023, pp. 1–3.

[4] F. S. Samani and R. Stadler, "A framework for dynamically meeting performance objectives on a service mesh," *arXiv preprint arXiv:2306.14178*, 2023.

[5] S. Schneider, R. Khalili, A. Manzoor, H. Qarawlus, R. Schellenberg, H. Karl, and A. Hecker, "Self-learning multi-objective service coordination using deep reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3829–3842, 2021.

[6] D. Garg, N. C. Narendra, and S. Tesfatsion, "Heuristic and reinforcement learning algorithms for dynamic service placement on mobile edge cloud," *arXiv preprint arXiv:2111.00240*, 2021.

[7] H. Qiu, W. Mao, C. Wang, H. Franke, A. Youssef, Z. T. Kalbarczyk, T. Başar, and R. K. Iyer, "{AWARE}: Automate workload autoscaling with reinforcement learning in production cloud systems," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 387–402.

[8] P. Liu, G. Bravo-Rocca, J. Guitart, A. Dholakia, D. Ellison, and M. Hodak, "Scanflow-k8s: Agent-based framework for autonomic management and supervision of ml workflows in kubernetes clusters," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2022, pp. 376–385.

[9] J.-S. Yang, P. Liu, and J.-J. Wu, "Workload characteristics-aware virtual machine consolidation algorithmsgarg2021heuristic," in *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, 2012, pp. 42–49.

[10] K. Hammar and R. Stadler, "An online framework for adapting security policies in dynamic it environments," in *2022 18th International Conference on Network and Service Management (CNSM)*. IEEE, 2022, pp. 359–363.

[11] Z. Yang, P. Nguyen, H. Jin, and K. Nahrstedt, "Miras: Model-based reinforcement learning for microservice resource allocation over scientific workflows," in *2019 IEEE 39th international conference on distributed computing systems (ICDCS)*. IEEE, 2019, pp. 122–132.

[12] D. Bertsekas, *Rollout, Policy Iteration, and Distributed Reinforcement Learning*, ser. Athena scientific optimization and computation series. Athena Scientific, 2021. [Online]. Available: https://books.google.se/books?id=0FflzQEACAAJ

[13] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[14] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[15] H. Wang, M. Preuss, and A. Plaat, "Warm-start alphazero self-play search enhancements," in *Parallel Problem Solving from Nature–PPSN XVI: 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part II 16*. Springer, 2020, pp. 528–542.

[16] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline monte-carlo tree search planning," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/8bb88f80d334b1869781beb89f7b73be-Paper.pdf

[17] Kubernetes community, "Production-grade container orchestration," 2014. [Online]. Available at: https://kubernetes.io/, Accessed on: June 7, 2022.

[18] Istio community, "Simplify observability, traffic management, security, and policy with the leading service mesh," 2017. [Online]. Available at: https://istio.io/, Accessed on: June 7, 2022.

[19] Forough Shahab Samani and Kim Hammar, "Online policy adaptation through rollouts," [Online]. Available at: https://github.com/foroughsh/online_policy_adaptation_using_rollout/, Accessed on: Sep 28 7, 2023., 2023.

[20] R. Bellman, *Dynamic Programming*. Dover Publications, 1957.

[21] D. Bertsekas, *Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control*, ser. Athena Scientific optimization and computation series. Athena Scientific, 2022. [Online]. Available: https://books.google.se/books?id=KRllEAAAQBAJ

[22] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback control of computing systems*. John Wiley & Sons, 2004.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, 2017, http://arxiv.org/abs/1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347

[24] L. Breiman, "Random Forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[25] Sklearn community, "sklearn.ensemble.RandomForestClassifier," 2007. [Online]. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, Accessed on: June 7, 2022.

[26] L. Chen, J. Xu, S. Ren, and P. Zhou, "Spatio–temporal edge service placement: A bandit learning approach," *IEEE Transactions on Wireless Communications*, vol. 17, no. 12, pp. 8388–8401, 2018.

[27] Z. Wang, P. Li, C.-J. M. Liang, F. Wu, and F. Y. Yan, "Autothrottle: A practical framework for harvesting cpus from slo-targeted microservices," *arXiv preprint arXiv:2212.12180*, 2022.

[28] F. Rossi, V. Cardellini, and F. L. Presti, "Self-adaptive threshold-based policy for microservices elasticity," in *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2020, pp. 1–8.

[29] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, and R. K. Iyer, "Firm: An intelligent fine-grained resource management framework for slo-oriented microservices," in *Proceedings of The 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*, 2020.

[30] Q. Fettes, A. Karanth, R. Bunescu, B. Beckwith, and S. Subramoney, "Reclaimer: A reinforcement learning approach to dynamic resource allocation for cloud microservices," *arXiv preprint arXiv:2304.07941*, 2023.

[31] W. Li, B. Liu, H. Gao, and X. Su, "Transfer learning based algorithm for service deployment under microservice architecture," in *International Conference on Communications and Networking in China*. Springer, 2021, pp. 52–62.

[32] H. Liu, P. Chen, and Z. Zhao, "Towards a robust meta-reinforcement learning-based scheduling framework for time critical tasks in cloud environments," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 2021, pp. 637–647.

[33] H. Liu, P. Chen, X. Ouyang, H. Gao, B. Yan, P. Grosso, and Z. Zhao, "Robustness challenges in reinforcement learning based time-critical cloud resource scheduling: A meta-learning based solution," *Future Generation Computer Systems*, vol. 146, pp. 18–33, 2023.

[34] X. Xiu, J. Li, Y. Long, and W. Wu, "Mrlcc: an adaptive cloud task scheduling method based on meta reinforcement learning," *Journal of Cloud Computing*, vol. 12, no. 1, pp. 1–12, 2023.

[35] S. Li, L. Wang, W. Wang, Y. Yu, and B. Li, "George: Learning to place long-lived containers in large clusters with operation constraints," in *Proceedings of the ACM Symposium on Cloud Computing*, 2021, pp. 258–272.

[36] S. Xue, C. Qu, X. Shi, C. Liao, S. Zhu, X. Tan, L. Ma, S. Wang, S. Wang, Y. Hu *et al.*, "A meta reinforcement learning approach for predictive autoscaling in the cloud," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 4290–4299.