Optimizing Latency in Inference Systems with Accuracy Constraints

ANONYMOUS AUTHOR(S)

Amidst the rapid advancements in Large Language Models (LLMs), optimizing the latency performance of online machine learning inference systems becomes paramount. Unlike standard queueing models that only analyze latency, inference systems require accurate output to meet certain benchmarks. In this paper, we consider such a queueing system with the goal of minimizing latency under the constraint that the average output accuracy is greater than a benchmark value a^*, which has not been previously studied to the best of our knowledge. We first identify a lower bound on the minimum achievable latency under any policy that achieves the benchmark accuracy a^* using a linear programming (LP) formulation. Building on the LP solution, we introduce the R-JIQ policy, which consistently meets the accuracy benchmark and asymptotically (as system size increases) achieves the optimal latency $T_{\text{LP-LB}}(\lambda)$. However, the R-JIQ policy relies on the knowledge of the arrival rate λ to solve the LP. To address this limitation, we propose the Track the Accuracy Difference (TAD) policy, which meets the accuracy constraint without relying on the arrival rate. While TAD performs well empirically, it does not always achieve asymptotically optimal latency. As a refinement to TAD, we present the TAD-OP policy that incorporates the concept of *ordered pairs* of servers into waterfilling to iteratively solve the LP. Experiments suggest that TAD-OP performs robustly across different system sizes and load scenarios, approaching near-optimal asymptotic performance.

1 INTRODUCTION

Machine learning (ML) has become ubiquitous in daily life over the last decade, whose growth is particularly amplified with the advent of transformers [25] and the emergence of Large Language Models (LLMs). Applications such as ChatGPT and Copilot have rapidly transformed into everyday tools, aiding users in communication and coding tasks, respectively. However, the growth of ML popularity is closely followed by a rapid increase in model sizes, resulting in increased inference times. For perspective, AlexNet [12], an early ML model, contains 62.3 million parameters while GPT-3 [3] boasts a staggering 175 billion parameters. This growing demand for ML applications, coupled with the exponential growth in model sizes, underscores the urgent need for efficiently designed computing systems to support them.

In this paper, we introduce a model for a computing system specifically tailored for serving ML inference queries, which we refer to as an *inference system*. To capture the performance of such a system, we focus on the fundamental tradeoff between accuracy and latency. Specifically, larger ML models yield higher accuracy but incur greater latency, whereas smaller models offer reduced latency at the expense of accuracy. In such a system, we aim to devise policies for assigning arriving inference queries to servers, with the performance goal of minimizing latency while achieving a desired level of accuracy.

Our approach complements the extensive efforts within the ML community to improve accuracy along with increased efficiency. Notable techniques for this purpose include knowledge distillation [8, 19], pruning [14], and early exits [2], just to name a few.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

, ,

© 2023 Association for Computing Machinery.

There have also been a lot of efforts on system-level optimizations for improving the performance of ML systems. For instance, *JellyBean* [30] optimizes ML inference workflows on heterogeneous infrastructures but lacks latency guarantees. *Clipper* [4] improves throughput, latency, and accuracy using caching, adaptive batching, and bandit-based model selection, but it does not fully capture the latency–accuracy tradeoff. INFaaS [20] offers a model-less system for distributed inference with user-defined performance metrics, distinct from our system-set criteria. Some other optimizations are explored in [18, 21, 23, 31]. However, these methods lack a theoretical framework to understand the fundamental latency–accuracy tradeoff.

In this paper, we approach the problem from a queueing system perspective. Traditional queueing theory has been instrumental in analyzing system latency but often overlooks accuracy considerations. For instance, classical policies such as Join-the-Shortest-Queue (JSQ) [6, 29], Join-the-Idle-Queue (JIQ) [15, 16, 24], and their low-overhead alternatives, JSQ-d [17, 26] and JIQ-d [27], all focus on minimizing latency in load-balancing systems with homogeneous servers. The Join-the-Fastest-of-the-Shortest-Queue (JFSQ) policy [28] is shown to be asymptotically optimal in latency with heterogeneous servers. A comprehensive survey is provided in [5].

There has also been a large body of work on maximizing network utility [9, 10, 13]. Although we can potentially model accuracy as a utility, existing work is more concerned with throughput maximization. Similarly, studies on scheduling under deadline constraints [11, 22] aim at minimizing job expiration or maximizing revenue rather than balancing latency and accuracy.

To demonstrate how accuracy considerations affect the query assignment problem, let us consider a load-balancing system where each server is associated with both a service rate and an accuracy level. When an ML query arrives in the system, we assign the query to one of the servers immediately. Since our goal is to minimize latency while achieving a desired level of accuracy, there are two intuitive job assignment policies: JIQ that prioritizes faster servers, and JIQ that prioritizes higher accuracy servers. However, Figure 1 reveals that prioritizing faster servers does not achieve the desired level of accuracy (shown as the blue line), whereas prioritizing higher accuracies results in large latency. In contrast, we plot the performance of one of our proposed policies, which successfully strikes a balance, achieving the desired level of accuracy while maintaining reasonable latency.

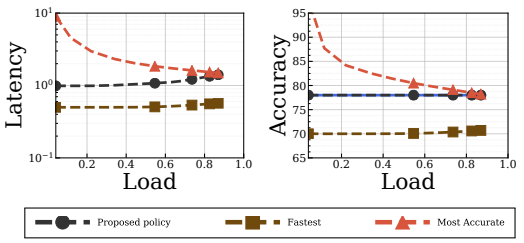

Fig. 1. Latency–accuracy tradeoff.

1.1 Main Contribution

Modeling. We represent the inference system using a queueing model that aims to minimize latency while ensuring the system achieves an average benchmark accuracy output a^*. To the best of our knowledge, this is the first work in modeling an inference system that captures the trade-off between latency and accuracy.

Lower Bound and Baseline Approach. We present a lower bound on the minimum achievable latency for any policy that satisfies the accuracy constraint using a linear program (LP). By merging existing queueing policies that minimize waiting time with the LP solution, we introduce the R-JIQ algorithm. This algorithm consistently meets the benchmark accuracy and demonstrates asymptotic latency optimality as the system scales.

Arrival-rate Oblivious Policies. While R-JIQ is asymptotically optimal, it relies on an accurate knowledge of the arrival rate and is vulnerable to any discrepancies in this estimation. This led us to develop the Track-the-Accuracy Difference (TAD) policy, a heuristic approach that satisfies the accuracy constraint without using the knowledge of the arrival rate. However, it does not achieve optimal latency. We then present the TAD-OP policy that emulates a waterfilling algorithm using order pairs of server classes. Experimental results demonstrate a robust performance of this policy across various system loads and sizes, achieving nearing optimal performance in large systems.

The Concept of Pairs. We introduce the idea of ordered pairs akin to priority classes in the literature. The pairs can balance the trade-off between latency and accuracy, something simple priority classes fail to achieve. While our motivation lies in modeling the inference system, the pair concept has broader applicability to any system with linear constraints, e.g., inference systems constrained by energy budgets. Further, for systems with more than one constraint, pairs motivate the extension to triplets or even larger groupings.

1.2 Paper Outline

We present the problem formulation and performance metrics in Section 2. Following this, Section 3.1 presents a lower bound on the minimum achievable latency for any system that maintains the benchmark accuracy using an LP defined in (3). The R-JIQ policy is introduced in Section 3.2, with its asymptotic optimal guarantees highlighted in Lemma 2. The potential limitations of the R-JIQ policy are discussed in Section 3.4. We introduce the TAD policy and the reason for its sub-optimal performance in Section 4. We introduce the concept of pairs in Section 5.1, which subsequently leads to the waterfilling algorithm, described in Section 5.3. Theorem 1 provides the convergence guarantee of the waterfilling algorithm to the solution of the LP. We present our main proposed TAD-OP policy in Section 5.4. We validate our policies with experimental results in Section 7 and conclude in Section 8.

2 PROBLEM FORMULATION

2.1 System model; Arrival, Departure, and Accuracy

We consider an online ML inference system with n servers, where each server stores one ML model. The n servers are divided into K distinct classes corresponding to K different ML models, with $\alpha_i > 0$ fraction of the total servers allocated to the i-th class such that $\sum_{i=1}^{K} \alpha_i = 1$. Inference queries arrive in the system according to a Poisson process with the rate $\Lambda = \lambda n$, where λ is the normalized arrival rate. An inference query can be interpreted as a batch of inference tasks that require execution on an ML model stored within the system. Upon arrival, we route the query to one of the servers according to a designated policy. Queries wait in a queue at the server according to a first-come-first-serve (FCFS) manner until they are served. An inference query on a server of class k requires an exponentially distributed random time with a rate μ_k independent of everything else. Upon being served from a server of class-k, the query receives an accuracy score of a_k. We assume that the service rates and the accuracies are ordered, i.e., $\mu_1 > \mu_2 > \cdots > \mu_K$ and $a_1 < a_2 < \cdots < a_K$. The assumption aligns with the properties of ML models; a larger ML model typically yields higher accuracy scores but

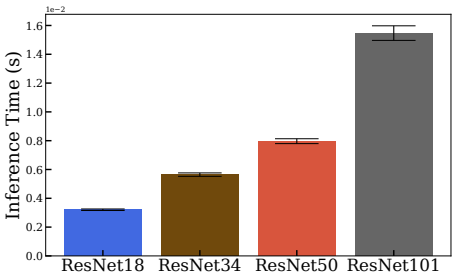

Fig. 2. Batch Inference time for different ResNet models accelerated using NVIDIA TitanX GPU

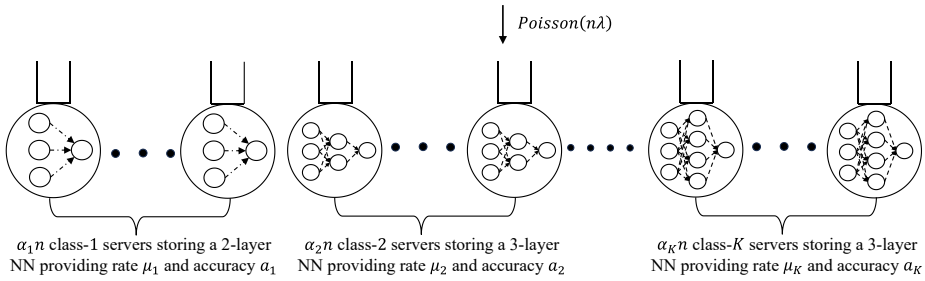

Fig. 3. An illustration of the inference system with n servers divided into K distinct classes. Jobs arrive according to a Poisson process with a rate $\Lambda = n\lambda$. Each server stores an ML model. Class-1 servers store the smallest model, providing the least accuracy a_1 and maximum speed μ_1, while class-K servers store the largest ML model, providing maximum accuracy and minimum latency.

requires longer processing times due to increased computational requirements, see Fig. 2. Fig. 3 provides an illustration of the inference system.

2.2 Performance Metric

A way to measure the performance of an inference system is to understand the user delay and the accuracy output. We measure the user delay using the *expected response time* of queries in the system. The response time of a query is the total time that a query spends in the system, which is the sum of the waiting time in the queue and the service time. Meanwhile, we measure the accuracy output using the *expected accuracy* of queries served by the inference system. For any policy π that determines a way to assign queries to servers, $\mathbb{E}[T_\pi]$ and $\mathbb{E}[a_\pi]$ denote the expected response time and expected accuracy of the system under the policy, respectively.

A conventional approach to designing policy π for queueing systems aims to minimize $\mathbb{E}[T_\pi]$. However, within the context of an inference system, prioritizing low latency could inadvertently lead to decreased $\mathbb{E}[a_\pi]$ since latency-efficient policies often favor faster but less accurate servers. Conversely, prioritizing servers with only high accuracy increases response times. We handle this trade-off using a *benchmark accuracy* level, denoted by a^*, representing the minimum required average accuracy of the system. A *feasible* policy, in this context, is defined as a policy π where $\mathbb{E}[a_\pi] \geq a^*$. Based on these definitions, this paper aims to design a policy π^* that is feasible and minimizes the expected response time.

2.3 Service Capacity Region

In this section, we study the service capacity region of the inference system. We define the *service capacity region* as the set of arrival rates such that for any λ in the interior of the set, there exists a policy that achieves the accuracy criteria and stabilizes the queue lengths. Without the accuracy constraint, the maximum traffic a queueing system can handle is restricted by the sum of service rates of all servers. Specifically, the system can handle any normalized arrival rate $\lambda < \sum_{i=1}^{K} \alpha_i \mu_i$. However, introducing the accuracy constraint changes the capacity region as most of the system's capacity comes from the faster servers, which, unfortunately, provides lower accuracy. We define $\lambda^{\max}$ in Definition 1 to describe the capacity region of our inference system under the accuracy constraint. The inference system can only be feasible if λ is less than $\lambda^{\max}$ with a higher λ indicating a greater difficulty in maintaining the benchmark accuracy a^*. For technical discussion, we use

vector notation to keep it concise. Define the vectors

$$\boldsymbol{\mu} = (\mu_1, \ldots, \mu_K), \boldsymbol{\mu}^{-1} = \left(\frac{1}{\mu_1}, \ldots, \frac{1}{\mu_K}\right), \boldsymbol{\mu_\alpha} = (\alpha_1\mu_1, \ldots, \alpha_K\mu_K), \boldsymbol{a} = (a_1, \ldots, a_K) \tag{1}$$

with $\mathbf{0}, \mathbf{1}$ representing vectors with all zeros and ones, respectively and $\boldsymbol{e}_i$ representing vectors with all zeros, except the i-th index which contains 1. We use "$\cdot$" to represent the vector inner product.

Definition 1. *The maximum normalized arrival rate that the inference system can serve while achieving the benchmark accuracy a^* is given by*

$$\lambda^{\max} = \sup\left\{\lambda' : \exists \boldsymbol{p} \in \mathbb{R}^K \text{ s.t. } \lambda\boldsymbol{p} \le \boldsymbol{\mu_\alpha}, \boldsymbol{p} \ge \mathbf{0}, \boldsymbol{p}\cdot\mathbf{1} = 1, \boldsymbol{p}\cdot\boldsymbol{a} \ge a^*\right\}. \tag{2}$$

The definition of $\lambda^{\max}$ is simple. Any stable policy yields a probability vector $\boldsymbol{p} = (p_1, \ldots, p_K)$, with p_i denoting the steady-state fraction of queries served using a class-i server. Since $\boldsymbol{p}$ is a probability vector, the vector must be non-negative, and the sum of its elements should equal one; hence, the constraints $\boldsymbol{p} \ge \mathbf{0}$, and $\boldsymbol{p}\cdot\mathbf{1} = 1$. For any i, the traffic served by class-i servers must not exceed the net capacity of class i servers, as indicated by $\lambda\boldsymbol{p} \le \boldsymbol{\mu_\alpha}$. Lastly, the system must maintain the average accuracy of a^* given by $\boldsymbol{p}\cdot\boldsymbol{a} \ge a^*$.

Since $\lambda^{\max}$ denotes the maximum feasible arrival rate, it is inherently bounded by the total system capacity, i.e., $\lambda^{\max} \le \sum_{i=1}^K \alpha_i\mu_i$. However, equality may not always hold. Consider an extreme case where $a^* = a_K$. In this scenario, using server classes other than class K would violate the accuracy constraint. Consequently, $\lambda^{\max} = \alpha_K\mu_K < \sum_{i=1}^K \alpha_i\mu_i$.

3 THE RANDOMIZED JOIN-THE-IDLE-QUEUE POLICY

In this section, we introduce our first algorithm, the Randomized Join-the-Idle-Queue (R-JIQ) policy, which is inspired from a lower-bound detailed in Section 3.1, We present Lemma 2 to show that the policy always achieves the benchmark accuracy a^*, and as n increases, it achieves the minimum expected response time. However, despite its asymptotic optimality, R-JIQ has its challenges. A primary concern, discussed in Section 3.4, is the requirement of knowing the arrival rate, λ.

3.1 A Lower Bound

This subsection presents Lemma 1, which establishes a lower bound on the expected response time of queries in our inference system under any policy that achieves the average accuracy a^*.

Lemma 1 (Lower Bound on Expected Response Time)**.** *For the inference system defined in Section 2 and a normalized arrival rate λ, a lower bound on the expected response time of the system under any feasible policy π is given by $T_{LP\text{-}LB}(\lambda)$, defined as*

$$\begin{aligned} T_{LP\text{-}LB}(\lambda) = \min_{\boldsymbol{p}\in\mathbb{R}^K} \quad & \boldsymbol{p}\cdot\boldsymbol{\mu}^{-1} \\ \text{s.t.} \quad & \lambda\boldsymbol{p} \le \boldsymbol{\mu_\alpha}, \boldsymbol{p} \ge \mathbf{0}, \boldsymbol{p}\cdot\mathbf{1} = 1, \boldsymbol{p}\cdot\boldsymbol{a} \ge a^*. \end{aligned} \tag{3}$$

where the acronym LP-LB stands for Linear Programming to estimate Lower Bound (LP-LB).

Proof of Lemma 1. Let $\boldsymbol{p} = (p_1, \ldots, p_K)$ be a probability vector, where p_k denotes the steady-state fraction of queries served using servers of type k. The constraints defined follow a similar logic as expressed in the definition of $\lambda^{\max}$. The expression $\boldsymbol{p}\cdot\boldsymbol{\mu}^{-1}$ represents the expected service time of the queries, effectively the expected response time minus the expected wait time. Thus, the optimal value $T_{\text{LP-LB}}(\lambda)$ to the Linear Program (LP) is a lower bound to the expected response time under any policy that maintains the average accuracy of a^* □

The definition of $\lambda^{\max}$ ensures that the LP, defined in (3), is feasible for any $\lambda \le \lambda^{\max}$. When the LP is feasible, we denote the optimal solution of the LP by $\boldsymbol{p}^*(\lambda) = (p_1^*(\lambda), \ldots, p_K^*(\lambda))$.

3.2 The R-JIQ Policy

The key recipe in designing a policy that ensures feasibility and minimization of expected response time is rooted in the solution of a linear programming problem, defined in (3). The first intuition provided by this LP is the appropriate allocation of incoming queries to different server classes, ensuring that class-i servers serve $p_i^*(\lambda)$ fraction of the total queries. Achieving this is straightforward, as we can probabilistically route $p_i^*(\lambda)$ fraction of incoming queries to class-i servers.

The second intuition provided by the LP is to minimize waiting time. In large queueing systems, zero waiting time is a common phenomenon. A well-known policy that achieves zero waiting time in a homogenous server system is the *Join-the-Idle Queue* (JIQ) policy [15, 16, 24], which routes incoming queries to idle servers whenever possible.

Based on the observations above, one would like to use the probability vector $\boldsymbol{p}^*(\lambda)$ to choose a class-i and then join an idle server within the class-i servers. However, exactly using $\boldsymbol{p}^*(\lambda)$ might be problematic, as the solution of the LP allows $\lambda p_i^*(\lambda) = \alpha_i \mu_i$ for some i, which can make the underlying Continuous-time Markov chain (CTMC) null recurrent. To handle the issue, we define the R-JIQ policy as follows. For any normalized arrival rate λ, we define the probability vector $\boldsymbol{p}^*_{\text{R-JIQ}}(\lambda) = (p^*_{\text{R-JIQ},1}(\lambda), \ldots, p^*_{\text{R-JIQ},K}(\lambda))$, defined as

$$\boldsymbol{p}^*_{\text{R-JIQ}}(\lambda) = \left(1 - \frac{1}{n^\gamma}\right) \boldsymbol{p}^*(\lambda) + \frac{1}{n^\gamma} \boldsymbol{p}^*(\lambda^{\max}), \tag{4}$$

where γ is a non-negative hyper-parameter. The roles of γ and $\boldsymbol{p}^*(\lambda^{\max})$ are to slightly adjust the probability vector $\boldsymbol{p}^*(\lambda)$ to guarantee zero waiting time, but without making significant enough changes that take $\boldsymbol{p}^*_{\text{R-JIQ}}(\lambda)$ far from $\boldsymbol{p}^*(\lambda)$. Upon the arrival of a query, the policy chooses server class-i with probability $p^*_{\text{R-JIQ},i}(\lambda)$ independently of other classes. Subsequently, the query is directed to one of the idle class-i servers. If there are no idle servers within the class, one server within the class is chosen uniformly at random. To determine idle servers within the system, a tokenization method can be implemented, where a server, upon becoming idle, sends a token to the central dispatcher, informing it about its availability.

Lemma 2 establishes the asymptotic (large enough n) optimality of the R-JIQ policy. It confirms that for any $\lambda < \lambda^{\max}$, as long as the system operates within the sub-Halfin-Whitt regime, the R-JIQ policy asymptotically achieves the minimal expected latency, $T_{\text{LP-LB}}(\lambda)$, while achieving the benchmark accuracy a^*. The feasibility guarantee of the policy is independent of the value of n, since $\boldsymbol{p}^*(\lambda) \cdot \boldsymbol{a} \geq a^*$ and $\boldsymbol{p}^*(\lambda^{\max}) \cdot \boldsymbol{a} \geq a^*$. While the algorithm is well defined for any value n, it may not achieve the expected response time of $T_{\text{LP-LB}}(\lambda)$ for small n. This limitation stems from the non-tight lower bound $T_{\text{LP-LB}}(\lambda)$; zero waiting time is difficult to achieve in smaller systems.

Lemma 2 (Asymptotic optimality of the R-JIQ algorithm)**.** *For the policy R-JIQ, and for any* $\frac{\lambda}{\lambda^{\max}} \leq 1 - \frac{1}{n^\beta}, 0 < \beta < \frac{1}{2}, \gamma = \frac{1}{2}\left(\frac{1}{2} - \beta\right)$ *and queues with limited buffer* $b = o(\sqrt{\ln n})$*, the following steady-state properties hold for large enough* n*.*

(1) The expected response time:

$$\mathbb{E}\left[T_{R\text{-}JIQ}\right] \leq \left(1 - \frac{1}{n^\gamma}\right) T_{LP\text{-}LB}(\lambda) + O\left(\min\left\{\frac{1}{n^\gamma}, \frac{1}{\sqrt{n \ln n}}\right\}\right). \tag{5}$$

(2) The expected achieved accuracy:

$$\mathbb{E}[a_{R\text{-}JIQ}] \geq a^*. \tag{6}$$

(3) The blocking probability:

$$\mathbb{E}[p^B_{R\text{-}JIQ}] \leq O\left(\frac{\ln n}{\sqrt{n}}\right). \tag{7}$$

Proof sketch of Lemma 2. The proof of Lemma 2 relies primarily on the properties of the JIQ policy, leveraging the results of [15]. The key idea involves dividing the system into K queueing subsystems, with subsystem-i housing all class-i servers. Notably, when $0 < \beta < \frac{1}{2}$, an appropriately chosen γ ensures that each subsystem operates within the sub-Halfin-Witt regime.

It is crucial to choose the hyper-parameter carefully. Smaller values of γ may lead to individual queueing systems experiencing loads beyond the sub-Halfin-Whitt regime, while larger values might take $\boldsymbol{p}^*_{\text{R-JIQ}}(\lambda)$ far from $\boldsymbol{p}^*(\lambda)$. We show that $\gamma = 0.5(0.5 - \beta)$ satisfies both criteria. This implies that each subsystem is in the sub-Halfin-Whitt regime, resulting in zero queueing delays.

Furthermore, as the system size n increases, the probability vector $\boldsymbol{p}^*_{\text{R-JIQ}}(\lambda)$ converges to $\boldsymbol{p}^*(\lambda)$ implying that $p^*_i(\lambda)$ fraction of queries is asymptotically served using class-i servers. As a result, the R-JIQ policy exhibits asymptotic stability and feasibility. A detailed proof is provided in Appendix A. □

Variants of R-JIQ. The asymptotic success behind the R-JIQ policy is due to the optimal choice of the traffic allocation represented by $\boldsymbol{p}^*_{\text{R-JIQ}}(\lambda)$ and the minimization of the waiting time. Hence, any policy that achieves zero wait time can be used for our purpose. Another decorated policy that excels at minimizing wait time is the *Join the Shortest Queue* (JSQ), which chooses the shortest queue. The *JSQ-d* policy modifies the JSQ policy by choosing d-servers uniformly at random and joining the shortest queue among those servers. Using similar arguments, their randomized counterparts given by R-JSQ and R-JSQ-d can also achieve asymptotic optimal latency and feasibility. However, these methods come with their own challenges, particularly in implementation. The token-based approach is unsuitable for queue-length-aware policies, posing difficulties in practical deployment.

3.3 Experiments: Asymptotic optimality of R-JIQ policy

In this subsection, we present experimental results to corroborate our result in Lemma 2. We consider inference systems with the numbers of servers n, ranging from 2^4 to 2^{12}. The load $\rho = \frac{\lambda}{\lambda^{\max}}$ is fixed at 0.5. We consider four classes of servers with $\boldsymbol{\mu} = (2, 1, 0.9, 0.1)$, $\boldsymbol{a} = (70, 75, 80, 100)$ and $\alpha_i = 0.25$ for all i. The benchmark accuracy a^* is detailed in the respective simulation captions. For every experiment, the queueing system starts with all empty queues and operates until $n \times 10^5$ queries leave the system, with results averaged over 50 trials. Fig. 4 illustrates the asymptotic optimality of the R-JIQ policy for different values of a^*. Besides the exponential service-time distribution, we also consider deterministic service time in our experiments. This is to mimic the reliable performance of GPUs as evidenced in Fig. 2 and the batching of inference tasks, which reduces variability across batches. For moderate load and higher accuracy constraint a^*, the latency achieved converges to $T_{\text{LP-LB}}(\lambda)$ as quickly as $n = 64$; however, the convergence is slower for lower values of a^*. Lower a^* implies higher $\lambda^{\max}$, which increases λ even at the same load, explaining slower convergence. We also observe that as the distribution changes from exponential to deterministic, convergence occurs faster because of a lesser variability in the service times.

3.4 Drawbacks of the R-JIQ policy

Although the R-JIQ policy always achieves the benchmark accuracy and asymptotically achieves the optimal response time, it has some drawbacks. The limitation of the R-JIQ policy arises primarily from the fact that the policy assumes complete knowledge of the arrival rate to compute the vector $\boldsymbol{p}^*(\lambda)$ and the hyper-parameter γ, which can be challenging to obtain in practice. One can also argue that the policy must solve the LP, given in (3), to determine the optimal probability. This is not a huge deal, as the time complexity to solve the LP is $poly(K)$ and is a one-time cost. Thus, the question arises: *Is there a way to mitigate the problem of the known arrival rate in the R-JIQ policy?*

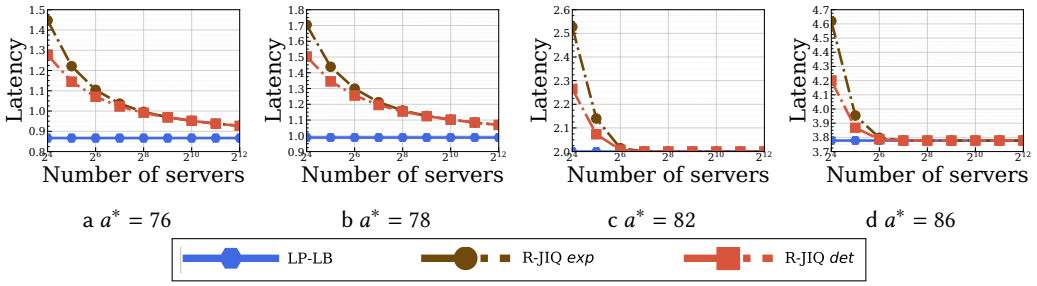

Fig. 4. An illustration of the asymptotic optimality of the R-JIQ policy, the latency achieved by the R-JIQ policy converges to the optimal latency $T_{\text{LP-LB}}(\lambda)$. The standard deviation for the plots is $O(1e^{-3})$.

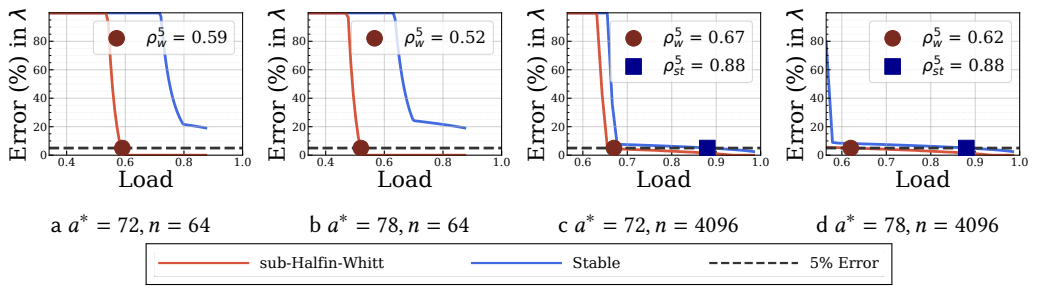

Fig. 5. An illustration of the sensitivity of the R-JIQ policy to noisy estimates of the arrival rate λ. The orange line displays the maximum percentage error in arrival rate to ensure the zero wait time criteria, while the blue line signifies the threshold to ensure system stability. The circle and square represent the load values, where the error threshold for the sub-Halfin-Whitt and stability criteria is 5%.

Estimating the arrival rate addresses a way to mitigate the problem. One way to do this involves tracking the total number of queries that enter the system. By counting arrivals over a time interval $[0, \tau]$ and dividing by τ, one can obtain an estimate $(\hat{\lambda}n)$ of the arrival rate, which in expectation is λn. A small τ results in a noisy estimate, while a large τ gives a good estimate, each with its cons.

A noisy estimate $(\hat{\lambda})$ can lead to system instability. This stems from the fact the solution of the LP enforces $\hat{\lambda}\boldsymbol{p}^*(\hat{\lambda}) \leq \boldsymbol{\mu_\alpha}$ and not $\lambda\boldsymbol{p}^*(\hat{\lambda}) \leq \boldsymbol{\mu_\alpha}$. The issue is further exacerbated, as the hyperparameter γ also depends on the arrival rate λ. Hence, any underestimation of λ can cause $\lambda\boldsymbol{p}^*(\hat{\lambda}) \not\leq \boldsymbol{\mu_\alpha}$ which violates the capacity constraint making the system unstable, see Example 1.

Example 1. Consider a queueing system with three server classes with $\boldsymbol{\mu} = (1, 0.5, 0.25)$, $\boldsymbol{a} = (40, 50, 100)$ and $\alpha_i = 1/3$ for all i and $a^* = 52$. Assume $\lambda = 0.8\lambda^{\max}$, but the estimated arrival rate $\hat{\lambda} = 0.79\lambda^{\max}$, accounting for 1.25% estimation error. If one computes the optimal probability using $\hat{\lambda}$, it turns out that the class-3 serves become unstable as $\lambda p_3^*(\hat{\lambda}) = 0.0843 > 0.0833 = \alpha_3\mu_3$.

Even if stability is maintained, the load in individual subsystems can go beyond the sub-Halfin-Whitt regime because of the incorrect estimate, which can increase the wait time. Fig. 5 illustrates issues of R-JIQ policy under noisy estimates. The system parameters $\boldsymbol{\mu}$, $\boldsymbol{\alpha}$, and $\boldsymbol{a}$ remain unchanged from the previous configurations in Section 3.3. The y-axis indicates the percentage error in arrival rates, whereas the x-axis represents the actual load of the system. The brown line displays the maximum percentage error in the estimate of the arrival rate λ, which the system can tolerate while ensuring each subsystem-i operates within the sub-Halfin-Whitt regime, signifying the zero

wait time criteria. The blue line signifies the maximum error threshold to ensure system stability. Naturally, as the load increases, the error tolerance of the estimate decreases since each subsystem tends to operate in heavier traffic. The same logic applies when happens when the benchmark accuracy a^* increases. The ρ_w^5, marked by a circle, represents the load where a 5% error pushes the system outside the sub-Halfin regime, thus increasing the waiting time. Similarly, ρ_{st}^5 signifies the load where a 5% error can lead to system instability. For smaller values of n and an error tolerance of 5%, the system can operate beyond the sub-Halfin-Whitt regime even for a moderate load level, approximately 0.55. As the system scales, the sensitivity increases; close to a load of 0.88, a 5% error can destabilize the system. Thus, a small interval of τ can be detrimental to stability and latency.

The instability issue motivates one to use a large τ to obtain a more accurate estimate, but that could render the system unresponsive to changes in arrival rates. Real-world scenarios often exhibit varying traffic loads influenced by factors such as time of day and location. The instantaneous traffic into the system can be significantly higher than the long-term average, which can cause instability issues. Hence, selecting an appropriate time horizon to estimate arrival rates is challenging.

4 TRACK THE ACCURACY DIFFERENCE

In this section, we present our second proposed algorithm, the *Track-the-Accuracy-Difference* (TAD) policy, an empirical approach to resolve the issue of known arrival rates. The algorithm is very intuitive and consistently achieves the benchmark accuracy. However, the algorithm does not achieve the optimal response under some circumstances, as evidenced in Example 2. The counterexample given in Example 2 motivates us to study the LP in greater detail, paving the way for our subsequent algorithm design.

The *Track-the-Accuracy-Difference* (TAD) achieves the benchmark accuracy using an online method, removing dependency on the arrival rate. The dispatcher maintains a variable Δ_{a^*} initialized at zero to indicate the system's deviation from the benchmark accuracy. Upon the arrival of a query, the dispatcher defines a set $\mathcal{I} = \{i : i \in \{1, \ldots, K\}, \Delta_{a^*} + a_i - a^* \geq 0\}$. Intuitively, the set $\mathcal{I}$ represents the classes of servers that can maintain a positive Δ_{a^*}. It then utilizes the *Join-the-Idle-Queue* strategy from R-JIQ to minimize the waiting time. Similar to R-JIQ, when a server becomes idle, it sends a token to the central dispatcher, informing the dispatcher of its availability. The dispatcher then aims to send the query to an idle server of class-i for some $i \in \mathcal{I}$. If there are multiple idle servers, preference is given to the faster server. If there are no idle class-i servers, for any $i \in \mathcal{I}$, a class-i from the set $\mathcal{I}$ is randomly chosen, and one server within the class is chosen uniformly at random. Upon routing a query to a class-i server, Δ_{a^*} is updated to $\Delta_{a^*} = \Delta_{a^*} + (a_i - a^*)$.

The TAD policy introduces the concept of Δ_{a^*}. The policy's strategy of maintaining a positive Δ_{a^*} guarantees the achievability of the benchmark accuracy. Because it prioritizes idle servers, one would expect the policy to stabilize the queues, i.e., the underlying CTMC is positive recurrent. Furthermore, since the policy prioritizes faster servers when Δ_{a^*} is positive, one may expect the policy to achieve the optimal response time $T_{\text{LP-LB}}(\lambda)$ asymptotically. Contrary to our expectations, the TAD policy's preference for faster servers can lead to sub-optimal performance, particularly in low-load scenarios. Example 2 illustrates one case of sub-optimality.

Example 2. Consider a queueing system with three server classes with $\boldsymbol{\mu} = (1, 0.5, 0.25)$, $\boldsymbol{a} = (40, 50, 100)$ and $\alpha_i = 1/3$ for all i and $a^* = 45$. Assume that the arrival rate λ is small; hence, there is almost no queueing. Under these parameters, the TAD policy would route queries to the servers in class-1 and 2 in a round-robin fashion. The expected response time of the system would be $\mathbb{E}[T_{\text{TAD}}] = 0.5(1/1 + 1/0.5) = 1.5$. However, an alternative policy, where one query is routed to a class 3 server, and the next 11 queries are routed to a class 1 server, performs better. The

accuracy achieved under this policy is $(40 \times 11 + 100)/12 = 45$ and it achieves a response time of $(\frac{11}{12} \times \frac{1}{1} + \frac{1}{12} \times \frac{1}{0.25}) = \frac{15}{12} = 1.25$, performing 16.67% better than the TAD policy.

5 ORDERED PAIRS SOLVE THE LINEAR PROGRAM

In this section, we introduce ordered pairs and their application in our proposed TAD-OP policy. Taking Example 2 as motivation, we analyze the LP in the low-load regime in Section 5.1. This analysis highlights the significance of pairs of server classes. We generalize this concept into routing tuples in Section 5.2. In Section 5.3, we present the Waterfilling algorithm that uses the routing tuples iteratively to solve the LP. Theorem 1 provides the convergence guarantee of the waterfilling algorithm to the LP solution. Finally, we present the *Track-the-Accuracy-Difference under Ordered pairs* (TAD-OP) policy in Section 5.4 that mimics the waterfilling algorithm in a stochastic sense.

5.1 Low Load Analysis: Introducing the concept of Pairs

To understand why the TAD policy sometimes fails to achieve optimal latency $T_{\text{LP-LB}}(\lambda)$, we study the LP, defined in (3), in the low-load regime. As $\lambda \to 0$, the capacity constraint, given by $\lambda \boldsymbol{p} \le \boldsymbol{\mu_\alpha}$, becomes redundant. Consequently, we introduce a relaxed version of the linear programming problem, defined in (8), where we omit the capacity constraints. Lemma 3 then provides a closed-form solution of the relaxed LP.

$$\begin{aligned} T_{\text{Relaxed-LP-LB}}(\lambda) = \min_{\boldsymbol{p} \in \mathbb{R}^K} \quad & \boldsymbol{p} \cdot \boldsymbol{\mu}^{-1} \\ \text{s.t.} \quad & \boldsymbol{p} \ge \boldsymbol{0}, \boldsymbol{p} \cdot \boldsymbol{1} = 1, \boldsymbol{p} \cdot \boldsymbol{a} \ge a^*. \end{aligned} \tag{8}$$

Lemma 3 (Optimality of two servers)**.** *For the relaxed linear program, defined in* (8)*, we have the following: for all $\lambda > 0$, the optimal solution $\bar{\boldsymbol{p}}^* = (\bar{p}_1^*, \bar{p}_2^*, \ldots, \bar{p}_K^*)$ is either $\bar{p}_i^* = 1$ with $a_i = a^*$ or $\bar{\boldsymbol{p}}^* = \boldsymbol{q}^{(i_1, j_1)}$ with*

$$i_1, j_1 = \underset{i,j: a_i < a^* < a_j}{\arg\min} \left(\frac{1}{\mu_i} \frac{a_j - a^*}{a_j - a_i} + \frac{1}{\mu_j} \frac{a^* - a_i}{a_j - a_i} \right) \tag{9}$$

where $q_i^{(i,j)} = \frac{a_j - a^*}{a_j - a_i}, q_j^{(i,j)} = \frac{a^* - a_i}{a_j - a_i}$ and $q_k^{(i,j)} = 0$ for any $k \ne i, j$.

The proof of Lemma 3 is given in Appendix B.1. The insight derived from Lemma 3 is instructive. Specifically, if no server class precisely achieves accuracy a^*, using only two server classes is optimal: the class i_1 with an accuracy below a^* and the class j_1 with an accuracy exceeding a^*. One can verify that $\{i_1, j_1\}$ in Example 2 is $\{1, 3\}$ explaining why TAD is suboptimal. So *what happens as the load increases and the classes i_1 and j_1 are insufficient to handle the traffic?* The natural choice would be to choose the next most effective pair, represented by $\{i_2, j_2\}$; however, we need to add a few more pairs to make the idea work. We generalize this idea to tuples in the next subsection.

5.2 Routing Tuples

In this subsection, we define *routing tuple* using Definition 2. Intuitively, the routing tuples describe a way to allocate the traffic λ into one or two server classes along with its cost addition to the LP.

Definition 2. *A tuple $(S, \boldsymbol{q}_S, v_S)$ is defined as a routing tuple if it satisfies the following conditions*

(1) Routing set: A set of server classes $S \subseteq \{1, 2, \ldots, K\}$ with size $|S| \le 2$,

(2) Routing vector: A K-dimensional vector $\boldsymbol{q}_S$ defined as

$$\boldsymbol{q}_S = \underset{\boldsymbol{q}' \in \mathbb{R}^K}{\arg\min} \{ \boldsymbol{q}' \cdot \boldsymbol{\mu}^{-1} : \boldsymbol{q}' = (q_1', \ldots, q_K'),\ \boldsymbol{q}' \cdot \boldsymbol{1} = 1,\ \boldsymbol{q}' \cdot \boldsymbol{a} \ge a^*,\ q_j' = 0, \forall j \notin S \}, \tag{10}$$

(3) Cost: $v_S = \boldsymbol{q}_S \cdot \boldsymbol{\mu}^{-1}$

One can notice similarities between the definition of routing vector and the LP, defined in (3). The routing vector $\boldsymbol{q}_S$ resembles the optimal routing probability $\boldsymbol{p}^*(\lambda)$ but when restricted to only using server classes in the set S, while v_S represents the cost. However, a key difference is that the definition of the routing vector does not enforce the non-negativity of the vector. The reason is that there might exist sets for which a non-negative routing vector is not possible. E.g., take $S = \{1, 2\}$ with $\boldsymbol{a} = (40, 50, 100)$ and $a^* = 60$; a non-negative routing vector cannot satisfy $\boldsymbol{q}_S \cdot \boldsymbol{a} \geq a^*$. Next, we concretely define all the routing tuples for our system.

We first define routing tuples with routing sets $S^{(i,j)} = \{i, j\}$ for all $1 \leq i < j \leq K$. The routing vector associated with $S^{(i,j)}$ is $\boldsymbol{q}^{(i,j)} = \left(q_1^{(i,j)}, \ldots, q_K^{(i,j)}\right)$. By solving the set of linear equations $\boldsymbol{q}' \cdot \boldsymbol{1}$ and $\boldsymbol{q}' \cdot \boldsymbol{a} = a^*$ subject to the condition that $q'_k = 0, k \neq i, j$, we can obtain that $q_i^{(i,j)} = \frac{a_j - a^*}{a_j - a_i}, q_j^{(i,j)} = \frac{a^* - a_i}{a_j - a_i}$ and $q_k^{(i,j)} = 0$ for $k \neq i, j$. If one increases the weight of the i-th index, then the accuracy constraint is not satisfied, while if one increases the weight of the j-th index, the cost will increase since $\mu_j > \mu_i$ explaining why $\boldsymbol{q}^{(i,j)}$ takes the above form. The associated cost is then given by $v^{(i,j)} = \boldsymbol{q}^{(i,j)} \cdot \boldsymbol{\mu}^{-1}$.

When the accuracies a_i, a_j satisfy $a_i < a^* < a_j$, then $\boldsymbol{q}^{(i,j)} \geq 0$ and $\boldsymbol{q}^{(i,j)} \cdot \boldsymbol{1} = 1$. In this case, the associated routing vector $\boldsymbol{q}^{(i,j)}$ is a valid probability vector. The use of the routing vector $\boldsymbol{q}^{(i,j)}$ to minimize cost is intuitive, as seen in Lemma 3. However, when $a_i < a_j < a^*$ or $a_i > a_j > a^*$, the associated routing vector $\boldsymbol{q}^{(i,j)}$ is non-positive. These routing vectors, though not non-negative, play a critical role in the convergence of our algorithm.

We also introduce tuples with a singleton routing set $S^{(i)} = \{i\}$, for all i such that $a_i \geq a^*$, along with the associated routing vector $\boldsymbol{q}^{(i)} = \boldsymbol{e}_i$ and the cost $v^{(i)} = 1/\mu_i$. These routing tuples aim to accommodate scenarios where the inference system can achieve an accuracy higher than a^* but not exactly a^*. For example, in an inference system with $\boldsymbol{a} = (40, 50)$ and a benchmark accuracy $a^* = 30$, it is impossible to achieve exactly the expected accuracy of a^*.

Using the above definition of routing tuples, we define the *optimal set of routing tuples* $\mathcal{S}^*_{\text{opt}}$ as

$$\mathcal{S}^*_{\text{opt}} = \{(S^{(i,j)}, \boldsymbol{q}^{(i,j)}, v^{(i,j)}) : i < j, v^{(i,j)} > 0\} \cup \{(S^{(i)}, \boldsymbol{q}^{(i)}, v^{(i)}) : i \text{ s.t. } a_i \geq a^*\}, \tag{11}$$

For simplicity, along with notational abuse, we refer to the i-th element of $\mathcal{S}^*_{\text{opt}}$ as $(S^*_i, \boldsymbol{q}^*_i, \boldsymbol{v}^*_i)$. The *optimality* refers to the fact that, for any $\lambda \leq \lambda^{\max}$, the subgradient of $(\lambda T_{\text{LP-LB}}(\lambda))$ equals v^*_i for some $i \leq |\mathcal{S}^*_{\text{opt}}|$ (refer to Lemma 5). For the purpose of our algorithm, we also assume that the set $\mathcal{S}^*_{\text{opt}}$ is ordered w.r.t. its third index, i.e., for any $i < j$, $v^*_i \leq v^*_j$. We also remove routing tuples with negative cost, as our waterfilling algorithm provided in Section 5.3 never uses these routing tuples.

5.3 The Waterfilling algorithm

Our previous discussions highlight the importance of solving the LP to develop an effective queueing algorithm. While the LP is deterministic, in contrast to the stochastic nature of the inference system, the solution methodology can be crucial in guiding the algorithm design for the system. This motivates us to design a deterministic solution of the LP that reduces the dependency on λ.

We accomplish this objective using the Waterfilling algorithm that solves the LP, defined in (3), using an iterative policy. The waterfilling algorithm uses the $\mathcal{S}^*_{\text{opt}}$ defined in Section 5.2 and mainly uses the arrival rate λ as a stopping criterion. A detailed description of this algorithm is presented in Algorithm 1. The outcome of the waterfilling algorithm is represented by the vector $\boldsymbol{p}$, accompanied by the cost function $N_{\text{WF}}(\lambda)$. For any given value of λ, we say that $N_{\text{WF}}(\lambda)$ exists if the waterfilling algorithm outputs a feasible solution.

Theorem 1 establishes that the solution from the waterfilling algorithm converges to the optimal solution $\boldsymbol{p}^*(\lambda)$. The cost of the waterfilling solution $N_{\text{WF}}(\lambda)$ converges to $N_{\text{LP-LB}}(\lambda)$. Using Little's

law, the term $N_{\text{LP-LB}}(\lambda)$ represents the minimum normalized steady-state number of queries in an inference system under any policy that achieves the benchmark accuracy a^*, highlighting the optimality of the water filling algorithm. However, Theorem 1 hinges on a uniqueness assumption given in Assumption 1. The uniqueness criteria is a minor assumption as a slight perturbation in system parameters achieves the criteria. The unique value of every v_i^* is a sufficient condition that ensures the uniqueness of the LP solution. We conjecture that if the LP has multiple optimums, the waterfilling algorithm converges to one of the optimums.

Assumption 1. *The linear program, defined in* (3)*, has a unique solution, for any* $\lambda \in (0, \lambda^{\max}]$.

Theorem 1 (Optimality of the Waterfilling algorithm)**.** *For all* $\lambda \in [0, \lambda^{\max}]$*, with Assumption 1 holding true, the linear program, defined in* (3)*, and the waterfilling algorithm are feasible and*

$$N_{WF}(\lambda) = N_{LP\text{-}LB}(\lambda) \triangleq \lambda T_{LP\text{-}LB}(\lambda). \tag{12}$$

The proof sketch of Theorem 1 is provided in Section 6. Intuitively, Algorithm 1 generalizes Lemma 3, determining traffic allocation when the server classes i_1 and j_1 are insufficient to handle traffic. The main tricky part is the inclusion of tuples with non-positive routing probability. We discuss these pairs in more detail in Section 6.

5.4 Track the Accuracy Difference under Ordered Pairs

In this subsection, we introduce our main proposed algorithm, the *Track-the-Accuracy-Difference under Ordered Pairs* (TAD-OP) policy. This policy is inspired by the Waterfilling algorithm presented in Algorithm 1 and the TAD policy elaborated in Section 4. The primary goal of the algorithm is to emulate the behavior of the Waterfilling algorithm without relying on the knowledge of the arrival rate λ. We leverage the following key principles to ensure this:

(1) *Routing vector*: We use Δ_{a^*} introduced in the TAD policy, which balances the accuracy and provides a heuristic way to apply the routing vector.
(2) *Zero Waiting Time*: The Join-the-Idle-Queue approach is utilized to achieve minimal wait time. Unlike the TAD policy, we prioritize idle queues over positive Δ_{a^*}. This comes from empirical observation (see Section 7), where the TAD policy becomes unstable in order to maintain a strictly positive Δ_{a^*}. Although this strategy might temporarily push the value of Δ_{a^*} to be negative, we anticipate that, in a steady state, Δ_{a^*} would oscillate around zero.
(3) *Eliminating the dependency on the arrival rate* λ: Within the waterfilling algorithm, the arrival rate serves as the stopping criterion; in our inference system, job departures serve a similar purpose. When the system has a sufficient number of jobs, the departure rate equates to the arrival rate, thereby serving as a self-regulating mechanism to prevent overfilling of servers.

The routing under tuples with non-positive routing probability remains tricky to handle. We explain how TAD-OP handles those tuples after describing the policy. We first define *routable* tuples based on the availability of idle servers in each class. A routing tuple $(S_i^*, \boldsymbol{q}_i^*, v_i^*)$, where $\boldsymbol{q}_i^* = (q_{i1}^*, \ldots, q_{iK}^*)$, is *routable* if all server classes $k \in S_i^*$ meet one of the following conditions:

(1) $q_{ik}^* > 0$ and there is at least one idle server of class k,
(2) $q_{ik}^* < 0$ and there is at least one busy server of class k.

To determine whether a tuple $(S_i^*, \boldsymbol{q}_i^*, v_i^*)$ with a non-negative routing vector is *routable*, the dispatcher can check for a token from server classes in S_i^*. For tuples $(S^{(i,j)}, \boldsymbol{q}^{(i,j)}, v^{(i,j)})$, with a non-positive routing vector $\boldsymbol{q}^{(i,j)}$ containing a positive element at i and negative at j, it is routable if a token from a class-i server is available and there are not more than $(\alpha_j n - 1)$ tokens from class-j servers. A similar method can be used if the signs of the indices are reversed.

A tuple with a non-positive routing vector must count the total number of tokens from each class. A simple implementation method is to store K variables at the dispatcher to track the number of tokens available from each class. Using routable tuples, we describe the TAD-OP policy in Algorithm 2. The dispatcher initializes the accuracy difference Δ_{a^*} to zero. For every job arrival, the dispatcher runs Algorithm 2, which outputs the queue index j and the updated Δ_{a^*}. The dispatcher then updates Δ_{a^*} and sends the query to the server, indexed as j.

Algorithm 1 Waterfilling

Input:
$\mathcal{S}^*_{\text{opt}}, \lambda, \boldsymbol{\mu}$
Initialize:
Weights:
$w_i \leftarrow 0$, for all $i \in 1, \ldots, |\mathcal{S}^*_{\text{opt}}|$
Initial probability: $\boldsymbol{p} \leftarrow \mathbf{0}$
Initial cost: $N_{\text{WF}} \leftarrow 0$
Transient arrival rate: $\lambda_{\text{tr}} \leftarrow \lambda$
for $k \leftarrow 1 : |\mathcal{S}^*_{\text{opt}}|$ **do**
$w_k \leftarrow \max\{w : \mathbf{0} \leq \lambda \boldsymbol{p} + w \boldsymbol{q}^*_k \leq \boldsymbol{\mu}_{\boldsymbol{\alpha}}\}$
if $w_k \notin [0, \lambda_{\text{tr}}]$ **then**
$w_k \leftarrow \max\{0, \min\{w_k, \lambda_{\text{tr}}\}\}$
end if
$\boldsymbol{p} \leftarrow \boldsymbol{p} + \dfrac{w_k \boldsymbol{q}^*_k}{\lambda}$
$N_{\text{WF}} \leftarrow N_{\text{WF}} + w_k v^*_k$
$\lambda_{\text{tr}} \leftarrow \lambda_{\text{tr}} - w_k$
if $\lambda_{\text{tr}} = 0$ **then**
Break
end if
end for
if $\lambda_{\text{tr}} > 0$ **then**
The problem is not feasible
else
Output:
Probability vector: $\boldsymbol{p}$
Tuple weights: w_i's
Cost: N_{WF}
end if

Algorithm 2 TAD-OP

Input:
$\mathcal{S}^*_{\text{opt}}, \boldsymbol{a}, a^*, \Delta_{a^*}$
for $k \leftarrow 1 : |\mathcal{S}^*_{\text{opt}}|$ **do**
if $(S^*_k, \boldsymbol{q}^*_k, v^*_k)$ is routable **then**
if $|S^*_k| = 1$ or $\boldsymbol{q}^*_k \not\geq 0$ **then**
Choose $i \in S^*_k$ s.t. $\boldsymbol{q}^*_k(i) > 0$
else
if $\Delta_{a^*} > 0$ **then**
Choose $i \in S^*_k$ s.t. $a_i < a^*$
else
Choose $i \in S^*_k$ s.t. $a_i > a^*$
end if
end if
Break
end if
end for
if no routable tuple **then**
for $k \leftarrow K : 1$ **do**
if #idle class-k server> 0 **then**
$i \leftarrow k$
Break
end if
end for
end if
if no idle server **then**
$i \leftarrow$ Pick randomly in $\{1, \ldots, K\}$
end if
if idle class-i server exists **then**
$j \leftarrow$ Any idle class-i server
else
$j \leftarrow$ Any randomly chosen class-i server
end if
$\Delta_{a^*} \leftarrow \Delta_{a^*} + a_i - a^*$
Output:
Server index: j
Updated accuracy difference : Δ_{a^*}

Intuition on the working of TAD-OP. For tuples with non-negative routing vector, the TAD-OP policy utilizes the sign of Δ_{a^*} to determine the server class, roughly ensuring that jobs are routed

according to the routing vector. However, when dealing with tuples with non-positive routing vectors, the TAD-OP policy operates with a slight delay. For any tuple, $\hat{S}_-$, with a non-positive routing vector, there always exists a corresponding tuple, $\hat{S}_+$, with a non-negative routing vector and a lower cost. The policy uses $\hat{S}_-$ when $\hat{S}_+$ is not routable. However, using the tuple $\hat{S}_-$ shifts the Δ_{a^*} away from zero. However, the tuple $\hat{S}_+$ eventually becomes routable as the system sees departures from server classes in $\hat{S}_+$. By this time, the already drifted Δ_{a^*} guides the policy to favor one class over the other within $\hat{S}_+$. This, combined with the fact that the less favored server classes in $\hat{S}_+$ have queries that are departing, emulates the negative probability.

5.5 Similarities with Existing Queueing Algorithms

At first glance, our approach using pairs and Δ_{a^*} might seem strange. However, its underlying logic bears similarities with established queueing algorithms. Without the accuracy constraint, the Join the Idle Queue (JIQ) policy [15] is near-optimal in the sub-Halfin-Whitt regime for homogeneous servers. When considering heterogeneous servers, the Join the Fastest of the Idle Queue (JFIQ) policy [28] is optimal. A common thread in these policies is prioritizing idle queues, followed by a preference for faster servers, essentially generating a priority class. In the context of an inference system that demands a balance between accuracy and latency, the use of ordered pairs recalls the priority classes introduced by JFSQ. However, a pair must be considered since a single-server class cannot balance the trade-off between minimizing latency and achieving accuracy.

6 THE PROOF SKETCH OF THEOREM 1

In this section, we present the proof sketch of Theorem 1. We start off by providing intuition behind tuples with non-positive routing probability.

Proper-pair vs Shuffle-pair. When the accuracies a_i, a_j satisfy $a_i < a^* < a_j$, then $\boldsymbol{q}^{(i,j)} \geq 0$ and we call all such routing tuples as *properly-paired* routing tuples. On the other hand, if $a_i < a_j < a^*$ or $a_i > a_j > a^*$, then $\boldsymbol{q}^{(i,j)} \ngeq \mathbf{0}$. We call such routing tuples as *shuffling-pair* routing tuples. Using such a tuple in the waterfilling algorithm indicates the movement of traffic from $i \to j$ or $j \to i$ depending on the value of a^* and its purpose is to revert bad decisions as explained in Example 3.

Example 3. Consider an inference system with $\boldsymbol{\mu} = (1, 0.5, 0.25)$, $\boldsymbol{a} = (40, 50, 100)$, and $\boldsymbol{\alpha} = (0.025, 0.95, 0.025)$. Given a benchmark accuracy of $a^* = 52$, it can be verified that the optimal pair is $\{1, 3\}$ with the corresponding routing vector $\boldsymbol{q}^{(1,3)} = (0.8, 0, 0.2)$. Using the pair $\{1, 3\}$, the maximum arrival rate λ that satisfies the capacity constraint is 0.0625. However, choosing the proper-pair $\{2, 3\}$ allows a larger $\lambda = 0.25$ due to the higher value of α_2. If one uses the optimal pair with its routing vector, then class-3 becomes fully occupied for any $\lambda \geq 0.0625$ with no proper-pair available after that. The shuffle pair $\{1, 2\}$ creates a way to revert the decision by taking traffic out of the class-1 servers and placing it in class-2. The redistribution increases the achieved accuracy as queries are taken from a less accurate class to a more accurate one. This indirectly provides leeway to accommodate more traffic in the class-1 servers without breaching the accuracy constraint. The redistribution can continue until the class-2 servers are filled, or the class-1 servers are empty, explaining the *shuffle*. In this example, it turns out that the class-1 server becomes empty; thus, the final traffic allocation is as if class-1 was never used, and only pair $\{2, 3\}$ is used.

With this intuition, we provide two key lemmas to characterize the LP. We first present Lemma 4 that establishes the convexity of $N_{\text{LP-LB}}(\lambda)$. We then state Lemma 5, which generalizes Lemma 3. For simplicity, we present all the lemmas for the case where there is no server class with accuracy exactly a^*; however, extending it should be trivial.

Lemma 4 (Convexity of LP). *For the LP, defined in* (3)*, and maximum feasible arrival rate* $\lambda^{\max}$*, it holds that*

(1) $\lambda^{\max}$ *is well defined and satisfies* $\lambda^{\max} \in (0, \sum_{i=1}^{K} \alpha_i \mu_i]$*,*
(2) the LP is feasible for all $\lambda \in [0, \lambda^{\max}]$*,*
(3) $N_{LP\text{-}LB}(\lambda)$ *is a continuous convex function w.r.t* λ *for all* $\lambda \in [0, \lambda^{\max}]$*,*
(4) For all $\lambda \in [0, \lambda^{\max})$*, the right derivative of the function* $N_{LP\text{-}LB}(\lambda)$*, given by* $N'_{LP\text{-}LB}(\lambda^+)$*, exists and satisfies* $N'_{LP\text{-}LB}(\lambda^+) \geq 0$*.*

The proof of Lemma 4 is given in Appendix B.2. Next, we define

$$A_{\text{LP-LB}}(\lambda) = \lambda \boldsymbol{p}^*(\lambda) = \left(\lambda p_1^*(\lambda), \ldots, \lambda p_K^*(\lambda)\right), \tag{13}$$

where $\boldsymbol{p}^*(\lambda) = \left(p_1^*(\lambda), \ldots, p_K^*(\lambda)\right)$ is the optimal solution of the LP, defined in (3).

Lemma 5 (Routing Tuples contain sub-gradients of $N_{\text{LP-LB}}(\lambda)$)**.** *Under Assumption 1, it holds that the right derivative of the function* $A_{LP\text{-}LB}(\lambda)$*, given by* $A'_{LP\text{-}LB}(\lambda^+)$*, exists for all* $\lambda \in [0, \lambda^{\max})$ *and* $A'_{LP\text{-}LB}(\lambda^+) = \boldsymbol{q}^*_{i(\lambda, \boldsymbol{a}, \boldsymbol{\mu})}$ *for some index* $i(\lambda, \boldsymbol{a}, \boldsymbol{\mu})$ *that depends only on the system parameters* $\lambda, \boldsymbol{a}, \boldsymbol{\mu}$*.*

This is the most critical lemma, and we provide a proof sketch of the lemma in Section 6.1. Intuitively, the right derivative $A'_{\text{LP-LB}}(\lambda^+)$ denotes the instantaneous optimal routing probability when the system currently has the optimal steady-state allocation under arrival rate λ. An immediate consequence of Lemma 5 is that the right derivative

$$N'_{\text{LP-LB}}(\lambda^+) = A'_{\text{LP-LB}}(\lambda^+) \cdot \boldsymbol{\mu}^{-1} = \boldsymbol{q}^*_{i(\lambda, \boldsymbol{a}, \boldsymbol{\mu})} \cdot \boldsymbol{\mu}^{-1} = v^*_{i(\lambda, \boldsymbol{a}, \boldsymbol{\mu})} \tag{14}$$

which explains the phrase "routing tuples contain sub-gradients of $N_{\text{LP-LB}}(\lambda)$". Observe that Lemma 3 is a special case of Lemma 5 since $A'_{\text{LP-LB}}(0^+) = \boldsymbol{q}^{(i_1, j_1)}$ follows from the latter lemma.

Next, define the terms $\lambda^{\max}_{\text{WF-F}}$ and $\lambda^{\max}_{\text{WF-O}}$ to characterize important properties of $N_{\text{WF}}(\lambda)$, as

$$\lambda^{\max}_{\text{WF-F}} = \sup\{\lambda' : \lambda' \geq 0, N_{\text{WF}}(\lambda) \text{ is feasible for all } \lambda \in [0, \lambda']\}, \tag{15}$$

and

$$\lambda^{\max}_{\text{WF-O}} = \sup\{\lambda' : \lambda' \geq 0, N_{\text{WF}}(\lambda), N_{\text{LP-LB}}(\lambda) \text{ are feasible}, N_{\text{WF}}(\lambda) = N_{\text{LP-LB}}(\lambda), \text{ for all } \lambda \in [0, \lambda']\}, \tag{16}$$

respectively. Intuitively, $\lambda^{\max}_{\text{WF-O}}$ denotes the maximum value of λ for which the waterfilling algorithm provides an optimal solution to the LP, defined in (3). Similarly, $\lambda^{\max}_{\text{WF-F}}$ denotes the maximum value of λ for which the waterfilling algorithm provides a feasible solution.

The proof of Theorem 1 proceeds as follows. We first show in Lemma 6 that $\lambda^{\max}_{\text{WF-O}}$ and $\lambda^{\max}_{\text{WF-F}}$ are well defined, and $\lambda^{\max}_{\text{WF-O}} \leq \lambda^{\max}_{\text{WF-F}} \leq \lambda^{\max}$. We then show in Lemma 7 that $\lambda^{\max}_{\text{WF-O}} = \lambda^{\max}_{\text{WF-F}}$, i.e., if the waterfilling algorithm outputs a feasible solution, it is also the optimal solution to the original LP, defined in (3). Finally, we use Lemma 8 to show that $\lambda^{\max}_{\text{WF-F}} = \lambda^{\max}$, i.e., the waterfilling algorithm N_{WF} is feasible if and only if the LP, defined in (3), is feasible, thus completing the proof of convergence. We provide the proofs of Lemma 6-8 in Appendix B.3-B.5, respectively.

Lemma 6 (Well Definedness)**.** *The parameters* $\lambda^{\max}_{WF\text{-}O}$ *and* $\lambda^{\max}_{WF\text{-}F}$ *are well defined and satisfy,*

$$\lambda^{\max}_{WF\text{-}O} \leq \lambda^{\max}_{WF\text{-}F} \leq \lambda^{\max}. \tag{17}$$

Lemma 7 (Feasibility equals Optimality)**.** *For all* $\lambda \in [0, \lambda^{\max}_{WF\text{-}F}]$*, the solution of the waterfilling algorithm is an optimal solution to the LP defined in* (3)*, i.e.,* $\lambda^{\max}_{WF\text{-}O} = \lambda^{\max}_{WF\text{-}F}$*.*

Lemma 8 (Waterfilling is Feasible)**.** *For all* $\lambda \in [0, \lambda^{\max}]$*, the waterfilling algorithm is feasible, i.e.,* $\lambda^{\max}_{WF\text{-}F} = \lambda^{\max}$*.*

6.1 Proof Sketch of Lemma 5

Proof of Lemma 5. The existence of a directional derivative under sensitivity analysis has been shown in [7]. Hence, $A'_{\text{LP-LB}}(\lambda^+)$ exists for all $\lambda \in [0, \lambda^{\max})$. Next, we state Lemma 9 and Lemma 10 which provide the key steps to prove Lemma 5.

Lemma 9. *For all $\lambda \in [0, \lambda^{\max})$, the right derivative $A'_{LP\text{-}LB}(\lambda^+)$ satisfy*

$$A'_{LP\text{-}LB}(\lambda^+) \cdot \mathbf{1} = 1 \tag{18}$$

$$A'_{LP\text{-}LB}(\lambda^+) \cdot \boldsymbol{a} \geq a^* \tag{19}$$

Lemma 10 (Unique sub-gradient)**.** *For any $\lambda \in [0, \lambda^{\max})$, if a vector $\boldsymbol{p} = (p_1, \ldots, p_K)$ satisfy the conditions*

(1) $\boldsymbol{p} \cdot \mathbf{1} = 1$,
(2) $\boldsymbol{p} \cdot \boldsymbol{a} \geq a^*$,
(3) for all index $i \in \{1, \ldots, K\}$, if $A'_{LP\text{-}LB}(\lambda^+)(i) = 0$, then $p_i = 0$ and if $A'_{LP\text{-}LB}(\lambda^+)(i) \neq 0$, then $p_i A'_{LP\text{-}LB}(\lambda^+)(i) \geq 0$,
(4) $\boldsymbol{p} \cdot \boldsymbol{\mu}^{-1} \leq A'_{LP\text{-}LB}(\lambda^+) \cdot \boldsymbol{\mu}^{-1}$,

then $\boldsymbol{p} = (p_1, \ldots, p_K) = A'_{LP\text{-}LB}(\lambda^+)$

The proof of Lemma 9 and Lemma 10 are given in Appendix B.6 and B.7, respectively. Intuitively, Lemma 10 suggests that if the vector $\boldsymbol{p}$ is not $A'_{\text{LP-LB}}(\lambda^+)$, then $A'_{\text{LP-LB}}(\lambda^+)$ is not the unique direction of optimal ascent and the underlying LP has multiple optimums, contradicting Assumption 1.

Now, to prove Lemma 5, we first argue that for any $\lambda \in [0, \lambda^{\max})$, $A'_{\text{LP-LB}}(\lambda^+)$ cannot have more than two non-zero elements. One can show that if $A'_{\text{LP-LB}}(\lambda^+)$ has at most two non-zero entries, then $A'_{\text{LP-LB}}(\lambda^+)$ must be of the form $\boldsymbol{q}^*_{i(\lambda,\boldsymbol{\mu},\boldsymbol{a})}$ for some index $i(\lambda, \boldsymbol{\mu}, \boldsymbol{a})$.

To prove this, consider the following possible cases.

(1) If $A'_{\text{LP-LB}}(\lambda^+)$ contains a single non-zero element at index i, Lemma 9 implies $A'_{\text{LP-LB}}(\lambda^+) = \boldsymbol{e}_i$ to satisfy the first statement and $a_i \geq a^*$ to satisfy the second one. Hence, this corresponds to using a singleton routing set $S^{(i)}$.
(2) If $A'_{\text{LP-LB}}(\lambda^+)$ contains two positive elements at index i, j s.t. $a_i < a^* < a_j$, the definition of routing tuples and Lemma 10 implies that $A'_{\text{LP-LB}}(\lambda^+) = \boldsymbol{q}^{(i,j)}$.
(3) The above argument holds when $A'_{\text{LP-LB}}(\lambda^+)(i) > 0, A'_{\text{LP-LB}}(\lambda^+)(j) < 0, a^* < a_i < a_j$ or $A'_{\text{LP-LB}}(\lambda^+)(i) < 0, A'_{\text{LP-LB}}(\lambda^+)(j) > 0, a_i < a_j < a^*$.

For any other case, it can be shown that either the second statement of Lemma 9 is false, or there exists a $\boldsymbol{p}$ satisfying the conditions of Lemma 10 that achieves a lower cost. E.g., if $A'_{\text{LP-LB}}(\lambda^+)$ contains two positive elements at index i, j s.t. $a_i < a_j < a^*$, then $A'_{\text{LP-LB}}(\lambda^+) \cdot \boldsymbol{a}$ is a convex combination of two elements, each of which is less than a^* implying $A'_{\text{LP-LB}}(\lambda^+) \cdot \boldsymbol{a} < a^*$. Similarly, if $A'_{\text{LP-LB}}(\lambda^+)$ contains two positive elements at index i, j s.t. $a^* < a_i < a_j$, then $\boldsymbol{p} = \boldsymbol{e}_i$ satisfies Lemma 10 implying $A'_{\text{LP-LB}}(\lambda^+)$ cannot have a positive element in the j-th index. Hence, $A'_{\text{LP-LB}}(\lambda^+)$ has at most two non-zero entries.

To conclude the proof, it suffices to show that $A'_{\text{LP-LB}}(\lambda^+)$ does not contain more than two non-negative elements. For the sake of contradiction, assume three non-zero elements exist at indices i, j, k s.t. $1 \leq i < j < k \leq K$. Define the vector $\boldsymbol{q} = (q_1, \ldots, q_K)$ as

$$\boldsymbol{q} = \boldsymbol{q}^{(i,j)} - \boldsymbol{q}^{(i,k)}, \tag{20}$$

where $\boldsymbol{q}^{(i,j)}, \boldsymbol{q}^{(i,k)}$ are defined Section 5.2. Clearly, $\boldsymbol{q} \neq \mathbf{0}$ as $q_j, q_k \neq 0$. Also, $\boldsymbol{q}$ satisfies $\boldsymbol{q} \cdot \mathbf{1} = 0$ and $\boldsymbol{q} \cdot \boldsymbol{a} = 0$ based on how $\boldsymbol{q}^{(i,j)}, \boldsymbol{q}^{(i,k)}$ are defined. Then, consider the vector $\boldsymbol{p}$ with step size $w_{\max}$ as

$$\boldsymbol{p} = A'_{\text{LP-LB}}(\lambda^+) - w_{\max} \boldsymbol{q} \operatorname{sign}\left(\boldsymbol{q} \cdot \boldsymbol{\mu}^{-1}\right), \tag{21}$$

$$w_{\max} = \frac{\min\{|A'_{\text{LP-LB}}(\lambda^+)(i)|, |A'_{\text{LP-LB}}(\lambda^+)(j)|, |A'_{\text{LP-LB}}(\lambda^+)(k)|\}}{\max_{\ell:\boldsymbol{q}(\ell)\neq 0}\{|\boldsymbol{q}(\ell)|\}} > 0. \tag{22}$$

Clearly, $\boldsymbol{p}$ satisfies $\boldsymbol{p} \neq A'_{\text{LP-LB}}(\lambda^+)$, $\boldsymbol{p}\cdot\mathbf{1} = A'_{\text{LP-LB}}(\lambda^+)\cdot\mathbf{1} = 1$ and $\boldsymbol{p}\cdot\boldsymbol{a} = A'_{\text{LP-LB}}(\lambda^+)\cdot\boldsymbol{a} \geq a^*$. Further, the definition of $w_{\max}$ implies that $\boldsymbol{p}$ satisfies the third condition in Lemma 10. Finally,

$$\begin{aligned}
\boldsymbol{p}\cdot\boldsymbol{\mu}^{-1} &= A'_{\text{LP-LB}}(\lambda^+)\cdot\boldsymbol{\mu}^{-1} - w_{\max}\boldsymbol{q}\cdot\boldsymbol{\mu}^{-1}\,\text{sign}\left(\boldsymbol{q}\cdot\boldsymbol{\mu}^{-1}\right) && (23)\\
&= A'_{\text{LP-LB}}(\lambda^+)\cdot\boldsymbol{\mu}^{-1} - w_{\max}\left|\boldsymbol{q}\cdot\boldsymbol{\mu}^{-1}\right| && (24)\\
&\leq A'_{\text{LP-LB}}(\lambda^+)\cdot\boldsymbol{\mu}^{-1}, && (25)
\end{aligned}$$

which violates Lemma 10, thus completing the proof. Note, the proof of the statement that $A'_{\text{LP-LB}}(\lambda^+)$ does not contain more than two non-zero elements uses shuffle-pairs. The proof breaks at this stage if shuffle-pairs are not considered. □

7 EXPERIMENTS

In this section, we experimentally demonstrate the performance of the R-JIQ, TAD, and TAD-OP policy and compare it with the lower bound $T_{\text{LP-LB}}(\lambda)$. We provide experiments with fixed arrival rates in Section 7.1. The main goal is to illustrate the performance of the different policies under different load regimes. We provide experiments with variable arrival rates in Section 7.2. Traffic variation is a common phenomenon observed in practical systems, and this section aims to demonstrate the policy's effectiveness in such scenarios.

The experiments presented in Section 7.1 and Section 7.2 share the same configuration as described in Section 3.3. We consider four classes of servers with $\boldsymbol{\mu} = (2, 1, 0.9, 0.1)$, $\boldsymbol{a} = (70, 75, 80, 100)$ and $\alpha_i = 0.25$ for all i. For any plot, solid, dash-dot, and dashed lines represent theoretical bound, arrival-rate aware, and arrival-rate oblivious policies.

7.1 Fixed Arrival Rate

We perform experiments for systems with different load values, system size, and service time distribution. For exponential job size distribution, we considered two extreme system sizes: (i) a small system with $n = 64$, demonstrated by Fig. 6, (ii) a large system with $n = 4096$, demonstrated by Fig. 7. We consider a mid-sized system with $n = 256$ for deterministic service time distribution as illustrated in Fig. 8. For each value of n, we consider load $\frac{\lambda}{\lambda^{\max}} = 1 - \frac{1}{n^\beta}$, where we vary $\beta \in [0.01, 0.495]$. The benchmark accuracy a^* is set according to the caption of each simulation. In each experiment, the queueing system starts with all empty queues and runs until $n \times 10^5$ queries leave the system, with results averaged over 50 runs. In the accuracy-vs-load plots, the lower bound denotes the expression $(\boldsymbol{p}^*(\lambda)\cdot\boldsymbol{a})$.

For smaller systems, as shown in Fig. 6, all proposed policies consistently meet the benchmark accuracy across various load levels. However, neither achieves the optimal latency $T_{\text{LP-LB}}(\lambda)$ as the system size is not large enough to ensure negligible waiting times. Among the three policies, the TAD-OP is the most latency-efficient for any benchmark accuracy. The TAD policy performs decently under low load conditions or low benchmark accuracy. It outperforms R-JIQ, at least at low load, when it does not choose a sub-optimal pair. However, ensuring a positive Δ_{a^*} adversely affects latency at heavy loads, sometimes leading to instability. The R-JIQ policy exhibits moderate performance across all loads and benchmark accuracy a^* and does not have instability issues, as observed in the TAD policy. Fig. 8 illustrates similar performance for mid-sized systems with deterministic service times.

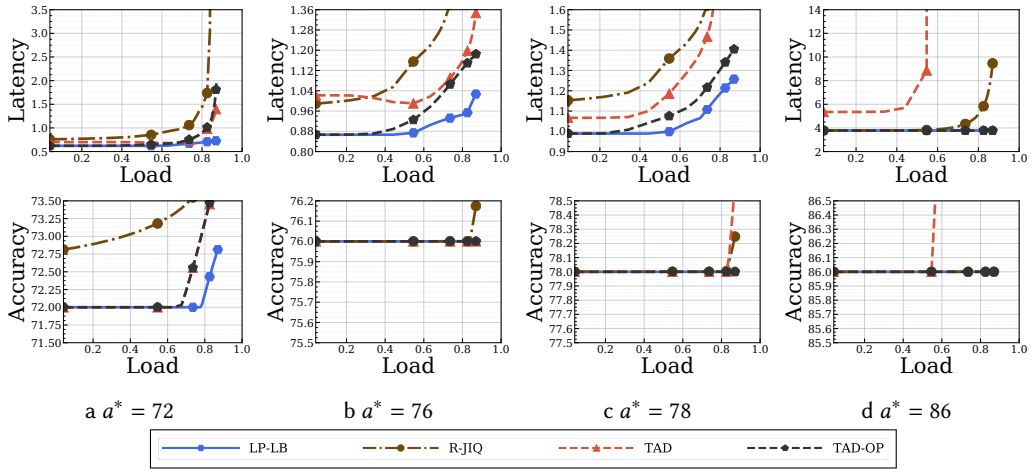

Fig. 6. The accuracy and latency plots as a function of load ρ for different values of the benchmark accuracy a^* for a small system with $n = 64$. All policies achieve the benchmark accuracy for all load values; however, the R-JIQ and TAD policies perform worse than the TAD-OP policy in terms of latency. The TAD policy can sometimes become unstable in the high-load regime since it prioritizes accuracy over idle queues. The R-JIQ policy performs decently in all load regimes and does not face the instability issue; however, it uses the knowledge of λ, unlike the other two policies. The standard deviation for any plotted point is $O(10^{-5})$.

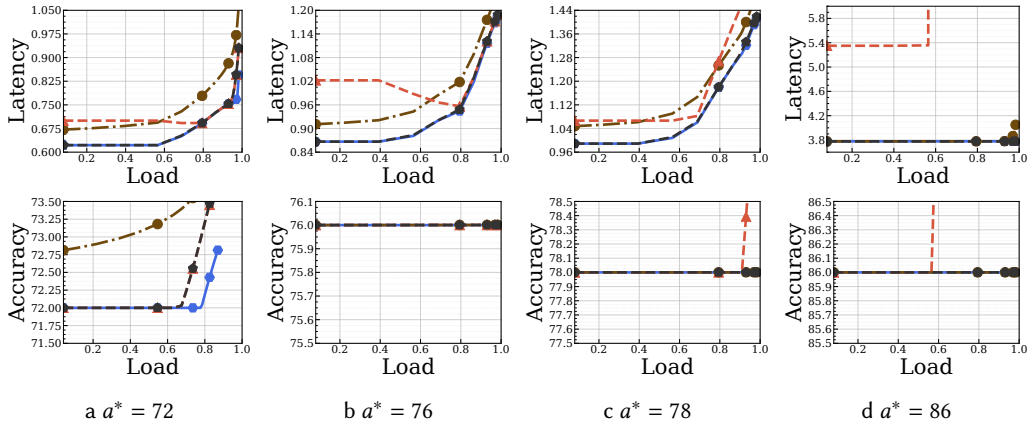

Fig. 7. The accuracy and latency plots as a function of load ρ for different values of the benchmark accuracy a^* for a large system with $n = 4096$. The relative performance of different policies is similar to its performance at small n given in Fig. 6. However, there is a major difference in the performance of the TAD-OP policy, which now achieves the latency lower bound $T_{\text{LP-LB}}(\lambda)$, highlighting the power of ordered pairs.

For large systems, as shown in Fig. 7, all the proposed policies again consistently meet the benchmark accuracy across various load levels. However, the latency performance becomes interesting. The latency of TAD-OP always matches the lower bound $T_{\text{LP-LB}}(\lambda)$ for any choice of system parameters, highlighting the power of ordered pairs. The performance of the R-JIQ policy becomes noticeably better compared to its performance in smaller systems, highlighting asymptotic optimality. Although the performance of the TAD policy has improved in the larger system, it still faces stability problems with extreme system settings.

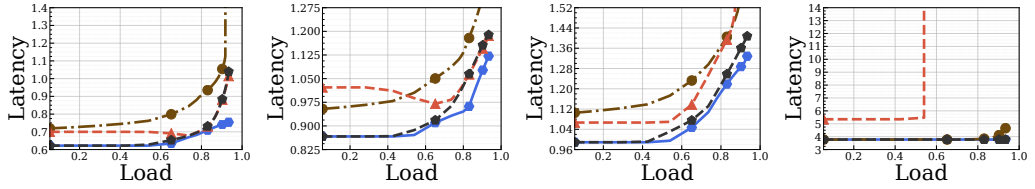

Fig. 8. The latency as a function of load ρ for a mid-sized system ($n = 256$) with deterministic service times.

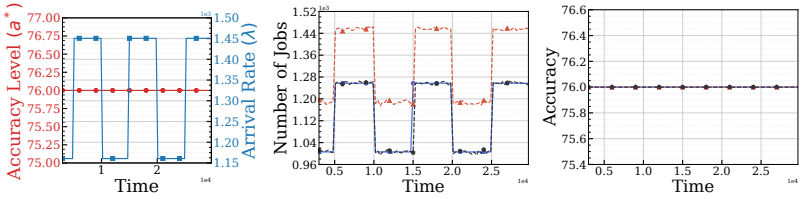

Fig. 9. $\rho_{\text{low}} = 0.4, \rho_{\text{high}} = 0.5, a^*_{\text{low}} = a^*_{\text{high}} = 76$

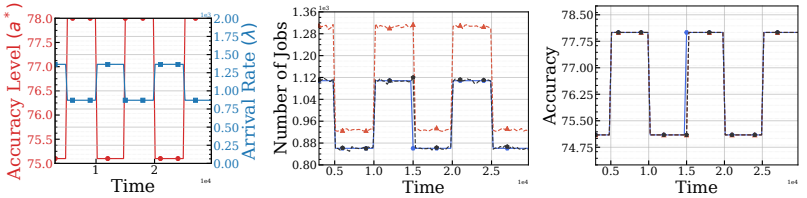

Fig. 10. $\rho_{\text{low}} = \rho_{\text{high}} = 0.4, a^*_{\text{low}} = 75.1, a^*_{\text{high}} = 78$

Fig. 11. Comparison of our TAD and TAD-OP policy under time-varying load or accuracy constraints. The system is characterized using four parameters $\rho_{\text{low}}, \rho_{\text{high}}, a^*_{\text{low}}, a^*_{\text{high}}$. The TAD-OP demonstrates excellent performance even under time-varying load or accuracy criteria and achieves near-optimal performance. The TAD policy achieves the benchmark accuracy but does not achieve the optimum number of queries.

7.2 Variable Arrival Rate

In real-world systems, traffic often sees periodic variations influenced by factors such as time of day or year. For example, Netflix's recommendation system might peak during the evening, while a news recommendation system could experience high activity in the morning hours. We model this variability within the system considering two phases: *low* and *high*. We define four parameters; the load and accuracy during the *low* phase are represented by ρ_{low} and a^*_{low}, respectively, while ρ_{high} and a^*_{high} represent the parameter in the *high* phase. Note that this variability allows us to change the a^* during different phases of the system; this models a situation where an inference system increases its accuracy constraint during certain times of the day to improve user experience.

We perform experiments with $n = 2^{12}$ servers, letting the system run until 3×10^5 units of time, and taking samples at every 10^3-th unit. The accuracy plot represents the running average accuracy of the last 10^3 queries entering the system, and our results are averaged over 500 individual runs. The response time is less intuitive under varying loads; hence, we plot the number of queries within each system. We compare it with the lower bound given by $nN_{\text{LP-LB}}(\lambda)$, where $N_{\text{LP-LB}}(\lambda)$ represents the optimal normalized steady-state number of queries in the system. For varying arrival

rates, the TAD and TAD-OP work as usual. However, when the accuracy constraint a^* changes, the calculation of the routing tuples changes. This requires the TAD-OP policy to use both $\mathcal{S}^*_{\text{opt-low}}$ and $\mathcal{S}^*_{\text{opt-high}}$, the different optimal sets for the different phases of the system. Similarly, both TAD and TAD-OP modify their Δ_{a^*} calculations based on the instantaneous phase of the system.

Fig. 9 and Fig. 10 illustrate the performance of the TAD and TAD-OP policy under varying system parameters. The R-JIQ policy is not included in these experiments due to its dependence on λ. One thing to note in Fig. 10 is that although the load is fixed, the arrival rate changes; this is because changing a^* changes $\lambda^{\max}$ and subsequently the arrival rate λ. The results in both experiments resemble those of the fixed arrival-rate experiments. Both policies are excellent at adapting to variable system parameters. The TAD-OP policy exhibits excellent performance under varying system load or accuracy requirements. The TAD policy achieves the benchmark accuracy a^* but fails to achieve optimality w.r.t. the number of queries in the system.

8 CONCLUSION

In this paper, we introduce the problem of minimizing latency in online inference systems under accuracy constraints. We establish a lower bound, $T_{\text{LP-LB}}(\lambda)$, on the minimum achievable latency for any policy that ensures an expected accuracy of a^* using a linear programming (LP) formulation. Capitalizing on this lower bound, we introduce the R-JIQ policy, which always achieves the benchmark accuracy a^* and asymptotically achieves the optimal latency $T_{\text{LP-LB}}(\lambda)$. However, a limitation of the R-JIQ policy is its dependence on knowing the arrival rate, λ. Even a minor mis-estimation of λ drastically impairs its performance and stability. To address this shortcoming, we propose the TAD policy, an empirical approach to remove the dependency on λ. While the TAD policy always meets the benchmark accuracy and performs decently under moderate loads, it sometimes fails to achieve optimal latency. Notably, under high loads, the TAD policy is vulnerable to instability. We then introduce the concept of pairs and present the waterfilling algorithm, an iterative method to solve the LP using pairs. Drawing inspiration from the waterfilling algorithm, we formulate the TAD-OP policy, which modifies the TAD policy using ordered pairs. Empirical evaluations reveal that the TAD-OP policy performs excellently, particularly in large systems, achieving near-optimal performance. Finally, the policy's ability to operate without knowledge of the arrival rate and to decipher a solution to the LP without full parameter awareness positions it as a highly suitable choice for real systems subjected to fluctuating loads.

8.1 Future Directions

There are several directions for future research. The primary task is to provide theoretical guarantees for the TAD-OP policy. We provide theoretical intuition through the waterfilling algorithm and empirical validations; however, there is a lack of theoretical understanding of the TAD-OP policy. Characterizing its latency is challenging due to the complex dynamics of the Δ_{a^*} variable. Another promising avenue is the generalization of the pair concept. We introduced pairs to handle a single linear constraint; as a generalization for systems with multiple constraints, pairs may be extended to triplets, quadruples, or even larger groupings. For instance, an inference system might need to maintain an accuracy constraint and an energy budget.

In addition, integrating pairs with the R-JIQ policy presents opportunities. Using the routing vector directly to route queries is an avenue to explore, but handling the negative routing vector remains challenging. Some preliminary experiments with query cancellation and rerouting have shown potential, but the practical implementation of such a policy is messy. Finding other methods to address the negative routing vector remains a direction for future research.

REFERENCES

[1] K. G. Binmore. 1982. *Mathematical Analysis: A Straightforward Approach* (2nd ed.). Cambridge University Press.

[2] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. 2017. Adaptive Neural Networks for Efficient Inference. In *Int. Conf. Machine Learning (ICML)* (Sydney, NSW, Australia). JMLR.org, 527–536.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances Neural Information Processing Systems (NEURIPS)*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. 1877–1901.

[4] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *USENIX Symp. Networked Systems Design and Implem. (NSDI)*. USENIX Association, Boston, MA, 613–627.

[5] Mark Van der Boor, Sem C. Borst, Johan S. H. Van Leeuwaarden, and Debankur Mukherjee. 2022. Scalable Load Balancing in Networked Systems: A Survey of Recent Advances. *SIAM Rev.* 64, 3 (2022), 554–622.

[6] A. Ephremides, P. Varaiya, and J. Walrand. 1979. *A Simple Dynamic Routing Problem.* Technical Report UCB/ERL M79/37. EECS Department, University of California, Berkeley.

[7] Jacques Gauvin and Robert Janin. 1988. Directional Behaviour of Optimal Solutions in Nonlinear Mathematical Programming. *Mathematics of Operations Research* 13, 4 (1988), 629–649.

[8] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. 2021. Knowledge Distillation: A Survey. *Int. J. Comput. Vision* 129, 6 (jun 2021), 1789–1819.

[9] I-Hong Hou and P. R. Kumar. 2010. Utility Maximization for Delay Constrained QoS in Wireless. In *Proc. IEEE Int. Conf. Computer Communications (INFOCOM)*. 1–9.

[10] I-Hong Hou and P. R. Kumar. 2010. Utility-Optimal Scheduling in Time-Varying Wireless Networks with Delay Constraints. In *ACM Int. Symp. Mobile Ad Hoc Networking and Computing (MobiHoc)*. Association for Computing Machinery, New York, NY, USA, 31–40.

[11] Navendu Jain, Ishai Menache, Joseph (Seffi) Naor, and Jonathan Yaniv. 2015. Near-Optimal Scheduling Mechanisms for Deadline-Sensitive Jobs in Large Computing Clusters. *ACM Trans. Parallel Comput.* 2, 1 (April 2015), 29 pages.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances Neural Information Processing Systems (NEURIPS)*, Vol. 25.

[13] Bin Li, Atilla Eryilmaz, and Ruogu Li. 2013. Wireless scheduling for network utility maximization with optimal convergence speed. In *Proc. IEEE Int. Conf. Computer Communications (INFOCOM)*. 2112–2120.

[14] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. 2021. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing* 461 (2021), 370–403.

[15] Xin Liu and Lei Ying. 2019. A Simple Steady-State Analysis of Load Balancing Algorithms in the Sub-Halfin-Whitt Regime. *Proc. ACM SIGMETRICS Int. Conf. Measurement and Modeling of Computer Systems* 46, 2 (Jan. 2019), 15–17.

[16] Yi Lu, Qiaomin Xie, Gabriel Kliot, Alan Geller, James R. Larus, and Albert Greenberg. 2011. Join-Idle-Queue: A novel load balancing algorithm for dynamically scalable web services. *Performance Evaluation* 68, 11 (Nov. 2011), 1056–1071.

[17] Michael Mitzenmacher. 1996. Load balancing and density dependent jump Markov processes. In *Proc. Conf. Found. of Comput. Sci.* IEEE, Burlington, VT, USA, 213–222.

[18] Vinod Nigade, Pablo Bauszat, Henri Bal, and Lin Wang. 2022. Jellyfish: Timely Inference Serving for Dynamic Edge Networks. In *2022 IEEE Real-Time Systems Symposium (RTSS)*. 277–290.

[19] Mary Phuong and Christoph Lampert. 2019. Towards Understanding Knowledge Distillation. In *Proceedings of Machine Learning Research (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 5142–5151.

[20] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-less Inference Serving. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 397–411.

[21] Francisco Romero, Mark Zhao, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. Llama: A Heterogeneous & Serverless Framework for Auto-Tuning Video Analytics Pipelines. In *Proceedings of the ACM Symposium on Cloud Computing*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3472883.3486972

[22] Sanjay Shakkottai and R. Srikant. 2002. Scheduling Real-Time Traffic with Deadlines over a Wireless Channel. *Wirel. Netw.* 8, 1 (Jan. 2002), 13–26.

[23] Haichen Shen, Lequn Chen, Yuchen Jin, Liangyu Zhao, Bingyu Kong, Matthai Philipose, Arvind Krishnamurthy, and Ravi Sundaram. 2019. Nexus: A GPU Cluster Engine for Accelerating DNN-Based Video Analysis. In *Proc. ACM Symp. Operating Systems Principles (SOSP)*. Association for Computing Machinery, New York, NY, USA, 322–337.

[24] Alexander L. Stolyar. 2015. Pull-Based Load Distribution in Large-Scale Heterogeneous Service Systems. *Queueing Syst. Theory Appl.* 80, 4 (aug 2015), 341–361.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances Neural Information Processing Systems (NEURIPS)*, Vol. 30. Curran Associates, Inc.

[26] Nikita Dmitrievna Vvedenskaya, Roland L'vovich Dobrushin, and Fridrikh Izrailevich Karpelevich. 1996. Queueing system with selection of the shortest of two queues: An asymptotic approach. *Problemy Peredachi Informatsii* 32, 1 (1996), 20–34.

[27] Chunpu Wang, Chen Feng, and Julian Cheng. 2018. Distributed Join-the-Idle-Queue for Low Latency Cloud Services. *IEEE/ACM Trans. Netw.* 26, 5 (oct 2018), 2309–2319.

[28] Wentao Weng, Xingyu Zhou, and R. Srikant. 2021. Optimal Load Balancing with Locality Constraints. *Proc. ACM Meas. Anal. Comput. Syst.* 4, 3, Article 45 (June 2021), 37 pages.

[29] Wayne L. Winston. 1977. Optimality of the shortest line discipline. *Journal of Applied Probability* 14 (1977), 181 – 189.

[30] Yongji Wu, Matthew Lentz, Danyang Zhuo, and Yao Lu. 2022. Serving and Optimizing Machine Learning Workflows on Heterogeneous Infrastructures. *Proc. VLDB Endow.* 16, 3 (2022), 406–419.

[31] Gingfung Yeung, Damian Borowiec, Renyu Yang, Adrian Friday, Richard Harper, and Peter Garraghan. 2022. Horus: Interference-Aware and Prediction-Based Scheduling in Deep Learning Systems. *IEEE Trans. Parallel Distrib. Syst.* 33, 1 (2022), 88–100.

A PROOF OF LEMMA 2

Recall Lemma 2

Lemma 2 (Asymptotic optimality of the R-JIQ algorithm)**.** *For the policy R-JIQ, and for any* $\frac{\lambda}{\lambda^{\max}} \leq 1 - \frac{1}{n^\beta}, 0 < \beta < \frac{1}{2}, \gamma = \frac{1}{2}\left(\frac{1}{2} - \beta\right)$ *and queues with limited buffer* $b = o(\sqrt{\ln n})$*, the following steady-state properties hold for large enough* n*.*

(1) The expected response time:

$$\mathbb{E}\left[T_{R\text{-}JIQ}\right] \leq \left(1 - \frac{1}{n^\gamma}\right) T_{LP\text{-}LB}(\lambda) + O\left(\min\left\{\frac{1}{n^\gamma}, \frac{1}{\sqrt{n}\ln n}\right\}\right). \tag{5}$$

(2) The expected achieved accuracy:

$$\mathbb{E}[a_{R\text{-}JIQ}] \geq a^*. \tag{6}$$

(3) The blocking probability:

$$\mathbb{E}[p^B_{R\text{-}JIQ}] \leq O\left(\frac{\ln n}{\sqrt{n}}\right). \tag{7}$$

Proof. The proof of Lemma 2 depends primarily on Lemma 11 taken from [15].

Lemma 11. *For a queueing system with* n *unit rate servers with queues at the servers having buffer length* $b = o(\sqrt{\ln n})$*, arrival rate* λn *with* $\lambda = 1 - \frac{1}{n^\beta}, 0 < \beta < 0.5, r = 1 - \frac{1}{2(b-1)}$*, we have* M*, the steady number of jobs in the system under the JIQ policy is bounded as follows.*

$$\mathbb{E}\left[\max\left\{\frac{M}{n} - \lambda - \frac{r\ln n}{n}, 0\right\}\right] \leq \frac{29b}{\sqrt{n}\ln n} \tag{26}$$

for n *sufficiently large. Further, the blocking probability is bounded as*

$$\mathbb{E}[p^B] \leq O\left(\frac{\ln n}{\sqrt{n}}\right). \tag{27}$$

Under the R-JIQ policy, consider K subsystems with sub-system i containing all class-i servers. The subsystem i sees arrival according to a Poisson process with rate $\Lambda p^*_{\text{R-JIQ},1}(\lambda)$. Clearly,

$$\begin{aligned}
\frac{\Lambda p^*_{\text{R-JIQ},i}(\lambda)}{\alpha_i\mu_i n} &= \frac{\lambda p^*_{\text{R-JIQ},i}(\lambda)}{\alpha_i\mu_i} && (28)\\
&= \frac{\lambda}{\alpha_i\mu_i}\left(\left(1 - \frac{1}{n^\gamma}\right)p^*_i(\lambda) + \frac{1}{n^\gamma}p^*_i(\lambda^{\max})\right) && (29)\\
&= \frac{\lambda p^*_i(\lambda)}{\alpha_i\mu_i}\left(1 - \frac{1}{n^\gamma}\right) + \frac{1}{n^\gamma}\frac{\lambda}{\lambda^{\max}}\frac{\lambda^{\max}p^*_i(\lambda^{\max})}{\alpha_i\mu_i} && (30)\\
&\leq 1 - \frac{1}{n^\gamma} + \frac{1}{n^\gamma}\left(1 - \frac{1}{n^\beta}\right) && (31)\\
&= 1 - \frac{1}{n^{\gamma+\beta}} && (32)\\
&= 1 - \frac{1}{n^{\beta'}} && (33)
\end{aligned}$$

where $\beta' = \frac{1}{2}(0.5 + \beta) < 0.5$. Further (31) follows from the fact that $\lambda p^*_i(\lambda) \leq \alpha_i\mu_i$ based on the definition of LP and $\frac{\lambda}{\lambda^{\max}} = 1 - \frac{1}{n^\beta}$. Hence, subsystem i is in the sub-Halfin-Whitt regime. Using

Lemma 11, M_i, the steady-state number of jobs in subsystem i is bounded as

$$\mathbb{E}[M_i] \leq n\alpha_i \left(\frac{\lambda p^*_{\text{R-JIQ},i}(\lambda)}{\alpha_i \mu_i} + \frac{r \ln(\alpha_i n)}{\alpha_i n} + \frac{29b}{\sqrt{\alpha_i n}\ln(\alpha_i n)} \right) \tag{34}$$

$$= n\lambda \frac{p^*_{\text{R-JIQ},i}(\lambda)}{\mu_i} + \left(r \ln(\alpha_i n) + \frac{29b\sqrt{\alpha_i n}}{\ln(\alpha_i n)} \right) \tag{35}$$

$$= n\lambda \frac{p^*_{\text{R-JIQ},i}(\lambda)}{\mu_i} + \max_i \left\{ r \ln(\alpha_i n) + \frac{29b\sqrt{\alpha_i n}}{\ln(\alpha_i n)} \right\} \tag{36}$$

$$\tag{37}$$

Taking a summation over all indices i, we get

$$\mathbb{E}\left[M_{\text{R-JIQ}}\right] = \sum_{i=1}^{K} \mathbb{E}[M_i] \tag{38}$$

$$\leq \sum_{i=1}^{K} \left[n\lambda \frac{p^*_{\text{R-JIQ},i}(\lambda)}{\mu_i} + \max_i \left\{ r \ln(\alpha_i n) + \frac{29b\sqrt{\alpha_i n}}{\ln(\alpha_i n)} \right\} \right] \tag{39}$$

$$= n\lambda \boldsymbol{p}^*_{\text{R-JIQ}}(\lambda) \cdot \boldsymbol{\mu}^{-1} + K \max_i \left\{ r \ln(\alpha_i n) + \frac{29b\sqrt{\alpha_i n}}{\ln(\alpha_i n)} \right\} \tag{40}$$

$$= n\lambda \left(\left(1 - \frac{1}{n^\gamma}\right) \boldsymbol{p}^*(\lambda) + \frac{1}{n^\gamma} \boldsymbol{p}^*(\lambda^{\max}) \right) \cdot \boldsymbol{\mu}^{-1} + K \max_i \left\{ r \ln(\alpha_i n) + \frac{29b\sqrt{\alpha_i n}}{\ln(\alpha_i n)} \right\} \tag{41}$$

$$= n\lambda \left(1 - \frac{1}{n^\gamma}\right) \boldsymbol{p}^*(\lambda) \cdot \boldsymbol{\mu}^{-1} + n^{1-\gamma} \lambda \boldsymbol{p}^*(\lambda^{\max}) \cdot \boldsymbol{\mu}^{-1} + K \max_i \left\{ r \ln(\alpha_i n) + \frac{29b\sqrt{\alpha_i n}}{\ln(\alpha_i n)} \right\} \tag{42}$$

$$= n\lambda \left(1 - \frac{1}{n^\gamma}\right) T_{\text{LP-LB}}(\lambda) + n^{1-\gamma} \lambda \boldsymbol{p}^*(\lambda^{\max}) \cdot \boldsymbol{\mu}^{-1} + K \max_i \left\{ r \ln(\alpha_i n) + \frac{29b\sqrt{\alpha_i n}}{\ln(\alpha_i n)} \right\}. \tag{43}$$

Using Little's law, we have that $T_{\text{R-JIQ}}$, the mean response time of the system is bounded as

$$\mathbb{E}\left[T_{\text{R-JIQ}}\right] = \frac{\mathbb{E}\left[M_{\text{R-JIQ}}\right]}{\lambda n} \tag{44}$$

$$\leq \left(1 - \frac{1}{n^\gamma}\right) T_{\text{LP-LB}}(\lambda) + n^{-\gamma} \boldsymbol{p}^*(\lambda^{\max}) \cdot \boldsymbol{\mu}^{-1} + \frac{K}{\lambda n} \max_i \left\{ r \ln(\alpha_i n) + \frac{29b\sqrt{\alpha_i n}}{\ln(\alpha_i n)} \right\} \tag{45}$$

$$= \left(1 - \frac{1}{n^\gamma}\right) T_{\text{LP-LB}}(\lambda) + O\left(\min\left\{ n^{-\gamma}, \frac{1}{\sqrt{n}\ln n} \right\} \right). \tag{46}$$

The result on blocking probability follows directly from Lemma 11. Finally, the accuracy achieved by the R-JIQ policy is bounded as

$$\mathbb{E}[a_{\text{R-JIQ}}] = \boldsymbol{p}^*_{\text{R-JIQ}}(\lambda) \cdot \boldsymbol{a} \tag{47}$$

$$\geq \left(1 - \frac{1}{n^\gamma}\right) \boldsymbol{p}^*(\lambda) \cdot \boldsymbol{a} + \frac{1}{n^\gamma} \boldsymbol{p}^*(\lambda^{\max}) \cdot \boldsymbol{a} \tag{48}$$

$$\geq a^* \tag{49}$$

as $\boldsymbol{p}^*(\lambda) \cdot \boldsymbol{a} \geq a^*$ for any $\lambda \leq \lambda^{\max}$. □

B PROOF OF LEMMAS USED IN THEOREM 1

B.1 Proof of Lemma 3

Proof. Recall Lemma 3.

Lemma 3 (Optimality of two servers)**.** *For the relaxed linear program, defined in* (8)*, we have the following: for all $\lambda > 0$, the optimal solution $\bar{\boldsymbol{p}}^* = (\bar{p}_1^*, \bar{p}_2^*, \ldots, \bar{p}_K^*)$ is either $\bar{p}_i^* = 1$ with $a_i = a^*$ or $\bar{\boldsymbol{p}}^* = \boldsymbol{q}^{(i_1,j_1)}$ with*

$$i_1, j_1 = \underset{i,j:a_i<a^*<a_j}{\arg\min} \left(\frac{1}{\mu_i} \frac{a_j - a^*}{a_j - a_i} + \frac{1}{\mu_j} \frac{a^* - a_i}{a_j - a_i} \right) \tag{9}$$

where $q_i^{(i,j)} = \frac{a_j - a^*}{a_j - a_i}, q_j^{(i,j)} = \frac{a^* - a_i}{a_j - a_i}$ and $q_k^{(i,j)} = 0$ for any $k \neq i, j$.

We define a probability vector to be *feasible* if it satisfies all the constraints of the relaxed LP, defined in (8). The key idea is to decompose any feasible probability vector as a convex combination of some predefined probabilities vectors, specifically $\boldsymbol{q}^{(i,j)}$ for all $i < j$ s.t. $a_i < a^* < a_j$. Let P_{feasible} and P_{decomp} denote the set of vectors of the form

$$P_{\text{feasible}} = \left\{ \boldsymbol{p} : \boldsymbol{p} \in \mathbb{R}^K, \boldsymbol{p} \geq \boldsymbol{0}, \boldsymbol{p} \cdot \boldsymbol{1} = 1, \boldsymbol{p} \cdot \boldsymbol{a} \geq a^* \right\}, \tag{50}$$

and

$$P_{\text{decomp}} = \Bigg\{ \boldsymbol{p} : \boldsymbol{p} \in \mathbb{R}^K, \boldsymbol{p} = \sum_{i,j:a_i<a^*<a_j} x_{(i,j)} \boldsymbol{q}^{(i,j)} + \sum_{i:a_i \geq a^*} y_{(i)} \boldsymbol{e}_i, \\ x_{(i,j)}, y_{(i)} \geq 0, \sum_{i,j:a_i<a^*<a_j} x_{(i,j)} + \sum_{i:a_i \geq a^*} y_{(i)} = 1 \Bigg\}, \tag{51}$$

respectively. It happens that the two above-defined sets are exactly the same, i.e., $P_{\text{feasible}} = P_{\text{decomp}}$. We prove this fact later in the section. Using the equality of P_{feasible} and P_{decomp}, we can rewrite the relaxed linear programming problem $T_{\text{Relaxed-LP-LB}}(\lambda)$, defined in (8), as

$$\begin{aligned} T_{\text{Relaxed-LP-LB}}(\lambda) = \min_{x_{(i,j)}, y_{(i)}} \quad & \left(\sum_{i,j:a_i<a^*<a_j} x_{(i,j)} v^{(i,j)} + \sum_{i:a_i \geq a^*} \frac{y_{(i)}}{\mu_i} \right) \\ \text{s.t.} \quad & 0 \leq x_{(i,j)} \leq 1, \text{ for all } i < j, a_i < a^* < a_j \\ & 0 \leq y_{(i)} \leq 1, \text{ for all } i, a_i \geq a^* \\ & \sum_{i,j:a_i<a^*<a_j} x_{(i,j)} + \sum_{i:a_i \geq a^*} y_{(i)} = 1. \end{aligned} \tag{52}$$

where

$$v^{(i,j)} = \boldsymbol{q}^{(i,j)} \cdot \boldsymbol{\mu}^{-1}. \tag{53}$$

Clearly, the cost of the linear program is

$$T_{\text{Relaxed-LP-LB}}(\lambda) = \min \left\{ \min_{i,j:a_i<a^*<a_j} \{v^{(i,j)}\}, \min_{i:a_i \geq a^*} \left\{ \frac{1}{\mu_i} \right\} \right\} = \min \left\{ v^{(i_1,j_1)}, \min_{i:a_i \geq a^*} \left\{ \frac{1}{\mu_i} \right\} \right\} \tag{54}$$

with the optimal solution corresponding to assigning the weight $x_{(i,j)}$ or $y_{(i)}$, that attains the minima, to one. Furthermore, an observation to note is that for any $i < j$ s.t. $a_i < a^* < a_j$, $v^{(i,j)}$ is a convex combination of $1/\mu_i$ and $1\mu_j$ and $\mu_i \geq \mu_j$ that further implies $v^{(i_1,j_1)} \leq v^{(i,j)} \leq 1/\mu_j$. Hence, the optimal cost of the relaxed LP is either $T_{\text{Relaxed-LP-LB}}(\lambda) = v^{(i_1,j_1)}$ with the solution being $\bar{\boldsymbol{p}}^* = \boldsymbol{q}^{(i_1,j_1)}$ or it is $T_{\text{Relaxed-LP-LB}}(\lambda) = 1/\mu_i$, satisfying $a_i = a^*$, with the solution being $\bar{\boldsymbol{p}}^* = \boldsymbol{e}_i$.

To complete the proof, it remains to prove that $P_{\text{decomp}} = P_{\text{feasible}}$. We first show that $P_{\text{decomp}} \subseteq P_{\text{feasible}}$. It is easy to verify that $\boldsymbol{q}^{(i,j)} \geq \mathbf{0}, \boldsymbol{q}^{(i,j)} \cdot \mathbf{1} = 1$ and $\boldsymbol{q}^{(i,j)} \cdot \boldsymbol{a} = a^*$ for any $i < j$ and $a_i < a^* < a_j$. Hence, for any $\boldsymbol{p} \in P_{\text{decomp}}$, we have $\boldsymbol{p} \geq 0$,

$$\boldsymbol{p} \cdot \mathbf{1} = \sum_{i,j:a_i<a^*<a_j} x_{(i,j)} \boldsymbol{q}^{(i,j)} \cdot \mathbf{1} + \sum_{i:a_i \geq a^*} y_{(i)} \boldsymbol{e}_i \cdot \mathbf{1} \tag{55}$$

$$= \sum_{i,j:a_i<a^*<a_j} x_{(i,j)} + \sum_{i:a_i \geq a^*} y_{(i)} \tag{56}$$

$$= 1, \tag{57}$$

and

$$\boldsymbol{p} \cdot \boldsymbol{a} = \sum_{i,j:a_i<a^*<a_j} x_{(i,j)} \boldsymbol{q}^{(i,j)} \cdot \boldsymbol{a} + \sum_{i:a_i \geq a^*} y_{(i)} \boldsymbol{e}_i \cdot \boldsymbol{a} \tag{58}$$

$$= \sum_{i,j:a_i<a^*<a_j} x_{(i,j)} a^* + \sum_{i:a_i \geq a^*} y_{(i)} a_i \tag{59}$$

$$\geq a^* \left(\sum_{i,j:a_i<a^*<a_j} x_{(i,j)} + \sum_{i:a_i \geq a^*} y_{(i)} \right) \tag{60}$$

$$= a^*. \tag{61}$$

Hence, $P_{\text{decomp}} \subseteq P_{\text{feasible}}$.

Next, we prove that $P_{\text{feasible}} \subseteq P_{\text{decomp}}$. To prove this, consider any $\boldsymbol{p} = (p_1, \ldots, p_K) \in P_{\text{feasible}}$. Note that we can write $\boldsymbol{p}$ as

$$\boldsymbol{p} = \sum_{i,j:a_i<a^*<a_j} x_{(i,j)} \boldsymbol{q}^{(i,j)} + \sum_{i=1}^{K} y_{(i)} \boldsymbol{e}_i \tag{62}$$

where $y_{(i)} = p_i$ for all $i = 1, \ldots, K$ and $x_{(i,j)} = 0$ for i, j s.t. $a_i < a^* < a_j$. The initial decomposition of $\boldsymbol{p}$ in (62) is similar to the form of decomposition that exists for probability vectors in P_{decomp}. The only difference lies in the indices of the second summation term, i.e., the summation changed from $\sum_{i:a_i \geq a^*}$ to $\sum_{i=1}^{K}$.

For the provided decomposition, we implement a recursive algorithm to transform $\boldsymbol{p}$ into a form that ensures it lies in P_{decomp}. At every step, if there exists i, j s.t. $y_{(i)}, y_{(j)} > 0$ and $a_i < a^* < a_j$, one can define $x'_{(i,j)} = \sup \left\{ w : w > 0, w q_i^{(i,j)} \leq y_{(i)}, w q_j^{(i,j)} \leq y_{(j)} \right\}$ and update the decomposition as $x_{(i,j)} = x_{(i,j)} + x'_{(i,j)}$, $y_{(i)} = y_{(i)} - x'_{(i,j)} q_i^{(i,j)}$ and $y_{(j)} = y_{(j)} - x'_{(i,j)} q_j^{(i,j)}$. The update would essentially change either $y_{(i)}$ or $y_{(j)}$ or both to zero while ensuring that the total sum of weights always equals one, i.e., $\sum_{i,j:a_i<a^*<a_j} x_{(i,j)} + \sum_{i=1}^{K} y_{(i)} = 1$.

This recursion would eventually stop when $y_{(i)} = 0$ for all i satisfying $a_i < a^*$ or $a_i > a^*$. However, if the final transformation is of the form $\boldsymbol{p} = \sum_{i,j:a_i<a^*<a_j} x_{(i,j)} \boldsymbol{q}^{(i,j)} + \sum_{i:a_i<=a^*} y_{(i)} \boldsymbol{e}_i$ with at least one index $\bar{i}$ s.t. $y_{\bar{i}} > 0$ and $a_{\bar{i}} < a^*$, then

$$\boldsymbol{p} \cdot \boldsymbol{a} = \sum_{i,j:a_i<a^*<a_j} x_{(i,j)} \boldsymbol{q}^{(i,j)} \cdot \boldsymbol{a} + \sum_{i:a_i<=a^*} y_{(i)} \boldsymbol{e}_i \cdot \boldsymbol{a} \tag{63}$$

$$= \sum_{i,j:a_i<a^*<a_j} x_{(i,j)} a^* + \sum_{i:a_i<=a^*} y_{(i)} a_i \tag{64}$$

$$= \sum_{i,j:a_i<a^*<a_j} x_{(i,j)} a^* + \sum_{i:a_i<=a^*, i \neq \bar{i}} y_{(i)} a_i + y_{\bar{i}} a_{\bar{i}} \tag{65}$$

$$< \sum_{i,j:a_i<a^*<a_j} x_{(i,j)} a^* + \sum_{i:a_i<=a^*, i\neq\bar{i}} y_{(i)} a^* + y_{\bar{i}} a^* \tag{66}$$

$$= a^* \left(\sum_{i,j:a_i<a^*<a_j} x_{(i,j)} + \sum_{i:a_i<=a^*} y_{(i)} \right) \tag{67}$$

$$= a^*, \tag{68}$$

which violates the condition $\boldsymbol{p} \cdot \boldsymbol{a} \geq a^*$. This proves that the final transformation has to be of the form $\boldsymbol{p} = \sum_{i,j:a_i<a^*<a_j} x_{(i,j)} \boldsymbol{q}^{(i,j)} + \sum_{i:a_i \geq a^*} y_{(i)} \boldsymbol{e}_i$ s.t. $x_{(i,j)}, y_{(i)} \geq 0$. Hence, $P_{\text{feasible}} \subseteq P_{\text{decomp}}$ which implies $P_{\text{feasible}} = P_{\text{decomp}}$

□

B.2 Proof of Lemma 4

Proof. Recall Lemma 4.

Lemma 4 (Convexity of LP). *For the LP, defined in* (3)*, and maximum feasible arrival rate* $\lambda^{\max}$*, it holds that*

(1) $\lambda^{\max}$ *is well defined and satisfies* $\lambda^{\max} \in (0, \sum_{i=1}^{K} \alpha_i \mu_i]$*,*
(2) the LP is feasible for all $\lambda \in [0, \lambda^{\max}]$*,*
(3) $N_{LP\text{-}LB}(\lambda)$ *is a continuous convex function w.r.t* λ *for all* $\lambda \in [0, \lambda^{\max}]$*,*
(4) For all $\lambda \in [0, \lambda^{\max})$*, the right derivative of the function* $N_{LP\text{-}LB}(\lambda)$*, given by* $N'_{LP\text{-}LB}(\lambda^+)$*, exists and satisfies* $N'_{LP\text{-}LB}(\lambda^+) \geq 0$*.*

(1) The supremum of a non-empty bounded set of real numbers always exists. Clearly, the linear program, defined in (3), is feasible for $\lambda = \sum_{i:a_i \geq a^*} \alpha_i \mu_i$ with a feasible solution being $\boldsymbol{p} = (p_1, \ldots, p_K)$ with $p_i = \alpha_i \mu_i / (\sum_{a_i \geq a^*} \alpha_i \mu_i) \mathbf{1}_{a_i \geq a^*}$.
Also, for any $\lambda > \sum_{i=1}^{K} \alpha_i \mu_i$, the LP , defined in(3), is infeasible as $\lambda \boldsymbol{p} \leq \boldsymbol{\mu_\alpha}$ implies $\lambda = \lambda \boldsymbol{p} \cdot \mathbf{1} \leq \boldsymbol{\mu_\alpha} \cdot 1 = \sum_{i=1}^{K} \alpha_i \mu_i$. Hence, $\lambda^{\max}$ is well defined and satisfies $0 < \sum_{i:a_i \geq a^*} \alpha_i \mu_i \leq \lambda^{\max} \leq \sum_{i=1}^{K} \alpha_i \mu_i$.
(2) It is easy to verify that $\boldsymbol{p}^*(\lambda^{\max})$ is a feasible solution to the LP, defined in (3), for any $\lambda \in [0, \lambda^{\max}]$.
(3) Take any $0 \leq \lambda_1 < \lambda_2 \leq \lambda^{\max}$ and the corresponding optimal solution of the LP $\boldsymbol{p}^*(\lambda_1)$ and $\boldsymbol{p}^*(\lambda_2)$, respectively. For any λ of the form $\lambda = \alpha\lambda_1 + (1-\alpha)\lambda_2$ for some constant $\alpha \in [0, 1]$, consider the solution $\boldsymbol{p} = \frac{\alpha\lambda_1 \boldsymbol{p}^*(\lambda_1) + (1-\alpha)\lambda_2 \boldsymbol{p}^*(\lambda_2)}{\lambda}$. Clearly, $\boldsymbol{p} \geq \mathbf{0}$,

$$\boldsymbol{p} \cdot \mathbf{1} = \frac{\alpha\lambda_1 \boldsymbol{p}^*(\lambda_1) \cdot \mathbf{1} + (1-\alpha)\lambda_2 \boldsymbol{p}^*(\lambda_2) \cdot \mathbf{1}}{\lambda} \tag{69}$$

$$= \frac{\alpha\lambda_1 + (1-\alpha)\lambda_2}{\lambda} \tag{70}$$

$$= 1, \tag{71}$$

and

$$\boldsymbol{p} \cdot \boldsymbol{a} = \frac{\alpha\lambda_1 \boldsymbol{p}^*(\lambda_1) \cdot \boldsymbol{a} + (1-\alpha)\lambda_2 \boldsymbol{p}^*(\lambda_2) \cdot \boldsymbol{a}}{\lambda} \tag{72}$$

$$\geq \frac{\alpha\lambda_1 a^* + (1-\alpha)\lambda_2 a^*}{\lambda} \tag{73}$$

$$= a^* \frac{\alpha\lambda_1 + (1-\alpha)\lambda_2}{\lambda} \tag{74}$$

$$= a^*. \tag{75}$$

Also,

$$\lambda \boldsymbol{p} = \alpha\lambda_1 \boldsymbol{p}^*(\lambda_1) + (1-\alpha)\lambda_2 \boldsymbol{p}^*(\lambda_2) \tag{76}$$
$$\leq \alpha \boldsymbol{\mu_\alpha} + (1-\alpha)\boldsymbol{\mu_\alpha} \tag{77}$$
$$= \boldsymbol{\mu_\alpha}. \tag{78}$$

Hence, $\boldsymbol{p}$ is a feasible solution to the LP. Finally,

$$N_{\text{LP-LB}}(\lambda) \leq \lambda \boldsymbol{p} \cdot \boldsymbol{\mu}^{-1} \tag{79}$$
$$= (\alpha\lambda_1 \boldsymbol{p}^*(\lambda_1) + (1-\alpha)\lambda_2 \boldsymbol{p}^*(\lambda_2)) \cdot \boldsymbol{\mu}^{-1} \tag{80}$$
$$= \alpha N_{\text{LP-LB}}(\lambda_1) + (1-\alpha) N_{\text{LP-LB}}(\lambda_2), \tag{81}$$

which proves convexity. The continuity follows directly from the convexity of the problem.

(4) A convex function on an open interval is semi-differentiable (see Thm 12.14 in [1]). The convexity of $N_{\text{LP-LB}}(\lambda)$ along with the feasibility in the range $[0, \lambda^{\max}]$ implies that the right derivative $N'_{\text{LP-LB}}(\lambda^+)$ exists for all $\lambda \in [0, \lambda^{\max})$. To prove that the right derivative is non-negative, it suffices to show that the function $N_{\text{LP-LB}}(\lambda)$ attains local minima at $\lambda = 0$. Clearly, $N_{\text{LP-LB}}(0) = 0$ and $N_{\text{LP-LB}}(\lambda) > 0$ for any feasible $\lambda > 0$ by its definition, which completes the proof.

□

B.3 Proof of Lemma 6

Proof.

Lemma 6 (Well Definedness). *The parameters $\lambda^{\max}_{WF\text{-}O}$ and $\lambda^{\max}_{WF\text{-}F}$ are well defined and satisfy,*

$$\lambda^{\max}_{WF\text{-}O} \leq \lambda^{\max}_{WF\text{-}F} \leq \lambda^{\max}. \tag{17}$$

Recall, Lemma 6 The supremum of a nonempty bounded set of real numbers always exists. Clearly, $\lambda^{\max}_{\text{WF-O}}$ and $\lambda^{\max}_{\text{WF-F}}$ are upper bounded by $(\boldsymbol{\mu_\alpha} \cdot \mathbf{1})$ to ensure the feasibility of the algorithm. Hence, to prove well definedness, it suffices to show that there exists at least one λ_1 such that for all $\lambda \in [0, \lambda_1]$, $N_{\text{LP-LB}}(\lambda)$ and $N_{\text{WF}}(\lambda)$ are feasible and $N_{\text{WF}}(\lambda) = N_{\text{LP-LB}}(\lambda)$.

Recall, the definition of i_1, j_1 given in Lemma 3,

$$i_1, j_1 = \underset{i,j: a_i < a^* < a_j}{\arg\min}\ v^{(i,j)}. \tag{82}$$

and define

$$\lambda_1 = \sup\{\lambda' : \lambda' \geq 0, \lambda' \boldsymbol{q}^{(i_1,j_1)} \leq \boldsymbol{\mu_\alpha}\} \tag{83}$$

Observe that the waterfilling algorithm will output $\boldsymbol{p} = \boldsymbol{q}^{(i_1,j_1)}$ with a cost of

$$N_{\text{WF}}(\lambda) = \lambda v^{(i_1,j_1)}, \text{ for all } \lambda \in [0, \lambda_1]. \tag{84}$$

This implies that $N_{\text{WF}}(\lambda)$ is feasible for all $\lambda \in [0, \lambda_1]$ implying $\lambda^{\max}_{\text{WF-F}}$ exists. It is also easy to verify that $N_{\text{LP-LB}}(\lambda)$ is feasible for all $\lambda \in [0, \lambda_1]$ with $\boldsymbol{q}^{(i_1,j_1)}$ being a feasible solution implying

$$N_{\text{LP-LB}}(\lambda) \leq \lambda v^{(i_1,j_1)}, \text{ for all } \lambda \in [0, \lambda_1]. \tag{85}$$

Furthermore, Lemma 3 states that for the relaxed linear program $N_{\text{Relaxed-LP-LB}}(\lambda)$, it is optimal to use the solution $\boldsymbol{q}^{(i_1,j_1)}$ for all $\lambda \geq 0$, i.e.,

$$N_{\text{Relaxed-LP-LB}}(\lambda) = \lambda v^{(i_1,j_1)} \text{ for all } \lambda > 0. \tag{86}$$

Also, for any $\lambda \in [0, \lambda_1]$, the routing set of $N_{\text{LP-LB}}(\lambda)$ is a subset of the routing set of its relaxed version $N_{\text{Relaxed-LP-LB}}(\lambda)$ and both linear programs are feasible implying

$$N_{\text{Relaxed-LP-LB}}(\lambda) \leq N_{\text{LP-LB}}(\lambda), \text{ for all } \lambda \in [0, \lambda_1]. \tag{87}$$

Using, (84), (85), (86) and (87), we have for all $\lambda \in [0, \lambda_1]$

$$N_{\text{WF}}(\lambda) = N_{\text{LP-LB}}(\lambda) = N_{\text{Relaxed-LP-LB}}(\lambda) = \lambda v^{(i_1, j_1)} \tag{88}$$

implying that $\lambda^{\max}_{\text{WF-O}}$ is well defined.

Next, we show that $\lambda^{\max}_{\text{WF-O}} \leq \lambda^{\max}_{\text{WF-F}} \leq \lambda^{\max}$. Clearly, $\lambda^{\max}_{\text{WF-O}} \leq \lambda^{\max}_{\text{WF-F}}$ as the set corresponding to $\lambda^{\max}_{\text{WF-O}}$ is a subset of the one corresponding to $\lambda^{\max}_{\text{WF-F}}$. Hence, it suffices to show that $\lambda^{\max}_{\text{WF-F}} \leq \lambda^{\max}$. This directly follows from the observations that the algorithm always maintains the capacity constraints $\mathbf{0} \leq \lambda \boldsymbol{p} \leq \boldsymbol{\mu}_{\boldsymbol{\alpha}}$ and the routing tuples always maintain the benchmark accuracy a^*. Hence, a feasible solution of the waterfilling algorithm $N_{\text{WF}}(\lambda)$ is always a feasible solution to the original LP $N_{\text{LP-LB}}(\lambda)$. □

B.4 Proof of Lemma 7

Proof. Recall Lemma 7

Lemma 7 (Feasibility equals Optimality). *For all $\lambda \in [0, \lambda^{\max}_{WF\text{-}F}]$, the solution of the waterfilling algorithm is an optimal solution to the LP defined in* (3)*, i.e., $\lambda^{\max}_{WF\text{-}O} = \lambda^{\max}_{WF\text{-}F}$.*

We prove this by using a contradiction. We assume $\lambda^{\max}_{\text{WF-O}} < \lambda^{\max}_{\text{WF-F}}$ and show that the waterfilling algorithm $N_{\text{WF}}(\lambda)$ cannot be suboptimal for any $\lambda \in (\lambda^{\max}_{\text{WF-O}}, \lambda^{\max}_{\text{WF-F}}]$. First consider the solution of the waterfilling algorithm $N_{\text{WF}}(\lambda^{\max}_{\text{WF-F}})$. Let w_i^{WF}s, the weights of the routing vector $\boldsymbol{q}_i^*$ of the sets S_i^*, be the output of the waterfilling algorithm $N_{\text{WF}}(\lambda^{\max}_{\text{WF-F}})$.

We first show that there exists an index $i^* \in \{1, \ldots, |\mathcal{S}^*_{\text{opt}}| - 1\}$ such that $\lambda^{\max}_{\text{WF-O}}$ is of the form

$$\lambda^{\max}_{\text{WF-O}} = \sum_{i=1}^{i^*} w_i^{\text{WF}} \tag{89}$$

We prove this by contradiction. Assume $\lambda^{\max}_{\text{WF-O}} \in (\sum_{i=1}^{i^*} w_i^{\text{WF}}, \sum_{i=1}^{i^*+1} w_i^{\text{WF}})$ for some $i^* \in \{1, \ldots, |\mathcal{S}^*_{\text{opt}}| - 1\}$, i.e., there exists some $\lambda \in (\lambda^{\max}_{\text{WF-O}}, \sum_{i=1}^{i^*+1} w_i^{\text{WF}})$ where $N_{\text{WF}}(\lambda) > N_{\text{LP-LB}}(\lambda)$. The assumption also implies that

$$N_{\text{WF}}(\lambda) = N_{\text{LP-LB}}(\lambda), \text{ for all } \lambda \in \left[\sum_{i=1}^{i^*} w_i^{\text{WF}}, \lambda^{\max}_{\text{WF-O}}\right]. \tag{90}$$

It also implies that the right derivatives of the functions also match, i.e.,

$$N'_{\text{LP-LB}}(\lambda^+) = N'_{\text{WF}}(\lambda^+), \text{ for all } \lambda \in \left[\sum_{i=1}^{i^*} w_i^{\text{WF}}, \lambda^{\max}_{\text{WF-O}}\right) \tag{91}$$

Now, using convexity of $N_{\text{LP-LB}}$, as shown in Lemma 4, we have that for all $\lambda \in [\sum_{i=1}^{i^*} w_i^{\text{WF}}, \sum_{i=1}^{i^*+1} w_i^{\text{WF}})$,

$$N_{\text{LP-LB}}(\lambda) \geq N_{\text{LP-LB}}\left(\sum_{i=1}^{i^*} w_i^{\text{WF}}\right) + \left(\lambda - \sum_{i=1}^{i^*} w_i^{\text{WF}}\right) N'_{\text{LP-LB}}\left(\left(\sum_{i=1}^{i^*} w_i^{\text{WF}}\right)^+\right) \tag{92}$$

$$= N_{\text{WF}}\left(\sum_{i=1}^{i^*} w_i^{\text{WF}}\right) + \left(\lambda - \sum_{i=1}^{i^*} w_i^{\text{WF}}\right) N'_{\text{WF}}\left(\left(\sum_{i=1}^{i^*} w_i^{\text{WF}}\right)^+\right) \tag{93}$$

$$= N_{\text{WF}}(\lambda), \tag{94}$$

where the last step follows from the fact that $N_{\text{WF}}(\lambda)$ is linear for any $\lambda \in [\sum_{i=1}^{i^*} w_i^{\text{WF}}, \sum_{i=1}^{i^*+1} w_i^{\text{WF}})$. Due to the nature of our waterfilling algorithm, the slope of $N_{\text{WF}}(\lambda)$ in the interval is $v_{i^*+1}^*$. This contradicts the initial assumption. The fact that $N'_{\text{WF}}(\lambda^+) \leq N'_{\text{LP-LB}}(\lambda^+)$ for all $\lambda \in [\sum_{i=1}^{i^*} w_i^{\text{WF}}, \sum_{i=1}^{i^*+1} w_i^{\text{WF}})$ explains the phrase *switching* routing tuples *does not help.*

The final statement to argue is that $\lambda_{\text{WF-O}}^{\max}$ cannot be of the form $\lambda_{\text{WF-O}}^{\max} = \sum_{i=1}^{i^*} w_i^{\text{WF}}$ for any constant $i^* \in \{1, \ldots, |\mathcal{S}_{\text{opt}}^*| - 1\}$. For the sake of contradiction, assume there exists an index $i^* \in \{1, \ldots, |\mathcal{S}_{\text{opt}}^*| - 1\}$ s.t. $\lambda_{\text{WF-O}}^{\max} = \sum_{i=1}^{i^*} w_i^{\text{WF}}$. This implies there exists some $\lambda' \in (\lambda_{\text{WF-O}}^{\max}, \sum_{i=1}^{i^*+1} w_i^{\text{WF}})$ s.t. $N_{\text{WF}}(\lambda) > N_{\text{LP-LB}}(\lambda)$ for all $\lambda \in (\lambda_{\text{WF-O}}^{\max}, \lambda')$. Lemma 5 states that the right derivative of the function $A_{\text{LP-LB}}(\lambda)$, given by $A'_{\text{LP-LB}}(\lambda^+)$, is always of the form $\boldsymbol{q}_i^*$ for some i. Clearly, at $\lambda_{\text{WF-O}}^{\max} = \sum_{i=1}^{i^*} w_i^{\text{WF}}$, the only feasible choices are $\boldsymbol{q}_i^*$ for all $i \geq i^* + 1$. Hence, the optimal choice is to use routing set $S_{i^*+1}^*$ with the routing vector $\boldsymbol{q}_{i^*+1}^*$ implying optimality of the waterfilling algorithm.

□

B.5 Proof of Lemma 8

Proof. Recall Lemma 8

Lemma 8 (Waterfilling is Feasible)**.** *For all $\lambda \in [0, \lambda^{\max}]$, the waterfilling algorithm is feasible, i.e., $\lambda_{\text{WF-F}}^{\max} = \lambda^{\max}$.*

We prove this by showing that for any $\lambda > \lambda_{\text{WF-F}}^{\max}$, the linear program, defined in (3), is infeasible.

Observe the solution $\boldsymbol{p} = (p_1, \ldots, p_K)$ provided by the waterfilling algorithm at $\lambda = \lambda_{\text{WF-F}}^{\max}$. Clearly, the solution $\boldsymbol{p}$ satisfies $\lambda_{\text{WF-F}}^{\max} p_i = \alpha_i \mu_i$ for all i satisfying $a_i \geq a^*$. If not, the waterfilling algorithm is feasible for a higher λ since it can push some traffic into the server class $\bar{i}$ that satisfies $\lambda_{\text{WF-F}}^{\max} p_{\bar{i}} < \alpha_{\bar{i}} \mu_{\bar{i}}$ and $a_{\bar{i}} \geq a^*$, contradicting the definition of $\lambda_{\text{WF-F}}^{\max}$.

Next, we argue that one of the following conditions must be true, either of which implies $\lambda_{\text{WF-F}}^{\max} = \lambda^{\max}$

(1) $\lambda_{\text{WF-F}}^{\max} \boldsymbol{p} = \boldsymbol{\mu_\alpha}$: Clearly, $\lambda_{\text{WF-F}}^{\max} = \lambda_{\text{WF-F}}^{\max} \boldsymbol{p} \cdot \mathbf{1} = \boldsymbol{\mu_\alpha} \cdot \mathbf{1}$. However, Lemma 4 states that $\lambda^{\max} \leq \boldsymbol{\mu_\alpha} \cdot \mathbf{1}$ and Lemma 6 states $\lambda_{\text{WF-F}}^{\max} \leq \lambda^{\max}$ implying $\lambda_{\text{WF-F}}^{\max} = \lambda^{\max} = \boldsymbol{\mu_\alpha} \cdot \mathbf{1}$.

(2) There exists an index $\bar{i} \in \{1, \ldots, K\}$ s.t. $a_{\bar{i}} < a^*$ and $\lambda_{\text{WF-F}}^{\max} p_i = 0, \forall i < \bar{i}, \lambda_{\text{WF-F}}^{\max} p_{\bar{i}} \in [0, \alpha_{\bar{i}} \mu_{\bar{i}}), \lambda_{\text{WF-F}}^{\max} p_i = \alpha_i \mu_i, \forall i > \bar{i}$ and $\boldsymbol{p} \cdot \boldsymbol{a} = a^*$: In the given scenario, there do not exist any *properly-paired* or *shuffling-pair* routing tuples, where more traffic can be added. Furthermore, because there exists at least one semi-filled server class $\bar{i}$ with an accuracy less than a^*, the waterfilling algorithm would have only used routing tuples that contain a pair of server classes. Since all such tuples exactly maintain an accuracy of a^*, the system accuracy $\boldsymbol{p} \cdot \boldsymbol{a}$ must be equal to a^*. Therefore, increasing λ beyond $\lambda_{\text{WF-F}}^{\max}$ can only decrease accuracy, making the linear program, defined in (3), infeasible.

□

B.6 Proof of Lemma 9

Recall Lemma 9

Lemma 9. *For all $\lambda \in [0, \lambda^{\max})$, the right derivative $A'_{LP\text{-}LB}(\lambda^+)$ satisfy*

$$A'_{LP\text{-}LB}(\lambda^+) \cdot \mathbf{1} = 1 \tag{18}$$

$$A'_{LP\text{-}LB}(\lambda^+) \cdot \boldsymbol{a} \geq a^* \tag{19}$$

Proof. Clearly, for any $\lambda \in [0, \lambda^{\max}]$

$$A_{\text{LP-LB}}(\lambda) \cdot \mathbf{1} = \lambda. \tag{95}$$

Hence, taking right derivative implies, for any $\lambda \in [0, \lambda^{\max})$,

$$A'_{\text{LP-LB}}(\lambda^+) \cdot \mathbf{1} = 1. \tag{96}$$

Next, define

$$\lambda^{\max}_{a^*} = \sup\{\lambda' : \lambda' \in [0, \lambda^{\max}], \boldsymbol{p}^*(\lambda'') \cdot \boldsymbol{a} = a^*, \text{ for all } \lambda'' \in [0, \lambda']\}. \tag{97}$$

Recall, the definition of i_1, j_1 given in Lemma 3,

$$i_1, j_1 = \underset{i,j:a_i<a^*<a_j}{\arg\min}\ v^{(i,j)}. \tag{98}$$

and define

$$\lambda_1 = \sup\{\lambda' : \lambda' \geq 0, \lambda' \boldsymbol{q}^{(i_1,j_1)} \leq \boldsymbol{\mu\alpha}\} \tag{99}$$

Lemma 3 states that for all $\lambda \in [0, \lambda_1]$, the optimal solution of the LP $N_{\text{LP-LB}}(\lambda)$ equals $\boldsymbol{p}^*(\lambda) = \boldsymbol{q}^{i_1 j_1}$ which satisfies $\boldsymbol{p}^*(\lambda) \cdot \boldsymbol{a} = a^*$. Hence, $\lambda^{\max}_{a^*}$ is well defined. Hence, for any $\lambda \in [0, \lambda^{\max}_{a^*}]$, we have

$$A_{\text{LP-LB}}(\lambda) \cdot \boldsymbol{a} = \lambda \boldsymbol{p}^*(\lambda) \cdot \boldsymbol{a} = \lambda a^*. \tag{100}$$

which implies for all $\lambda \in [0, \lambda^{\max}_{a^*})$, we have

$$A'_{\text{LP-LB}}(\lambda^+) \cdot \boldsymbol{a} = a^*. \tag{101}$$

Next, we show that for $\lambda \in [\lambda^{\max}_{a^*}, \lambda^{\max})$, the gradient function satisfies,

$$A'_{\text{LP-LB}}(\lambda^+) \cdot \boldsymbol{a} > a^*. \tag{102}$$

We first argue that for any $\boldsymbol{p}^*(\lambda) \cdot \boldsymbol{a} > a^*$, there exists an index $\bar{i} \in \{1, \ldots, K\}$ s.t. $a_{\bar{i}} > a^*$, $\lambda p^*_i(\lambda) = \alpha_i \mu_i$ for all $i < \bar{i}$, $\lambda p^*_i(\lambda) = 0$ for all $i > \bar{i}$ and $\lambda p^*_{\bar{i}}(\lambda) \in [0, \alpha_{\bar{i}} \mu_{\bar{i}})$. We prove this by contradiction. Assume there exist indices $i < j$ st $a_i < a_j$ and $\lambda p^*_i(\lambda) < \alpha_i \mu_i$ and $\lambda p^*_j(\lambda) > 0$. If not, then one can increase $p_i(\lambda)$ and decrease $p_j(\lambda)$ to decrease the cost $\lambda \boldsymbol{p}(\lambda) \cdot \boldsymbol{\mu}^{-1}$ while maintaining feasibility. Further, since $\boldsymbol{p}(\lambda) \cdot \boldsymbol{a} > a^*$ there exist at least one index $\bar{i}'$ s.t. $p_{\bar{j}}(\lambda) > 0$ which proves the claim. This also implies that there does not exist any feasible solution $\boldsymbol{p} = (p_1, \ldots, p_K)$ to the LP $N_{\text{LP-LB}}(\lambda)$ that satisfies $\boldsymbol{p} \cdot \boldsymbol{a} = a^*$. This holds as, for any $\lambda \in (\sum_{i=1}^{\bar{i}-1} \mu_i, \sum_{i=1}^{\bar{i}} \mu_i]$ and any valid probability vector $\boldsymbol{p} = (p_1, \ldots, p_K)$, it holds that

$$\sum_i p_i a_i \geq \sum_{i=1}^{\bar{i}-1} \frac{\alpha_i \mu_i}{\lambda} a_i + (\lambda - \sum_{i=1}^{\bar{i}-1} \alpha_i \mu_i) a_{\bar{i}} \tag{103}$$

$$> a^*. \tag{104}$$

The first inequality follows from the fact that we are assigning maximum possible weights to $a_1, \ldots, a_{\bar{i}}$ and a_i are monotonically increasing, while the second inequality follows from the fact that the assigned weight p_i matches with the optimal probability vector $\boldsymbol{p}^*(\lambda)$ which attains an accuracy greater than a^*. Hence, for any λ s.t. $\boldsymbol{p}^*(\lambda) \cdot \boldsymbol{a} > a^*$, there does not exists any feasible solution $\boldsymbol{p}$ that satisfies $\boldsymbol{p} \cdot \boldsymbol{a} = a^*$.

This implies for any $\lambda > \lambda^{\max}_{a^*}$, $\boldsymbol{p}^*(\lambda) \cdot \boldsymbol{a} > \lambda a^*$. We can prove this using contradiction. Assume there exists a $\lambda' \in (\lambda^{\max}_{a^*}, \lambda^{\max}]$ s.t. $\boldsymbol{p}^*(\lambda') \cdot \boldsymbol{a} = a^*$. However, $\boldsymbol{p}^*(\lambda')$ is a feasible solution to the LP $N_{\text{LP-LB}}(\lambda)$ for all $\lambda \in [\lambda^{\max}_{a^*}, \lambda')$. Using previous argument, this implies $\boldsymbol{p}^*(\lambda) \cdot \boldsymbol{a} = a^*$ for all $\lambda \in [\lambda^{\max}_{a^*}, \lambda')$ which violates the definition of $\lambda^{\max}_{a^*}$. Hence, for any $\lambda > \lambda^{\max}_{a^*}$, $\boldsymbol{p}^*(\lambda) \cdot \boldsymbol{a} > \lambda a^*$.

Hence, for any $\lambda \in [\lambda^{\max}_{a^*}, \lambda^{\max})$ there exists an index $\bar{i} \in \{1, \ldots, K\}$ s.t. $a_{\bar{i}} > a^*$, $\lambda p^*_i(\lambda) = \alpha_i \mu_i$ for all $i < \bar{i}$, $\lambda p^*_i(\lambda) = 0$ for all $i > \bar{i}$ and $\lambda p^*_{\bar{i}}(\lambda) \in [0, \alpha_{\bar{i}} \mu_{\bar{i}})$. Since $\lambda p^*_i(\lambda) = \mu_i$ for all i s.t. $a_i < a^*$, $A'_{\text{LP-LB}}(\lambda^+)(i) \leq 0$ for all such i. Combining this with $A'_{\text{LP-LB}}(\lambda^+) \cdot \mathbf{1} = 1$ implies

$$A'_{\text{LP-LB}}(\lambda^+) \cdot \boldsymbol{a} > a^*. \tag{105}$$

□

B.7 Proof of Lemma 10

Proof. Recall Lemma 10

Lemma 10 (Unique sub-gradient). *For any $\lambda \in [0, \lambda^{\max})$, if a vector $\boldsymbol{p} = (p_1, \ldots, p_K)$ satisfy the conditions*

(1) $\boldsymbol{p} \cdot \mathbf{1} = 1$,
(2) $\boldsymbol{p} \cdot \boldsymbol{a} \geq a^*$,
(3) for all index $i \in \{1, \ldots, K\}$, if $A'_{LP\text{-}LB}(\lambda^+)(i) = 0$, then $p_i = 0$ and if $A'_{LP\text{-}LB}(\lambda^+)(i) \neq 0$, then $p_i A'_{LP\text{-}LB}(\lambda^+)(i) \geq 0$,
(4) $\boldsymbol{p} \cdot \boldsymbol{\mu}^{-1} \leq A'_{LP\text{-}LB}(\lambda^+) \cdot \boldsymbol{\mu}^{-1}$,

then $\boldsymbol{p} = (p_1, \ldots, p_K) = A'_{LP\text{-}LB}(\lambda^+)$

Assume that there exist a $\bar{\lambda} \in [0, \lambda^{\max})$ and a vector $\boldsymbol{p} = (p_1, \ldots, p_K)$ that satisfy the conditions given in Lemma 10. The proof proceeds as follows. We observe the solution of the LP $N_{\text{LP-LB}}(\lambda)$ in some right neighbourhood and prove that if $\boldsymbol{p} \neq A'_{\text{LP-LB}}(\lambda^+)$, then the LP has multiple optima.

Define,

$$\lambda_{\text{next-barrier}} = \sup \Bigg\{ \lambda' : \lambda' \geq 0, \mathbf{0} \leq \bar{\lambda}\boldsymbol{p}^*(\bar{\lambda}) + \lambda'' A'_{\text{LP-LB}}(\bar{\lambda}^+) \leq \boldsymbol{\mu}_{\boldsymbol{\alpha}}, \\ \mathbf{0} \leq \bar{\lambda}\boldsymbol{p}^*(\bar{\lambda}) + \lambda''\boldsymbol{p} \leq \boldsymbol{\mu}_{\boldsymbol{\alpha}} \text{ for all } \lambda'' \in [0, \lambda'] \Bigg\} \tag{106}$$

We later show that $\lambda_{\text{next-barrier}}$ is well-defined and strictly positive. Now, for any $\lambda \in [\bar{\lambda}, \bar{\lambda} + \lambda_{\text{next-barrier}})$, consider the routing probability $\boldsymbol{p}_1^* = \frac{1}{\lambda}\left(\bar{\lambda}\boldsymbol{p}^*(\bar{\lambda}) + (\lambda - \bar{\lambda})\boldsymbol{p}\right)$. First, we show that $\boldsymbol{p}_1^*$ is a feasible solution in the interval $(\bar{\lambda}, \bar{\lambda} + \lambda_{\text{next-barrier}})$. Clearly, $\mathbf{0} \leq \lambda\boldsymbol{p}_1^* = \bar{\lambda}\boldsymbol{p}^*(\bar{\lambda}) + (\lambda - \bar{\lambda})\boldsymbol{p} \leq \boldsymbol{\mu}_{\boldsymbol{\alpha}}$, using the definition of $\lambda_{\text{next-barrier}}$. Furthermore,

$$\begin{aligned} \boldsymbol{p}_1^* \cdot \mathbf{1} &= \frac{1}{\lambda}\left(\bar{\lambda}\boldsymbol{p}^*(\bar{\lambda}) + (\lambda - \bar{\lambda})\boldsymbol{p}\right) \cdot \mathbf{1} && (107)\\ &= \frac{1}{\lambda}\left(\bar{\lambda} + (\lambda - \bar{\lambda})\right) && (108)\\ &= 1, && (109) \end{aligned}$$

$$\begin{aligned} \boldsymbol{p}_1^* \cdot \boldsymbol{a} &= \frac{1}{\lambda}\left(\bar{\lambda}\boldsymbol{p}^*(\bar{\lambda}) + (\lambda - \bar{\lambda})\boldsymbol{p}\right) \cdot \boldsymbol{a} && (110)\\ &\geq \frac{1}{\lambda}\left(\bar{\lambda}a^* + (\lambda - \bar{\lambda})a^*\right) && (111)\\ &= a^*, && (112) \end{aligned}$$

and

$$\begin{aligned} \lambda\boldsymbol{p}_1^* \cdot \boldsymbol{\mu}^{-1} &= \left(\bar{\lambda}\boldsymbol{p}^*(\bar{\lambda}) + (\lambda - \bar{\lambda})\boldsymbol{p}\right) \cdot \boldsymbol{\mu}^{-1} && (113)\\ &= \bar{\lambda}\boldsymbol{p}^*(\bar{\lambda}) \cdot \boldsymbol{\mu}^{-1} + (\lambda - \bar{\lambda})\boldsymbol{p} \cdot \boldsymbol{\mu}^{-1} && (114)\\ &= N_{\text{LP-LB}}(\bar{\lambda}) + (\lambda - \bar{\lambda})\boldsymbol{p} \cdot \boldsymbol{\mu}^{-1} && (115)\\ &\leq N_{\text{LP-LB}}(\bar{\lambda}) + (\lambda - \bar{\lambda})A'_{\text{LP-LB}}(\bar{\lambda}^+) \cdot \boldsymbol{\mu}^{-1} && (116)\\ &= N_{\text{LP-LB}}(\bar{\lambda}) + (\lambda - \bar{\lambda})N'_{\text{LP-LB}}(\bar{\lambda}^+) && (117)\\ &\leq N_{\text{LP-LB}}(\lambda), && (118) \end{aligned}$$

where the last step uses convexity of LP proved in Lemma 4. Since, $N_{\text{LP-LB}}(\lambda)$ is the optimal solution, the equality must hold, implying that $\boldsymbol{p}_1^*$ is an optimal solution in the interval $(\bar{\lambda}, \bar{\lambda} + \lambda_{\text{next-barrier}})$. Using similar arguments, one can show that $\boldsymbol{p}_2^* = \frac{1}{\lambda}\left(\bar{\lambda}\boldsymbol{p}^*(\bar{\lambda}) + (\lambda - \bar{\lambda})A'_{\text{LP-LB}}(\bar{\lambda}^+)\right)$ is also an optimal solution in the interval $(\bar{\lambda}, \bar{\lambda} + \lambda_{\text{next-barrier}})$. Further, $\boldsymbol{p} \neq A'_{\text{LP-LB}}(\bar{\lambda}^+)$ implies $\boldsymbol{p}_1^* \neq \boldsymbol{p}_2^*$ implying there exist multiple optimal solutions, which contradicts Assumption 1, thus completing the proof.

Hence, the only remaining step is to show that $\lambda_{\text{next-barrier}}$ is well-defined and strictly positive. Observe that the condition $\bar{\lambda}\boldsymbol{p}^*(\bar{\lambda}) + \lambda' A'_{\text{LP-LB}}(\bar{\lambda}^+) \leq \boldsymbol{\mu_\alpha}$ implies $\bar{\lambda}\boldsymbol{p}^*(\bar{\lambda}) \cdot \mathbf{1} + \lambda' A'_{\text{LP-LB}}(\bar{\lambda}^+) \cdot \mathbf{1} = \bar{\lambda} + \lambda' \leq \sum_{i=1}^K \mu_i$. Hence, the supremum is bounded and therefore well-defined.

Next, we show that $\lambda_{\text{next-barrier}}$ is strictly positive. Define

$$\lambda_{\text{min-next-barrier}} = \frac{\min_{i:A'_{\text{LP-LB}}(\bar{\lambda}^+)(i)\neq 0}\left\{\bar{\lambda}p_i^*(\bar{\lambda})\mathbf{1}\left\{A'_{\text{LP-LB}}(\bar{\lambda}^+)(i) < 0\right\} + \left(\alpha_i\mu_i - \bar{\lambda}p_i^*(\bar{\lambda})\right)\mathbf{1}\left\{A'_{\text{LP-LB}}(\bar{\lambda}^+)(i) > 0\right\}\right\}}{\max\left\{\max_{i:p'_i\neq 0}\left\{|p'_i|\right\}, \max_{i:A'_{\text{LP-LB}}(\bar{\lambda}^+)(i)\neq 0}\left\{|A'_{\text{LP-LB}}(\bar{\lambda}^+)(i)|\right\}\right\}}. \tag{119}$$

For any index i, if the sign of the right derivative at the index i is negative, i.e., $A'_{\text{LP-LB}}(\bar{\lambda}^+)(i) < 0$, then the definition of right derivative implies $\bar{\lambda}p_i^*(\bar{\lambda})$ has to be a positive quantity. Similarly, if $A'_{\text{LP-LB}}(\bar{\lambda}^+)(i) > 0$, then the $\alpha_i\mu_i - \bar{\lambda}p_i^*(\bar{\lambda}) > 0$. Hence, $\lambda_{\text{min-next-barrier}}$ is a strictly positive quantity.

Now, for any index i s.t. $p_i > 0$ and for any $\lambda'' \in [0, \lambda_{\text{min-next-barrier}}]$, we have $A'_{\text{LP-LB}}(\bar{\lambda}^+)(i) > 0$ and

$$0 \leq \bar{\lambda}p_i^*(\bar{\lambda}) \leq \bar{\lambda}p_i^*(\bar{\lambda}) + \lambda'' p_i \leq \bar{\lambda}p_i^*(\bar{\lambda}) + \lambda_{\text{min-next-barrier}}p_i \tag{120}$$

$$\leq \bar{\lambda}p_i^*(\bar{\lambda}) + \frac{\mu_i - \bar{\lambda}p_i^*(\bar{\lambda})}{p_i}p_i \tag{121}$$

$$= \mu_i \tag{122}$$

For any index i s.t. $p_i < 0$ and for any $\lambda'' \in [0, \lambda_{\text{min-next-barrier}}]$, we have $A'_{\text{LP-LB}}(\bar{\lambda}^+)(i) < 0$ and

$$\alpha_i\mu_i \geq \bar{\lambda}p_i^*(\bar{\lambda}) \geq \bar{\lambda}p_i^*(\bar{\lambda}) + \lambda'' p_i \geq \bar{\lambda}p_i^*(\bar{\lambda}) + \lambda_{\text{next-barrier}}p_i \tag{123}$$

$$\geq \bar{\lambda}p_i^*(\bar{\lambda}) + \frac{\bar{\lambda}p_i^*(\bar{\lambda})}{|p_i|}p_i \tag{124}$$

$$= 0 \tag{125}$$

Hence, for all $\lambda'' \in [0, \lambda_{\text{min-next-barrier}}]$

$$\mathbf{0} \leq \bar{\lambda}\boldsymbol{p}^*(\bar{\lambda}) + \lambda''\boldsymbol{p} \leq \boldsymbol{\mu_\alpha} \tag{126}$$

Using similar arguments, we can show that

$$\mathbf{0} \leq \bar{\lambda}\boldsymbol{p}^*(\bar{\lambda}) + \lambda'' A'_{\text{LP-LB}}(\bar{\lambda}^+) \leq \boldsymbol{\mu_\alpha} \tag{127}$$

Hence, $\lambda_{\text{next-barrier}} \geq \lambda_{\text{min-next-barrier}} > 0$, which completes the proof. □