Efficient Reinforcement Learning for Routing Jobs in Heterogeneous Queueing Systems

Neharika Jali **Guannan Qu** **Weina Wang** **Gauri Joshi**
Carnegie Mellon University

Abstract

We consider the problem of efficiently routing jobs that arrive into a central queue to a system of heterogeneous servers. Unlike homogeneous systems, a threshold policy, that routes jobs to the slow server(s) when the queue length exceeds a certain threshold, is known to be optimal for the one-fast-one-slow two-server system. But an optimal policy for the multi-server system is unknown and non-trivial to find. While Reinforcement Learning (RL) has been recognized to have great potential for learning policies in such cases, our problem has an exponentially large state space size, rendering standard RL inefficient. In this work, we propose ACHQ, an efficient policy gradient based algorithm with a low dimensional soft threshold policy parameterization that leverages the underlying queueing structure. We provide stationary-point convergence guarantees for the general case and despite the low-dimensional parameterization prove that ACHQ converges to an approximate global optimum for the special case of two servers. Simulations demonstrate an improvement in expected response time of up to $\sim 30\%$ over the greedy policy that routes to the fastest available server.

1 INTRODUCTION

With the recent exponential increase in large-scale cloud based services, we observe a paradigm shift in the nature of these systems and the way they serve jobs. A case in point is devices across generations of technology with varying capabilities present in the cluster. It is not uncommon to find a ~ 4 TFLOP K80, a ~ 30 TFLOP L4 and a ~ 100 TFLOP TPUv3 in the same facility (Google, 2023b,a). Another example is the differences in server speeds arising due to the fractional allocation of resources in serverless computing setups (Amazon, 2023). ML inference deployments with different servers hosting different sized models is yet another instance of varying service times (Shazeer et al., 2017). Motivated by such heterogeneous systems, in this work, we consider the problem of efficient routing of jobs in queueing systems with servers of different speeds. We consider a stylized model with a single central queue and a system of heterogeneous servers as shown in Figure 1. Besides the cloud computing systems that motivated us, this model and the proposed routing policies are also applicable in other domains such as packet routing in communication networks (Srikant and Ying, 2013) or operations resource management in healthcare, manufacturing or ride-sharing (Walton and Xu, 2021; Tsitsiklis and Xu, 2017).

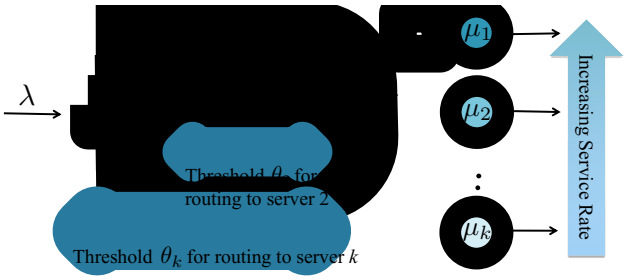


Figure 1: k Server heterogeneous queueing system with service rates μ_i, and job arrival rate λ. We also illustrate a threshold routing policy that routes jobs to a slower server i when the queue length is $> \theta_i$.

Traditional methods in queueing are often designed for homogeneous systems and do not account for the heterogeneity in service rates (Harchol-Balter, 2013). While these policies may still be efficient in particular load regimes or for certain types of heterogeneity, they are not optimal for the general case. For instance, a work-conserving routing policy that keeps all servers busy whenever there are jobs in the system is latency-

Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

optimal for homogeneous systems. However, it is sub-optimal for heterogeneous systems because it can be beneficial to keep slow servers idle and hold jobs in the central queue until the queue length exceeds a certain threshold or one of the fast server(s) becomes available. For the case of two servers - one fast and one slow - such a threshold policy has been shown to be optimal (Lin and Kumar, 1984; Koole, 1995; Walrand, 1984). Despite many efforts, extending the proof of optimality of the threshold policy to a general multi-server case has been open for nearly four decades (Koole, 2022). Moreover, the threshold is a complicated and unknown function of the service rates and the job arrival rate.

Main Contributions. Motivated by difficulty in finding a closed-form expression for the policy, we present a reinforcement learning (RL) approach for finding the best routing policy in multi-server heterogeneous queueing systems. We make the following key contributions to address the inefficiency of off-the-shelf RL methods arising due to a large dimensional state space. (a) We leverage the underlying queueing structure to design a low-dimensional soft threshold policy parameterization and propose ACHQ, an efficient policy gradient-based algorithm. (b) We provide stationary-point guarantees, and for the special case of two servers, establish convergence to an approximate global optimum. (c) We demonstrate an improvement in expected response time of up to $\sim 30\%$ over the greedy policy that routes to the fastest available server. To the best of our knowledge, ours is the first to propose an RL approach to this problem and one of few recent papers that leverage the queueing structure to design an efficient RL approach with provable guarantees.

2 RELATED WORK

The problem of heterogeneous servers was first proposed by Larsen (1981) in which the optimal policy was conjectured to be of threshold type. For two servers, a threshold policy was proved to be optimal using techniques of policy iteration (Lin and Kumar, 1984), value iteration (Koole, 1995) and sample path arguments (Walrand, 1984). While Rykov (2001) claimed to have extended the proof to the general case of multiple heterogeneous servers, De Vericourt and Yong-Pin (2006) later showed that it was incomplete. The completeness and correctness of the proof in another attempt in Luh and Viniotis (2002) has also been questioned by the authors of De Vericourt and Yong-Pin (2006). Hence, despite several attempts, proving the optimality of a policy for the general multi-server case has been an open problem for nearly 40 years (Koole, 2022).

Given the above challenges in deriving closed-form optimal policies, RL and approximate dynamic programming have been a natural choice for designing data-driven policies for queueing systems. Several works use these techniques effectively in applications such as scheduling in queueing networks (Moallemi et al., 2008), inventory control (Mannor et al., 2003), emergency response assignment (van Barneveld et al., 2018) and cooling in Google datacenters (DeepMind, 2023).

In recent years, there has been a renewed interest with a focus on large scale settings with possibly incomplete information and establishing guarantees of stability, convergence and correctness (Ayesta, 2022; Walton and Xu, 2021). Dai and Gluzman (2022) addresses the challenges of infinite state space, unbounded costs, and long-run average cost objective in queueing network control problems by proposing PPO and TRPO based deep RL algorithms with the policy optimization step enhanced by Lyapunov function arguments. Liu et al. (2022) proposes a truncation-based solution and establishes guarantees of optimality in the setting of unbounded state space by applying RL methods over a finite subset of the state space and a known stabilizing policy for the rest. Robledo et al. (2022) focuses on developing intermediate solutions between model-free and model-based methods by exploiting the queueing structure to develop efficient learning algorithms. An alternate line of empirical work include Staffolani et al. (2023) that implements a DoubleDQN method to learn task allocation in distributed queues and Mao et al. (2019) that designs a REINFORCE based algorithm to learn workload specific efficient scheduling policies in distributed data processing jobs with dependency graphs. While further instances of RL in queues can be found in Section 5 of Walton and Xu (2021) and Ayesta (2022), to the best of our knowledge, our work is the first to study the multi-server heterogeneous problem from an RL perspective.

Reinforcement Learning approaches applied to large scale systems often suffer from state space explosion and standard algorithms are plagued by the curse of dimensionality. Actor Critic Policy Gradient offer a low dimensional solution by parameterizing the value function and policy. While actor-only methods are at a disadvantage due to high variance and inefficient use of samples between parameter updates and critic-only ones lack guarantees of optimality of resulting policy, actor critic methods provide the best of both worlds. We include works most relevant to our problem setting that establish guarantees of convergence below.

Konda and Tsitsiklis (1999) presents an asymptotic analysis of the average cost two timescale actor critic algorithm with a linear approximation of action-value function. A finite time convergence to a stationary point is established in Wu et al. (2020) for the average cost two timescale advantage actor critic algorithm

with a TD update of the critic and linear approximation of the state value function without assuming a compatible function approximation. Further, the authors also remark that their results may be extendable to the discounted setting using the same proof technique. Kumar et al. (2023) analyze the finite time discounted cost actor critic algorithm with a Monte Carlo based update of the critic which is a linearly approximated action value function. A drawback of this paper is that it assumes that the action value function is indeed a linear function and can be approximated without error which might not be true for real world problems in general.

While there are a few papers that prove convergence to the global optimum for the special cases of linear MDPs and linear quadratic regulators, convergence for a general MDP is studied sparingly. Agarwal et al. (2021) proposes a discounted cost actor critic based Natural Policy Gradient algorithm with linear approximation of the action value function and compatible function approximation and establishes its global convergence. Bhandari and Russo (2024) provide guarantees of optimality of a stationary point for a general policy gradient algorithm in the discounted setting when certain conditions of differentiability, closure and structure of the policy iteration objective are satisfied. To the best of our knowledge, demonstrating global convergence for the general MDP in the average cost setting is an open problem.

3 PROBLEM FORMULATION

We first formally define the model and describe the characteristics of an optimal policy. We then model the problem as a Markov decision process. Finally, we demonstrate that standard RL methods suffer from the curse of dimensionality due to a large state space.

3.1 Queueing Setup

Model. We consider a queueing model with a single central queue and k heterogeneous servers as shown in Figure 1. Jobs arrive into the queue according to a Poisson process with a known parameter $\lambda > 0$. The queue is limited to hold at most l_M jobs. A router then decides when and which server to route a job to according to a policy π. The time a job spends in service at server i is modeled as an exponential random variable with a rate μ_i which is also known to the router. Without loss of generality, we assume that the servers are ordered as $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_k$. The system is stable when $\lambda < \sum_{i=1}^{k} \mu_i$. We consider non-preemptive execution, where a job once routed to a server cannot be recalled or interrupted until it runs to completion.

Performance Metric. The response time $T_{r,i}$ of the i-th job is defined as the time it takes from its arrival into the queue to its exit from the system after service which is a sum of its waiting and service times. The expected response time of the system following policy π can be written as

$$T_r = \lim_{n\to\infty} \frac{1}{n} \mathbb{E}_\pi \left[\sum_{i=0}^{n} T_{r,i} \right] \tag{1}$$

where $\mathbb{E}_\pi$ denotes the expectation with respect to the stochastic process when policy π is used. The objective, thus, is to find a policy that minimizes the expected response time T_r.

Slow Now versus Fast Later. If the goal were to maximize throughput instead of average response time, the problem becomes trivial and the optimal policy would be to just keep all the servers busy. For our objective (1), a greedy policy that always routes to the fastest among the available servers is not optimal because faster server(s) can become available soon after the router assigns all jobs in the queue to slower server(s). Thus, the goal of a policy optimizing the expected response time is to balance the trade-off between sending a job right now to a slower server, or waiting for a faster server to become available.

Threshold Policy. As depicted in Figure 1, a threshold policy is a rule that chooses to route a job to the fastest available server (say server i) when the number of jobs waiting in the queue exceeds a threshold θ_i and wait for the next timestep otherwise. Note that the threshold $\theta_1 = 0$, since it is optimal to route jobs to the fastest server whenever it is available. Such a policy captures the intuition that if there are a lot of jobs in the queue and there is pressure to clear the backlog, the router should send a job to a slower server right now. But if there are only a few jobs in the queue, then it should wait for a faster server to become available. For a queueing system with only two servers, one fast and one slow, a threshold policy has been proved to be optimal (Lin and Kumar, 1984; Koole, 1995; Walrand, 1984). In this work, we consider the more general k server case, which has been an open problem for nearly four decades (Koole, 2022).

3.2 Queue as a Markov Decision Process

The Markov chain view of the queue renders utilizing tools from reinforcement learning a natural choice. We model the queue as an average cost discrete-time Markov decision process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, c)$ where $\mathcal{S}, \mathcal{A}$ are the state and action spaces respectively, $\mathcal{P}$ the transition probability and c the cost. The

continuous time queue described in the previous subsection can be converted to an equivalent discrete-time representation by sampling the systems at instants of arrivals and departures (Lippman, 1975). Hence, we have a discrete-time system, where at each timestep, whose beginning is marked by an arrival or a departure, the router decides if and to which server to send a job waiting in the queue.

The state of the system at time t is represented as $\boldsymbol{s}^{(t)} = (L^{(t)}, B_1^{(t)}, \ldots, B_k^{(t)})$ where $L^{(t)} \in [l_M]$ is the number of jobs in the queue and $B_i^{(t)} \in \{0, 1\}$ represents if server i is available or busy respectively. An action $a \in \mathcal{A}$ corresponds to routing a job to an idle server i, $a = i$, or keeping the job in the queue and choosing to wait for the next time step, $a = 0$. Notice that the state space scales exponentially in the number of servers $|\mathcal{S}| = (l_M + 1)2^k$ and algorithms that depend on the size of the state space suffer from the curse of dimensionality.

State transitions occur whenever there is an arrival or a departure and the corresponding transition probabilities $\Pr(\boldsymbol{s}'|\boldsymbol{s}, a)$ are a function of the arrival λ and service rates $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_k)$ and the action taken. Note that the finite Markov chain under consideration is irreducible and aperiodic, which implies ergodicity. For ergodic systems, by Little's Law, the objective of minimizing the expected response time is equivalent to minimizing the time-averaged number of jobs in the system (including both those waiting in queue and those being currently served) (Harchol-Balter, 2013). We, hence, define the cost as the total number of jobs in the system $c(\boldsymbol{s}) = \sum_i s_i = L + \sum_j B_j$, where s_i denotes the i-th element of $\boldsymbol{s}$. The policy is represented as $\pi(\cdot|\boldsymbol{s})$. Denote the stationary distribution over the states induced by the policy as ν_π.

The average cost is defined as

$$c_\pi := \lim_{T\to\infty} \frac{\sum_{t=0}^{T} c(\boldsymbol{s}^{(t)})}{T} = \mathbb{E}_{\boldsymbol{s}\sim\nu_\pi}[c(\boldsymbol{s})]. \tag{2}$$

Further, we also define the overall cost accumulated when starting from state $\boldsymbol{s}^{(0)}$ and following policy π by the state-value function

$$V^\pi(\boldsymbol{s}) := \mathbb{E}\left[\sum_{t=0}^{\infty}(c(\boldsymbol{s}^{(t)}) - c_\pi)|\boldsymbol{s}^{(0)} = \boldsymbol{s}\right].$$

Another useful quantity is the action-value function

$$Q^\pi(\boldsymbol{s}, a) := \mathbb{E}\left[\sum_{t=0}^{\infty}(c(\boldsymbol{s}^{(t)}) - c_\pi)|\boldsymbol{s}^{(0)} = \boldsymbol{s}, a^{(0)} = a\right].$$

The goal thus is to find the optimal policy $\pi^\star(\cdot|\boldsymbol{s})$ that minimizes the average cost, $\pi^\star = \arg\min_\pi c_\pi$.

3.3 Relative Value Iteration: The Curse of Dimensionality

Recall that for the multi-server case, the optimal policy is unknown. Relative Value Iteration (RVI) which is the Value Iteration algorithm adapted to average cost (White, 1963) addresses this. We include a brief description below and pseudo-code in Appendix C.

Define the Bellman operator over the value function as

$$T(V)(\boldsymbol{s}) = \min_{a\in\mathcal{A}} c(\boldsymbol{s}) + \sum_{\boldsymbol{s}'\in\mathcal{S}} \Pr(\boldsymbol{s}'|\boldsymbol{s}, a)V(\boldsymbol{s}').$$

Value Iteration computes $V(\boldsymbol{s})$ by iteratively updating values for every state using the Bellman equation in every round. Additionally, in the average cost case, RVI subtracts the value of a reference state $\boldsymbol{s}^\star \in \mathcal{S}$, from the values of other states to mitigate the numerical instability caused by large value functions (White, 1963)

$$V^{(t+1)}(\boldsymbol{s}) \leftarrow T(V^{(t)})(\boldsymbol{s}) - T(V^{(t)})(\boldsymbol{s}^\star).$$

While it is convenient to use RVI when there are a limited number of servers and a small buffer, it becomes computationally intractable in large scale real-world applications. As the state space scales exponentially in the number of servers, the complexity of each iteration of value function updates also scales exponentially. For example, there are $20,480$ states in a system with $k = 10$ servers and a buffer capacity of $l_M = 20$. To address this issue, we leverage the underlying structure in the queueing model and present a function approximation-based policy gradient algorithm in the next section.

4 ACTOR CRITIC FOR HETEROGENEOUS QUEUES

In this section, we present ACHQ – Actor Critic for Heterogeneous Queues. Algorithm 1 is a policy gradient method with function approximation that leverages threshold-like-policy properties of the queueing system. We first briefly outline the two time-scale Actor Critic algorithm (Sutton and Barto, 2018) and then describe the parameterization that we design to mitigate the curse of dimensionality.

4.1 Preliminaries

Consider the policy $\pi_{\boldsymbol{\theta}}(\cdot|s)$, also known as the *actor*, to be parameterized by $\boldsymbol{\theta}$. Denote the stationary distribution induced over the states by this policy by $\nu_{\boldsymbol{\theta}}$, value function by $V^{\boldsymbol{\theta}}$ and action-value function by $Q^{\boldsymbol{\theta}}$. The performance of $\pi_{\boldsymbol{\theta}}$ is measured by the expected cost under the stationary distribution $\nu_{\boldsymbol{\theta}}$ which is

$$J(\boldsymbol{\theta}) := c_{\pi_{\boldsymbol{\theta}}} = \mathbb{E}_{\boldsymbol{s}\sim\nu_{\boldsymbol{\theta}}}[c(\boldsymbol{s})]. \tag{3}$$

Algorithm 1 ACHQ

1: **Initialize** actor parameters $\boldsymbol{\theta}^{(0)}$; critic parameters $\boldsymbol{\omega}^{(0)}$; average cost estimator $\eta^{(0)}$; step sizes $\alpha^{(t)}$ for actor, $\beta^{(t)}$ for critic, $\zeta^{(t)}$ for average cost estimator

2: Draw $\boldsymbol{s}^{(0)}$ from some initial distribution
3: **for** $t = 0, 1, 2, \ldots$ **do**
4: Take action $a^{(t)} \sim \pi_{\boldsymbol{\theta}^{(t)}}(\cdot|\boldsymbol{s}^{(t)})$
5: Observe cost $c^{(t)} = c(\boldsymbol{s}^{(t)})$
6: Observe next state $\boldsymbol{s}^{(t+1)} \sim \Pr(\cdot|\boldsymbol{s}^{(t)}, a^{(t)})$
7: $\delta^{(t)} = c^{(t)} - \eta^{(t)} + \boldsymbol{\phi}(\boldsymbol{s}^{(t+1)})^T\boldsymbol{\omega}^{(t)} - \boldsymbol{\phi}(\boldsymbol{s}^{(t)})^T\boldsymbol{\omega}^{(t)}$

8: $\eta^{(t+1)} = \eta^{(t)} + \zeta^{(t)}(c^{(t)} - \eta^{(t)})$
9: $\boldsymbol{\omega}^{(t+1)} = \Pi_{R_{\boldsymbol{\omega}}}\left(\boldsymbol{\omega}^{(t)} + \beta^{(t)}\delta^{(t)}\boldsymbol{\phi}(\boldsymbol{s}^{(t)})\right)$
10: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha^{(t)}\delta^{(t)}\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}^{(t)}}(a^{(t)}|\boldsymbol{s}^{(t)})$
11: **end for**

Actor. The algorithm finds the optimal in the class of policies parameterized by $\boldsymbol{\theta}$ as $\boldsymbol{\theta}^\star = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ using gradient descent which can be represented as

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla J(\boldsymbol{\theta}^{(t)}).$$

By the Policy Gradient theorem, we can represent $\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{s}\sim\nu_{\boldsymbol{\theta}}, a\sim\pi_{\boldsymbol{\theta}}}[Q^{\boldsymbol{\theta}}(\boldsymbol{s}, a)\nabla \log \pi_{\boldsymbol{\theta}}(\boldsymbol{s}, a)]$ (Sutton and Barto, 2018). In practice, a baseline (such as V) is subtracted from the action-value function Q while estimating gradients to reduce variance which is as follows

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{s},a}[\Delta^{\boldsymbol{\theta}}(\boldsymbol{s}, a)\nabla \log \pi_{\boldsymbol{\theta}}(\boldsymbol{s}, a)]$$

where $\Delta^{\boldsymbol{\theta}}(\boldsymbol{s}, a) := Q^{\boldsymbol{\theta}}(\boldsymbol{s}, a) - V^{\boldsymbol{\theta}}(\boldsymbol{s})$ is the *advantage*.

Critic. To mitigate the exponential state space problem, we model the critic to be approximated by a linear function

$$\hat{V}(\boldsymbol{s}; \boldsymbol{\omega}) = \boldsymbol{\phi}(\boldsymbol{s})^T\boldsymbol{\omega}. \tag{4}$$

The gradient step for the critic can be represented as

$$\boldsymbol{\omega}^{(t+1)} = \boldsymbol{\omega}^{(t)} + \beta(V(\boldsymbol{s}^{(t)}) - \boldsymbol{\phi}(\boldsymbol{s}^{(t)})^T\boldsymbol{\omega})\boldsymbol{\phi}(\boldsymbol{s}^{(t)}).$$

Algorithm. Putting together the actor and critic, as described above, we have Algorithm 1. Expression in line 7 is a result of the advantage estimated as

$$\begin{aligned}\delta^{(t)} &= \hat{Q}(\boldsymbol{s}^{(t)}, a^{(t)}) - \hat{V}(\boldsymbol{s}^{(t)}) \\ &= c^{(t)} - \eta^{(t)} + \hat{V}(\boldsymbol{s}^{(t+1)}) - \hat{V}(\boldsymbol{s}^{(t)})\end{aligned}$$

where $\eta^{(t)}$ denotes the average cost until timestep t. This estimate is then used in the gradient step for the actor parameters in line 10. Using a TD(0) style update, for the critic parameters results in the expression in line 9 where $\Pi_{R_{\boldsymbol{\omega}}}$ represents the projection to appropriately chosen radius $R_{\boldsymbol{\omega}}$.

4.2 Low Dimensional Parameter Design

Recall that the number of states in our problem scales exponentially in the number of servers. To alleviate this issue of a very large state space, we design a low dimensional actor and critic leveraging properties of the queueing system. A threshold policy, as defined in Section 3.1, is known to be optimal for the two server case and is conjectured to be optimal for the multi-server system (Lin and Kumar, 1984). Moreover, we observe the optimal policy to be of threshold type in simulations in Section 6.1 for the multi-server model.

Policy Parameters. First, we note that if the fastest server is free and there is a job waiting in the queue, then the job is routed to the fastest server irrespective of the number of jobs waiting in the queue. This can be argued because the job can be served no faster than by the fastest server and hence there is no incentive to wait. We then parameterize the actor by $\boldsymbol{\theta} \in \mathbb{R}^{k-1}$ as a soft threshold policy with one threshold per server for servers $2, \ldots, k$. If $f = \arg\max_i \mu_i(1 - B_i)$ is the index of the fastest among the available servers in state $\boldsymbol{s}$, θ_f the threshold corresponding to it, s_0 the number of jobs in the queue and σ a hyperparameter that controls the slope or sharpness of the decision boundary, then the probability of routing a job to server f is

$$\pi_{\boldsymbol{\theta}}(a = f|\boldsymbol{s}) = \frac{e^{\sigma(s_0 - \theta_f)}}{1 + e^{\sigma(s_0 - \theta_f)}}. \tag{5}$$

Further, the probability of waiting until the next timestep is $\pi_{\boldsymbol{\theta}}(a = 0|\boldsymbol{s}) = 1 - \pi_{\boldsymbol{\theta}}(a = f|\boldsymbol{s})$. This choice of parameterization is a soft, differentiable version of the threshold policy conjectured to be optimal in Lin and Kumar (1984).

Value Function Approximation. We choose the features for linear approximation of the value function

$$\boldsymbol{\phi}(\boldsymbol{s}) = \frac{\boldsymbol{s}}{l_M + k}. \tag{6}$$

This reflects the intuition that the more the number of jobs in the system, the longer it takes to serve them and hence higher is the cost. Note that the feature vector is normalized by $l_M + k$ to ensure that $||\boldsymbol{\phi}(\cdot)|| < 1$.

5 CONVERGENCE GUARANTEES

We first show that ACHQ converges to a stationary point for the multi-server case by applying results from Wu et al. (2020). We then prove that the stationary point is an approximate global optimum for the special case of two servers in the discounted cost setting by applying results from Bhandari and Russo (2024).

5.1 Convergence to Stationary Point

In this subsection, we show that the soft threshold policy defined in Equation (5) converges to a first order stationary point by applying the results of finite time two timescale actor critic methods from Wu et al. (2020). We detail the assumptions considered, verify that they hold for our problem and then state the convergence rate and sample complexity for Algorithm 1.

Assumption 5.1 (Bounded Feature Norm)**.** *The norm of the feature vector is bounded i.e.* $||\boldsymbol{\phi}(\cdot)|| < 1$.

Assumption 5.2 (Negative Definite $\boldsymbol{A}$)**.** *For all policy parameters* $\boldsymbol{\theta}$, $\boldsymbol{A} := \mathbb{E}_{\boldsymbol{s},a,\boldsymbol{s}'}[\boldsymbol{\phi}(\boldsymbol{s})(\boldsymbol{\phi}(\boldsymbol{s}') - \boldsymbol{\phi}(\boldsymbol{s}))^T]$ *is negative definite.*

Assumption 5.3 (Uniform Ergodicty)**.** *Consider the Markov chain generated by the rule* $a^{(t)} \sim \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s}^{(t)})$, $\boldsymbol{s}^{(t+1)} \sim \Pr(\cdot|\boldsymbol{s}^{(t)}, a^{(t)})$ *and the stationary distribution* $\nu_{\boldsymbol{\theta}}$ *induced by policy* $\pi_{\boldsymbol{\theta}}$*. There exists an* $m > 0$ *and* $\chi \in (0,1)$ $\forall \tau \geq 0, \forall \boldsymbol{s} \in \mathcal{S}$ *such that*

$$d_{TV}(\Pr(\boldsymbol{s}^{(\tau)} \in \cdot|\boldsymbol{s}^{(0)} = \boldsymbol{s}), \nu_{\boldsymbol{\theta}}(\cdot)) \leq m\chi^{\tau}.$$

Assumption 5.4 (Bounded and Lipschitz Continuous Gradients and Policy)**.** *There exists constants* $y_1, y_2, y_3 > 0$ *such that for all given states* $\boldsymbol{s}$*, actions* a *and* $\forall \boldsymbol{\theta} \in \mathbb{R}^k$

$$||\nabla \log \pi_{\boldsymbol{\theta}}(a|\boldsymbol{s})|| \leq y_1,$$
$$||\nabla \log \pi_{\boldsymbol{\theta}_1}(a|\boldsymbol{s}) - \nabla \log \pi_{\boldsymbol{\theta}_2}(a|\boldsymbol{s})|| \leq y_2||\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2||,$$
$$|\pi_{\boldsymbol{\theta}_1}(a|\boldsymbol{s}) - \pi_{\boldsymbol{\theta}_2}(a|\boldsymbol{s})| \leq y_3||\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2||.$$

Assumption 5.5 (Bounded Value Function Approximation Error)**.** *The value function approximation error*

$$\epsilon_{app}(\boldsymbol{\theta}) := \sqrt{\mathbb{E}_{\boldsymbol{s}\sim\nu_{\boldsymbol{\theta}}}\left[(\boldsymbol{\phi}(\boldsymbol{s})^T\boldsymbol{\omega}^{\star}(\boldsymbol{\theta}) - V^{\boldsymbol{\theta}}(\boldsymbol{s}))^2\right]}$$

is uniformly bounded for all potential policies by some constant $\epsilon_{app} \geq 0$, $\forall \boldsymbol{\theta}$ *as* $\epsilon_{app}(\boldsymbol{\theta}) \leq \epsilon_{app}$.

Assumption Verification. Assumption 5.1 is ensured by design in Equation (6). Assumption 5.2 is equivalent to Φ, the matrix whose columns are $\boldsymbol{\phi}(\boldsymbol{s})$, being full rank (Zhang et al., 2021). This condition is satisfied in our case due to the presence of $\boldsymbol{e}_i$ among columns of Φ where e_i is a vector with all but i-th element as zeros and i-th element as $1/(l_M + k)$. Assumption 5.3 is fulfilled due to the irreducibility and aperiodicity of our finite Markov chain (Bhandari et al., 2018). It is easy to verify that properties in Assumption 5.4 hold for our policy parameterization in Equation (5). Satisfaction of Assumption 5.5 is demonstrated by experiments in Section 6.1 that show the value function is approximately linear in the state vector. Note that proving the value function to be linear in the state vector is a hard problem since the optimal policy is unknown for the multi-server system.

We now formally state the convergence rate and sample complexity. Note that this result follows from Theorem 4.5, 4.7 of Wu et al. (2020) that establish the convergence of the actor and critic respectively.

Theorem 5.1 (Wu et al. (2020), Corollary 4.9)**.** *Under assumptions 5.1-5.5, choosing the actor step size* $\alpha^{(t)} = \mathcal{O}(1/(1+t)^{r_\alpha})$ *and the critic step size* $\beta^{(t)} = \mathcal{O}(1/(1+t)^{r_\beta})$*, where* $0 < r_\beta < r_\alpha < 1$*, we have*

$$\min_{0\leq i\leq t} \mathbb{E}||\nabla J(\boldsymbol{\theta}^{(i)})||^2 = \mathcal{O}(\epsilon_{app}) + \mathcal{O}\left(\frac{1}{t^{1-r_\alpha}}\right) + \mathcal{O}\left(\frac{\log t}{t^{r_\beta}}\right) + \mathcal{O}\left(\frac{1}{t^{2(r_\alpha - r_\beta)}}\right). \tag{7}$$

Algorithm 1 can find an ϵ*-approximate stationary point of* $J(\cdot)$ *within* τ *steps as*

$$\min_{0\leq i\leq \tau} \mathbb{E}||\nabla J(\boldsymbol{\theta}^{(i)})||^2 \leq \mathcal{O}(\epsilon_{app}) + \epsilon, \tag{8}$$

where $r_\alpha = 3/5, r_\beta = 2/5$ *and the total number of iterations* $\tau = \tilde{\mathcal{O}}(\epsilon^{-2.5})$.

5.2 Approximate Optimality of Stationary Point for Two Servers

While we proved that ACHQ converges to a stationary point in the previous section, we are yet to establish global optimality of such a stationary point. Moreover, it is unknown whether the optimal policy even lies in the class of threshold policies. Recall that finding an optimal policy for the general multi-server case is an open problem in queueing theory. Here, we consider the special case of two servers and a discounted cost MDP. We show that any stationary point is approximately optimal by using results from Bhandari and Russo (2024). We define some preliminaries, describe the required conditions, show that they hold for our queueing model and finally state the formal result.

Here we look at the special case of two servers $k = 2$ - one fast and one slow. Recall that the optimal policy is proven to be of threshold type for this case. Further, we consider the discounted cost MDP here. While we consider the average cost MDP in Algorithm 1 and the guarantees of convergence to a stationary point in Section 5.1, we note that a global optimality analysis for the average cost MDPs is still an open problem. We also observe empirically in Section 6.2, a similar performance (and often with fewer samples) with a discounted cost. We continue to use the soft threshold policy parameterization described in Equation (5) and represent by Θ the class of such policies.

The state-value function $V_\gamma^\pi(\boldsymbol{s})$ is now defined as the overall discounted cost accumulated when starting from

state $\boldsymbol{s}$ and following policy π

$$V_\gamma^\pi(\boldsymbol{s}) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t c(\boldsymbol{s}^{(t)}) | \boldsymbol{s}^{(0)} = \boldsymbol{s}\right]$$

where γ is the discount factor. Represent the distribution from which the initial state is chosen by ξ. The performance measure can hence be written as

$$J_\gamma(\pi) = \mathbb{E}_{\boldsymbol{s}\sim\xi}\left[V^\pi(\boldsymbol{s})\right]$$

and the optimal policy is $\pi^\star = \arg\min_\pi J_\gamma(\pi)$.

Define the weighted Policy Iteration (PI) objective or the "Bellman" cost function as

$$\begin{aligned}\mathcal{B}\left(\bar{\pi}|\nu, V_\gamma^\pi\right) &:= \mathbb{E}_{\boldsymbol{s}\sim\nu}\left[(T_{\bar{\pi}} V_\gamma^\pi)(\boldsymbol{s})\right] \\ &= \mathbb{E}\left[Q_\gamma^\pi(\boldsymbol{s}, \bar{\pi}(\boldsymbol{s}))\right] \end{aligned} \tag{9}$$

for a probability distribution ν over state space $\mathcal{S}$ where $T_{\bar{\pi}}$ is the Bellman operator with respect to policy $\bar{\pi}$. Further, define the effective concentrability coefficient κ_ξ for the class of value functions $\mathcal{V}_\Theta = \{V^{\pi_\theta} : \theta \in \Theta\}$ to be the smallest scalar such that $\forall V \in \mathcal{V}_\Theta$

$$||V - V^\star||_{1,\xi} \leq \frac{\kappa_\xi}{(1-\gamma)} ||V - TV||_{1,\xi}$$

where $||V(\boldsymbol{s})||_{1,\xi} = \mathbb{E}_{\boldsymbol{s}\sim\xi}[V(\boldsymbol{s})]$.

Assumption 5.6 (Differentiability)**.** *For each policy parameter θ and induced stationary distribution ν_θ, the functions $\bar{\theta} \mapsto \mathcal{B}(\bar{\theta}|\nu_\theta, V_\gamma^\theta)$ and $\bar{\theta} \mapsto \mathcal{B}(\theta|\nu_{\bar{\theta}}, V_\gamma^\theta)$ are continuously differentiable on an open set containing θ.*

Assumption 5.7 (Closure under approximate policy improvement)**.** *For each $\theta \in \Theta$, there exists $\epsilon_b \geq 0$ such that*

$$\min_{\theta^+\in\Theta} \mathcal{B}(\theta^+|\nu_\theta, V_\gamma^\theta) \leq \min_{\pi\in\Pi} \mathcal{B}(\pi|\nu_\theta, V_\gamma^\theta) + \epsilon_b$$

where Π is the set of all policies and ϵ_b is referred to as the inherent Bellman error of the policy class.

Assumption 5.8 (Stationary points of the weighted PI objective)**.** *For each $\bar{\theta} \in \Theta$, the function $\bar{\theta} \mapsto \mathcal{B}(\bar{\theta}|\nu_\theta, V_\gamma^\theta)$ has no sub-optimal stationary points.*

Assumption Verification. It is easy to see that Assumption 5.6 holds for our policy parameterization in Equation (5). Next, we prove that Assumption 5.7 holds for our model in Appendix A using monotonicity arguments. Further, we note that $\epsilon_b \to 0$ as the sharpness hyperparameter $\sigma \to \infty$. This agrees with Lemma 4 in Lin and Kumar (1984) which shows that the (hard) threshold policy is closed under policy improvement when there are only two servers. Finally, we verify Assumption 5.8 by showing that there are indeed no sub-optimal stationary points in Appendix B where we establish convexity of the weighted PI objective.

We now state the formal result of approximate optimality of the stationary point for the two server system in the discounted cost setting.

Theorem 5.2 (Bhandari and Russo (2024), Theorem 5)**.** *If assumptions 5.6-5.8 hold, then J_γ is continuously differentiable and any stationary point θ of $J_\gamma(\cdot)$ satisfies,*

$$J_\gamma(\pi_\theta) - J_\gamma(\pi^\star) \leq \frac{\kappa_\xi}{(1-\gamma)} \cdot \epsilon_b \tag{10}$$

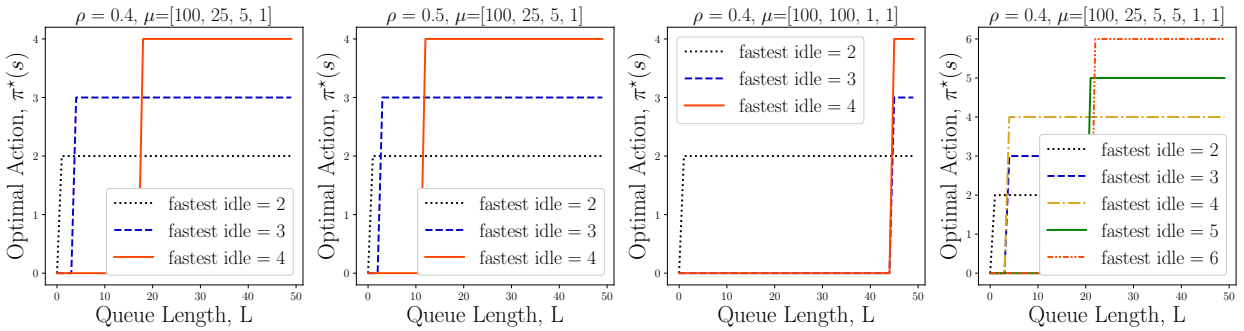
where κ_ξ is the effective concentrability coefficient and ϵ_b is the inherent Bellman error arising due to closure under approximate policy improvement.

6 EXPERIMENTS

In this section we show empirically that the optimal policy obtained via RVI is of threshold type even for the multi-server case and compare the performance of RVI and ACHQ against two baselines.

Simulation Setup. We simulate the queueing system as a discrete time system by sampling the continuous time model. If an idle server is imagined to be serving a fictitious job, sampling the continuous time system at instants of arrivals and departures (both true and fictitious) gives us an equivalent discrete time representation (Lippman, 1975). Note that the standard technique of including fictitious departures ensures that the sample instants are equally exponentially spaced out. We, hence, now have a discrete time system where in at each timestep, whose beginning is marked by an arrival or a departure, the policy decides which server to send a job waiting in the queue to or chooses to keep the job in the queue and wait for the next timestep. The job is then routed accordingly and a transition to the next state occurs as a result of an arrival whose probability is $\lambda/(\lambda + \sum_i \mu_i)$ or a departure whose probability is $\mu_i/(\lambda + \sum_i \mu_i)$.

Baselines. We compare against two baseline policies — fastest-available-server (FAS) and ratio-of-service-rate-thresholds (RSRT). FAS routes a job waiting in the queue to the fastest among the available servers at each timestep. RSRT is a threshold policy where a job is routed to the fastest among the available servers (say server f) only if the number of jobs waiting in the queue exceeds the threshold $\theta_f = \left(\sum_{i=1}^{f-1} \mu_i\right)/\mu_f$. The RSRT threshold values, proposed in Larsen (1981), represent the maximum number of jobs in the queue beyond which waiting for a faster server is detrimental in a very lightly loaded system ($\lambda \to 0$) where all servers $i < f$ are used. While FAS can be viewed as a pessimistic policy that always routes at the first



(a) $\boldsymbol{\mu} = [100, 25, 5, 1]$, $\rho = 0.4$ (b) $\boldsymbol{\mu} = [100, 25, 5, 1]$, $\rho = 0.5$ (c) $\boldsymbol{\mu} = [100, 100, 1, 1]$, $\rho = 0.4$ (d) $\boldsymbol{\mu} = [100, 25, 5, 5, 1, 1]$, $\rho = 0.4$

Figure 2: For multi-server heterogeneous systems, optimal policy is observed to be of threshold type where jobs are routed to the fastest available server only when the queue length exceeds the threshold

available opportunity, RSRT is an optimistic policy that believes in the benefit of waiting for a faster server. The goal of our algorithm, thus, is to find an optimal balance between optimism and pessimism.

6.1 Relative Value Iteration

In this subsection, we consider four configurations (a)-(d) of the heterogeneous queueing system with a buffer capacity $l_M = 100$ as examples. (a) is the base case where $\boldsymbol{\mu} = [100, 25, 5, 1]$, $\rho = 0.4$. We choose a higher load in (b) where $\boldsymbol{\mu} = [100, 25, 5, 1]$, $\rho = 0.5$. We increase the degree of heterogeneity in (c) and pick $\boldsymbol{\mu} = [100, 100, 1, 1]$, $\rho = 0.4$. We vary the number of servers in (d) with $\boldsymbol{\mu} = [100, 25, 5, 5, 1, 1]$, $\rho = 0.4$. Note that we define load of the system as $\rho = \lambda/(\sum_i \mu_i)$.

Threshold Policy. Recall that a threshold policy is a rule where a job is sent to the fastest among the available servers (say server f) when the number of jobs waiting in the queue is above a threshold θ_f and otherwise waits for the next timestep. In Figure 2, we plot the optimal action against the number of jobs waiting in the queue for different states of server occupation. Note that $a = f$ represents the job being routed to the fastest available server f and $a = 0$ denotes waiting for the next timestep with the job remaining in the queue. We observe that the optimal policy is indeed of threshold type. Further, an important observation we make is that the faster the service rate of a server, the smaller is the threshold corresponding to it. We notice from (a) and (c) that thresholds of a server are affected by servers faster than it. Moreover, we see from (a) and (d) that the servers with the same service rate and same set of faster servers have the same threshold despite the number and rates of the slower servers. We also remark that the thresholds depend on load of the system from (a) and (b).

	RVI	FAS	RSRT
(a)	5.48 ± 0.03	7.72 ± 0.04	10.04 ± 0.05
(b)	8.11 ± 0.04	9.72 ± 0.05	17.15 ± 0.08
(c)	2.36 ± 0.01	4.64 ± 0.03	2.37 ± 0.01
(d)	5.46 ± 0.03	9.56 ± 0.04	10.45 ± 0.06

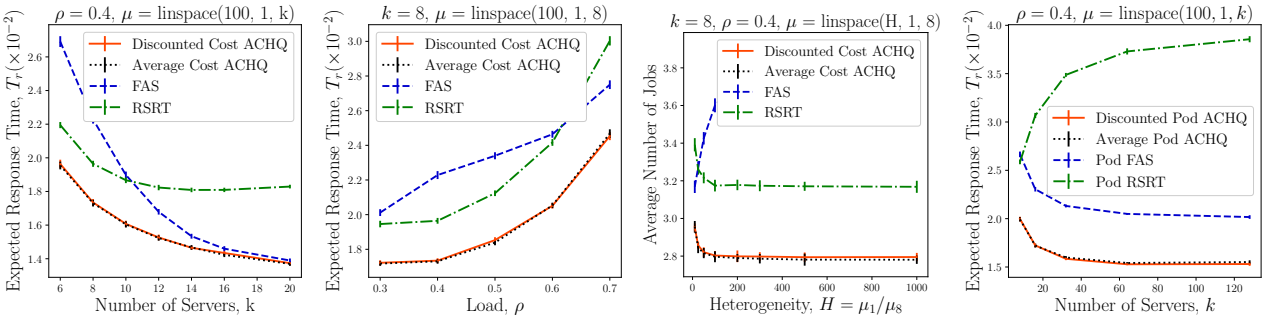
Table 1: Expected Response Time, T_r ($\times 10^{-2}$)

Performance Improvement. We observe in Table 1 that the optimal policy found by Relative Value Iteration (RVI) improves the expected response time by up to $\sim 50\%$ over the FAS baseline. On the other hand, while RVI consistently outperforms RSRT, the amount of gain is highly dependent on the instance.

Value Function Approximation. On approximating the value function $V(\boldsymbol{s})$ obtained for the configurations (a) - (d) with a linear function, we obtain R^2 values of $0.941, 0.943, 0.942, 0.942$ respectively. This indicates a very high degree of correlation and hence shows that a linear function is a good approximation.

6.2 ACHQ

Here, we consider a representative example of 8 servers whose service rates are linearly spaced between 100 and 1 with a load $\rho = \lambda/(\sum_i \mu_i) = 0.4$ and compare the performance with varying number of server, heterogeneity of service rates and load. We set the sharpness hyperparameter $\sigma = 1$ and the learning rates for actor $\alpha = 10^{-3}$, critic $\beta = 10^{-3}$ and average cost estimator $\zeta = 10^{-2}$. We observe in Figure 4, that the average number of jobs in the system and the threshold values converge during learning. We see that both the average cost and discount cost versions of the algorithm perform similarly. Note that by Little's Law, the average number of jobs in the system, n, and expected response time, T_r, are related as $n = \lambda T_r$.



(a) Number of Servers (b) Load (c) Heterogeneity (d) Pod

Figure 3: ACHQ shows up to $\sim 30\%$ improvement over the FAS and RSRT baselines

With an increase in the *number of servers*, we observe in Figure 3a that while ACHQ outperforms both the baselines FAS and RSRT, the gap to FAS reduces and gap to RSRT increases. We attribute this to the fact that a moderately fast server is often available as the system scales up. As the *load* increases in Figure 3b, we notice that the gain of ACHQ over FAS increases initially and then decreases. This can be explained by the fact that at high loads even the slower servers are required and at low loads the slower servers are used sparingly. Only at medium loads do we observe non-trivial thresholds and usage of slower servers. With an increase in *heterogeneity* in Figure 3c, we observe an increased performance gain of ACHQ over FAS. This is because FAS ignores the option of waiting for a increasingly faster server as the heterogeneity increases.

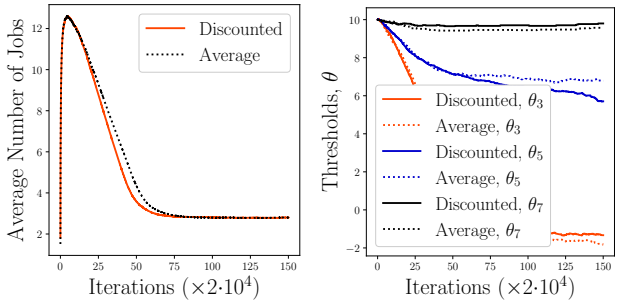


Figure 4: Convergence of ACHQ: Instance of 8 servers with $\boldsymbol{\mu} = \text{linspace}(100, 1)$ and load $\rho = 0.4$

Checking the availability of each server at every timestep becomes expensive as the number of servers in the system scale up. In real world large scale systems, a router often samples a subset of d servers and makes a routing decision based on these. In homogeneous systems, these Power-of-d-Choice (Pod) policies have been shown to be asymptotically optimal (Mukherjee et al., 2020). We implement a power of 4 choices version of ACHQ where at each timestep, 4 servers are sampled without replacement and a job is sent to the fastest among these 4 servers if the number of jobs in the queue exceeds its threshold. As shown in Figure 3d, our algorithm continues to outperform FAS and RSRT. Note that unlike in the case where we are allowed to observe the state of all servers, the gap between FAS and ACHQ does not decrease.

Remark. While we consider that the rates are known in this work, ACHQ can easily be extended to the case of unknown arrival and/or service rates by incorporating a simple upper confidence bound based estimation step.

7 CONCLUSION

We consider the open problem of routing in heterogeneous multi-server queueing systems. We propose ACHQ, an efficient policy gradient based algorithm where we leverage the underlying queueing structure by designing a low dimensional soft threshold parameterized policy. We provide stationary-point convergence guarantees for the general case and convergence to an approximate global optimum for the special case of two servers. We also demonstrate an improvement in the expected response time of up to $\sim 30\%$ over the greedy policy of routing to the fastest available server. Directions of future work include proving that the optimal policy is indeed of threshold type even in the multi-server case and expanding to other distributions of arrival and service rates.

Acknowledgements

This work was supported in part by the CMU Dean's Fellowship, Navneet Foundation Endowed Fellowship, CMU CyLab Seed Fund, C3 AI Institute Fund, and NSF grants 2154171, 2339112, ECCS-2145713, CCF-2045694, CNS-2112471, CPS-2111751, ONR-N00014-23-1-2149.

References

Agarwal, A., Kakade, S. M., Lee, J. D., and Mahajan, G. (2021). On the theory of policy gradient methods: optimality, approximation, and distribution shift. *J. Mach. Learn. Res.*, 22(1).

Amazon (2023). Aws lambda. https://aws.amazon.com/lambda.

Ayesta, U. (2022). Reinforcement learning in queues. *Queueing Systems*, 100(3-4):497–499.

Bhandari, J. and Russo, D. (2024). Global optimality guarantees for policy gradient methods. *Operations Research*.

Bhandari, J., Russo, D., and Singal, R. (2018). A finite-time analysis of temporal difference learning with linear function approximation. In *Conference on learning theory*, pages 1691–1692. PMLR.

Dai, J. G. and Gluzman, M. (2022). Queueing network controls via deep reinforcement learning. *Stochastic Systems*, 12(1):30–67.

De Vericourt, F. and Yong-Pin, Z. (2006). On the incomplete results for the heterogeneous server problem. *Queueing Systems*, 52(3):189.

DeepMind, G. (2023). Deepmind ai reduces google data centre cooling bill by 40%.

Google (2023a). Cloud tensor processing units. https://cloud.google.com/tpu.

Google (2023b). Gpu platforms. https://cloud.google.com/compute/docs/gpus.

Harchol-Balter, M. (2013). *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press.

Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in neural information processing systems*, 12.

Koole, G. (1995). A simple proof of the optimality of a threshold policy in a two-server queueing system. *Systems and Control Letters*, 26(5):301–303.

Koole, G. (2022). The slow-server problem with multiple slow servers. *Queueing Systems*, 100:469–471.

Kumar, H., Koppel, A., and Ribeiro, A. (2023). On the sample complexity of actor-critic method for reinforcement learning with function approximation. *Machine Learning*, pages 1–35.

Larsen, R. L. (1981). Control of multiple exponential servers with application to computer systems.

Lin, W. and Kumar, P. (1984). Optimal control of a queueing system with two heterogeneous servers. *IEEE Transactions on Automatic Control*, 29(8):696–703.

Lippman, S. A. (1975). Applying a new device in the optimization of exponential queuing systems. *Operations Research*, 23(4):687–710.

Liu, B., Xie, Q., and Modiano, E. (2022). Rl-qn: A reinforcement learning framework for optimal control of queueing systems. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems*, 7(1):1–35.

Luh, H. P. and Viniotis, I. (2002). Threshold control policies for heterogeneous server systems. *Mathematical Methods of Operations Research*, 55:121–142.

Mannor, S., Rubinstein, R. Y., and Gat, Y. (2003). The cross entropy method for fast policy search. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 512–519.

Mao, H., Schwarzkopf, M., Venkatakrishnan, S. B., Meng, Z., and Alizadeh, M. (2019). Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, page 270–288.

Moallemi, C. C., Kumar, S., and Van Roy, B. (2008). Approximate and data-driven dynamic programming for queueing networks.

Mukherjee, D., Borst, S. C., Van Leeuwaarden, J. S., and Whiting, P. A. (2020). Asymptotic optimality of power-of-d load balancing in large-scale systems. *Mathematics of Operations Research*, 45(4):1535–1571.

Robledo, F., Borkar, V., Ayesta, U., and Avrachenkov, K. (2022). Qwi: Q-learning with whittle index. *ACM SIGMETRICS Performance Evaluation Review*, 49(2):47–50.

Rykov, V. V. (2001). Monotone control of queueing systems with heterogeneous servers. *Queueing systems*, 37(4):391–403.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*.

Srikant, R. and Ying, L. (2013). *Communication networks: an optimization, control, and stochastic networks perspective*. Cambridge University Press.

Staffolani, A., Darvariu, V.-A., Bellavista, P., and Musolesi, M. (2023). Rlq: Workload allocation with reinforcement learning in distributed queues. *IEEE Transactions on Parallel and Distributed Systems*, 34(3):856–868.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.

Tsitsiklis, J. N. and Xu, K. (2017). Flexible queueing architectures. *Operations Research*, 65(5):1398–1413.

van Barneveld, T., Jagtenberg, C., Bhulai, S., and van der Mei, R. (2018). Real-time ambulance relocation: Assessing real-time redeployment strategies for ambulance relocation. *Socio-Economic Planning Sciences*, 62:129–142.

Walrand, J. (1984). A note on "optimal control of a queuing system with two heterogeneous servers". *Systems and Control Letters*, 4(3):131–134.

Walton, N. and Xu, K. (2021). Learning and information in stochastic networks and queues. In *Tutorials in Operations Research: Emerging Optimization Methods and Modeling Techniques with Applications*, pages 161–198. INFORMS.

White, D. J. (1963). Dynamic programming, markov chains, and the method of successive approximations. *J. Math. Anal. Appl*, 6(3):373–376.

Wu, Y. F., Zhang, W., Xu, P., and Gu, Q. (2020). A finite-time analysis of two time-scale actor-critic methods. *Advances in Neural Information Processing Systems*, 33:17617–17628.

Zhang, S., Zhang, Z., and Maguluri, S. T. (2021). Finite sample analysis of average-reward td learning and q-learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 1230–1242.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Not Applicable]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Not Applicable]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Not Applicable]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

A CLOSURE UNDER APPROXIMATE POLICY IMPROVEMENT

In this section, we verify that Assumption 5.7 holds for our problem by presenting a proof of closure of the soft threshold policy under approximate policy improvement. This follows the arguments made in Lemma 4 of Lin and Kumar (1984) which shows that (hard) threshold policies are closed under policy improvement for the two server case.

Recall that the queue is represented as a discrete time system where at each timestep, whose beginning is marked by an arrival or a departure, the router decides if and which server to send a job waiting in the queue to. We consider without loss of generality that the arrival and service rates are normalized as $\lambda + \sum_r \mu_i = 1$. The soft threshold policy parameterized by $\theta \in \Theta$, can be written as

$$\begin{aligned}
&\pi_\theta(a = 0|\boldsymbol{s} = (l, 1, 1)) = 1 \\
&\pi_\theta(a = 1|\boldsymbol{s} = (l, 0, 1)) = 1 \quad \text{if } l > 0 \\
&\pi_\theta(a = 0|\boldsymbol{s} = (l, 0, 1)) = 1 \quad \text{if } l = 0 \\
&\pi_\theta(a = 2|\boldsymbol{s} = (l, 1, 0)) = \frac{e^{\sigma(l-\theta)}}{1 + e^{\sigma(l-\theta)}} \\
&\pi_\theta(a = 0|\boldsymbol{s} = (l, 1, 0)) = \frac{1}{1 + e^{\sigma(l-\theta)}} \\
&\pi_\theta(a = 1|\boldsymbol{s} = (l, 0, 0)) = 1 \quad \text{if } l > 0 \\
&\pi_\theta(a = 0|\boldsymbol{s} = (l, 0, 0)) = 1 \quad \text{if } l = 0
\end{aligned} \tag{11}$$

where $\pi_\theta(a|\boldsymbol{s})$ represents the probability that a router following policy π_θ takes action a.

For a given soft threshold policy π_θ, represent the policy obtained by policy improvement over the set of all policies as $\pi' = \arg\min_{\pi \in \Pi} \mathcal{B}(\pi|\nu_\theta, V_\gamma^\theta)$. Similarly, denote the policy obtained by policy improvement over the set of soft threshold policies as $\theta' = \arg\min_{\theta^+ \in \Theta} \mathcal{B}(\theta^+|\nu_\theta, V_\gamma^\theta)$. To verify Assumption 5.7 for our problem, we want to show that the weighted PI objective of θ' is approximately equal to that of π'. Formally, we want to show that there exists a $\theta' \in \Theta$ for every soft threshold policy $\theta \in \Theta$ such that

$$\mathcal{B}(\theta'|\nu_\theta, V_\gamma^\theta) \leq \mathcal{B}(\pi'|\nu_\theta, V_\gamma^\theta) + \epsilon_b$$

where ϵ_b is referred to as the inherent Bellman error.

We first consider the improvement over the set of all policies π'. Define

$$\begin{aligned}
h_0 &= V_\gamma^\theta(0, 1, 0) - V_\gamma^\theta(0, 0, 1) \\
h_l &= V_\gamma^\theta(l, 1, 0) - V_\gamma^\theta(l - 1, 1, 1) \quad \text{for } l \geq 1.
\end{aligned} \tag{12}$$

For state $\boldsymbol{s} = (l, 1, 0)$, notice that $h_l < 0$ implies that the optimal action under π' is to wait until the next timestep and $h_l \geq 0$ implies that the optimal action is to route a job to the slow server. If $h_l < 0$ for $l \leq l^\star$ and $h_l \geq 0$ for $l > l^\star$, then the improved policy π' is a threshold policy where a job is routed to the slow server only when the number of jobs in the queue exceeds the threshold $l^\star$ (see Lemma 4, Lin and Kumar (1984)).

Now, if we pick $\theta' = l^\star$, the weighted PI objective of improved soft threshold policy θ' will approximately be equal to that of π' with ϵ_b characterizing the error. The Bellman approximation error ϵ_b arises due to the difference between π' being a (hard) threshold policy and θ' being a soft threshold policy. Note that $\epsilon_b \to 0$ as the decision sharpness hyperparameter $\sigma \to \infty$ and the soft threshold tends towards a hard threshold.

We dedicate the rest of the section to proving that the structure required over h_l where $h_l < 0$ for $l \leq l^\star$ and $h_l \geq 0$ for $l > l^\star$ is indeed true using monotonicity arguments.

Applying Bellman equation to $V_\gamma^\pi(l,1,0)$ and $V_\gamma^\pi(l-1,1,1)$, we have

$$V_\gamma^\pi(l,1,0) = (l+1) + \gamma\lambda\left[\frac{e^{\sigma(l+1-\theta)}}{1+e^{\sigma(l+1-\theta)}}V_\gamma^\pi(l,1,1) + \frac{1}{1+e^{\sigma(l+1-\theta)}}V_\gamma^\pi(l+1,1,0)\right] + \gamma\mu_1 V_\gamma^\pi(l-1,1,0) + \gamma\mu_2\left[\frac{e^{\sigma(l-\theta)}}{1+e^{\sigma(l-\theta)}}V_\gamma^\pi(l-1,1,1) + \frac{1}{1+e^{\sigma(l-\theta)}}V_\gamma^\pi(l,1,0)\right], \tag{13}$$

$$V_\gamma^\pi(l-1,1,1) = (l+1) + \gamma\lambda V_\gamma^\pi(l,1,1) + \gamma\mu_1 V_\gamma^\pi(l-2,1,1) + \gamma\mu_2\left[\frac{e^{\sigma(l-1-\theta)}}{1+e^{\sigma(l-1-\theta)}}V_\gamma^\pi(l-2,1,1) + \frac{1}{1+e^{\sigma(l-1-\theta)}}V_\gamma^\pi(l-1,1,0)\right]. \tag{14}$$

Subtracting the two, we get,

$$\begin{aligned} h_l &= V_\gamma^\pi(l,1,0) - V_\gamma^\pi(l-1,1,1) \\ &= \gamma\lambda\left[\frac{V_\gamma^\pi(l+1,1,0) - V_\gamma^\pi(l,1,1)}{1+e^{\sigma(l+1-\theta)}}\right] + \gamma\mu_1\left[V_\gamma^\pi(l-1,1,0) - V_\gamma^\pi(l-2,1,1)\right] \\ &+ \gamma\mu_2\Bigg[\frac{e^{\sigma(l-\theta)}}{1+e^{\sigma(l-\theta)}}V_\gamma^\pi(l-1,1,1) + \frac{1}{1+e^{\sigma(l-\theta)}}V_\gamma^\pi(l,1,0) \\ &- \frac{e^{\sigma(l-1-\theta)}}{1+e^{\sigma(l-1-\theta)}}V_\gamma^\pi(l-2,1,1) - \frac{1}{1+e^{\sigma(l-1-\theta)}}V_\gamma^\pi(l-1,1,0)\Bigg]. \end{aligned} \tag{15}$$

Now, we argue that for a large enough sharpness hyperparameter σ, there exists an $l_1 > 0$ such that $h_l \geq 0$ for all $l > l_1$ and denote the smallest such l_1 as $l^\star$. This can be seen by the fact that terms 1 and 3 corresponding to a departure from the second server dominate the equation at large queue lengths and we know

$$\frac{e^{\sigma(l-\theta)}}{1+e^{\sigma(l-\theta)}}V_\gamma^\pi(l-1,1,1) \geq \frac{e^{\sigma(l-1-\theta)}}{1+e^{\sigma(l-1-\theta)}}V_\gamma^\pi(l-2,1,1).$$

This is because $V_\gamma^\pi(l-1,1,1) \geq V_\gamma^\pi(l-2,1,1)$ following arguments of monotonicity of Bellman operator as in Lemma 1 of Lin and Kumar (1984).

With some algebraic manipulation, we obtain,

$$\begin{aligned} (1-b\gamma)h_l &\geq b_1\gamma\lambda(h_{l+1} - h_l) + b_2\gamma\mu_1(h_{l-1} - h_l) \\ &+ \frac{\gamma\mu_2}{1+e^{\sigma(l-1-\theta)}}\left[\frac{1+e^{\sigma(l-1-\theta)}}{1+e^{\sigma(l-\theta)}}V_\gamma^\pi(l-1,1,1) - V_\gamma^\pi(l-1,1,0)\right] \end{aligned} \tag{16}$$

where $b_1 = 1/(1+e^{\sigma(l+1\theta)})$, $b_2 = 1$, $b_3 = 1/(1+e^{\sigma(l-\theta)})$ and $b = b_1\lambda + b_2\mu_1 + b_3\mu_2$. For a large enough sharpness constant σ, the third term on the right hand side in the expression above is always positive. We hence have

$$-(1-a\gamma)h_l + \gamma\lambda(h_{l+1} - h_l) \leq \gamma\mu_1(h_l - h_{l-1}). \tag{17}$$

We know $h_{l^\star+1} \geq 0$ and $h_{l^\star} < 0$. Using $l = l^\star, l^\star - 1, \ldots$ in Equation (17), it follows that

$$h_0 < h_1 < \ldots h_{l^\star} < 0. \tag{18}$$

Putting Equation (15) and Equation (18) together, we have $h_0 < h_1 < \ldots h_{l^\star} < 0$ and $h_l \geq 0$ for all $l > l^\star$. The improved policy π' is thus a threshold policy with threshold $l^\star$. Choosing $\theta' = l^\star$ results in approximate equality of the weighted PI objectives as detailed above and concludes the proof.

B STATIONARY POINTS OF WEIGHTED PI OBJECTIVE

In this section, we show that the weighted PI objective has no sub-optimal stationary points by arguing that it is convex. Let $l^\star$ be such that $Q^\pi_\gamma((l,1,0),a=2) < Q^\pi_\gamma((l,1,0),a=0)$ for all $l \geq l^\star$. The existence of such an $l^\star$ is guaranteed by Equation (15) in Appendix A above. Now, consider the value of the weighted PI objective $\mathcal{B}(\bar{\theta}|\nu_\theta, V^\theta_\gamma)$ with increasing values $\bar{\theta}$. As $\bar{\theta}$ increases towards $l^\star$, the number of states where the sub-optimal action of routing a job to the slow server reduces and hence the weighted PI objective monotonically reduces. The minimum is obtained at $\bar{\theta} = l^\star$. Further, as $\bar{\theta}$ increases beyond $l^\star$, the number of states where the sub-optimal action of idling until the next timestep is chosen more often resulting in a monotonic increase in the weighted PI objective. Thus, the weighted PI objective is a convex function in $\bar{\theta}$ and there are no sub-optimal stationary points.

C RELATIVE VALUE ITERATION

We detail the Relative Value Iteration algorithm below for the sake for completeness. Note that the span semi-norm used in the algorithm is defined as $sp(f(x)) := [\max_x f(x)][\min_x f(x)]$.

Algorithm 2 Relative Value Iteration

Initialize $V^{(0)}(\boldsymbol{s}) = 0\ \forall \boldsymbol{s} \in \mathcal{S}$, ϵ
while $sp(V^{(t)} - V^{(t-1)}) > \epsilon$ **do**
  for $\boldsymbol{s} \in \mathcal{S}$ **do**
    $V^{(t+1)}(\boldsymbol{s}) \leftarrow T(V^{(t)})(\boldsymbol{s}) - T(V^{(t)})(\boldsymbol{s}^\star)$
  end for
  $t \leftarrow t+1$
end while
$\pi(\boldsymbol{s}) = \arg\min_a c(\boldsymbol{s}) + \sum_{\boldsymbol{s}' \in \mathcal{S}} \Pr(\boldsymbol{s}'|\boldsymbol{s}, a) V^{(t)}(\boldsymbol{s}')$