Managing Heterogeneous Multi-Class Queues in Real Time: A Comparison of Queueing Rules, Rolling-Horizon Optimization, and Reinforcement Learning

L. Trassoudaine[1,2] , M-H. Ibrahim[1], O. Ben-Ammar[1], S. Harispe[1]

***Abstract*— Managing a pool of heterogeneous multi-class server queues in real time is challenging, especially under heavy traffic and operational constraints. We study dynamic assignment policies and aim to minimize waiting times while maximizing customer throughput under real-time constraints. We compare several policies, including queueing-theory policies SJF, FCFS, and cμ-rule, a myopic rolling-horizon LP scheduling approach, and a PPO-based reinforcement learning approach. Simulation results show that SJF achieves the lowest waiting times, while LP maximizes throughput in Heavy Traffic, and PPO is competitive but does not dominate classical dispatching rules. All methods meet the sub-second decision requirement. These results highlight the strong performance of dynamic queueing theory-based optimization methods and the competitiveness of rolling horizon-based scheduling approaches for real-time queue management.**

I. INTRODUCTION

Queue management has been extensively studied in the literature and plays a critical role in many application domains, such as healthcare systems, where reducing waiting times and ensuring timely treatment of patients can be of vital importance. In this study, we investigate the design of control policies for assigning customers to servers within a heterogeneous server pool. Customers arriving at the system are associated with a specific and unique service requirement, modeled as a class drawn from a known finite set. Each customer class can only be processed by a subset of compatible servers, with compatibility relations known a priori. Server heterogeneity arises from two sources: first, service requirements differ across job types, leading to type-dependent average processing times; second, servers may differ in their processing characteristics, resulting in server–job-type–dependent service rates. Furthermore, customers waiting for service may abandon the system before being served, and servers may become temporarily unavailable due to unplanned interruptions.

The objective is to develop an allocation policy (i.e., queue discipline) that simultaneously minimizes customer waiting times and reduces the number of customers leaving the system without receiving service. The proposed study aims at comparing different approaches for defining control policies using (i) traditional Operations Research (OR), and in particular, queueing theory, as well as (ii) machine learning-based modeling relying on Reinforcement Learning (RL) and, in particular, Deep RL (DRL).

The key insights of our study are:

1) The classic Shortest Job First (SJF) and cμ-rule queueing policies perform very well in all configurations.
2) The myopic rolling horizon Linear Programming (LP) approach performs very well in heavy traffic in terms of customer throughput. Although its execution time is significantly longer than other policies, it remains acceptable.
3) While the studied RL approach underperforms compared to the queueing-theoretic policies, the performance gap remains moderate, and the method achieves satisfactory results overall.

The paper is organized as follows. Section (II) reviews the literature on commonly used approaches for solving queueing problems, drawing from both OR (e.g., queueing theory, combinatorial optimization) and RL techniques. Section (III) formalizes the problem under study. Section (IV) introduces the various models compared in our experiments, including the newly proposed approaches. Section (V) describes the evaluation protocol, and Section (VI) presents and discusses the results before concluding the paper.

II. RELATED WORK

The optimization of resource allocation in dynamic contexts has been extensively studied in scientific literature for over 70 years. A significant portion of the OR literature on this subject comes from the field of queueing theory. Several methodological developments have focused on simple policies that provide good theoretical performance guarantees for specific network structures and particular objective functions. For example, the optimality of Shortest Job First–based policies has been discussed since 1956 [1], [2]. More complex policies have been studied as MaxWeight [3], [4], cμ-rule [5], and its extensions to generalized costs [6] and customer abandonment [7], or Maximum Pressure [8]. They all offer their own optimality results, within specific conditions and objectives, and most of the time in heavy traffic conditions. Heavy traffic refers to a situation where a queueing system is operating near its maximum capacity, meaning the demand for service is close to (or temporarily exceeds) the system's ability to serve customers. Formally, heavy traffic occurs when the traffic intensity $\rho = \frac{\lambda}{\mu}$ approaches 1, with λ and μ the arrival and service rates respectively [9].

[1] SyCoIA, IMT Mines Ales, Ales, France
[2] ESII, Laverune, France
lois.trassoudaine@mines-ales.fr

These policies often achieve near-optimal performance under their modeling assumptions. However, their practical applicability may be limited in environments characterized by non-stationarity, complex operational constraints, or rapidly evolving system dynamics.

In response to these limitations, a variety of approaches based on Artificial Intelligence (AI), and more specifically on machine learning, have been explored in recent years. The goal in this case is to learn policies in simulated contexts or from data, rather than defining them ad hoc. The approaches studied very often adopt the RL paradigm adapted to sequential decision problems. In this case, they are mostly based on advanced modeling techniques based on Markov Decision Processes (MDP) [10] and DRL [11], [12]. The state of the art highlights that they also offer very encouraging performance in certain contexts that are difficult to address with standard approaches [13]—at the cost of losing significant attractive properties, e.g., convergence guarantees.

We can also refer to the literature on scheduling, especially on dynamic and rolling-horizon scheduling [14]. Early studies of dynamic scheduling in queueing networks considered fluid approximations of stochastic systems [15]. In this approach, the evolution of key quantities, such as inventory levels or queue lengths, is approximated using ordinary differential equations. The fluid network is then discretized into a grid, and a sequence of linear programs is solved at regular time intervals to determine dynamic allocation policies. The problem can also be formulated as a dynamic flexible workshop scheduling problem [16], [17]. A recent systematic review [18] identifies three types of strategies for addressing this kind of problem: predictive-reactive scheduling, completely reactive scheduling, and proactive scheduling. The authors highlight that many methods have been proposed, and that hybrid solutions have emerged as particularly promising and effective.

A concept closely related to these strategies is Model Predictive Control, which involves optimizing system behavior over a predicted time horizon at discrete intervals and applying only the first step as the control decision. This approach can also be applied to queueing systems [19].

To the best of our knowledge, comparative analyses of these approaches under varying queue loads and operational constraints remain limited. In this work, we benchmark traditional queueing policies, a myopic rolling horizon scheduling approach, and a Proximal Policy Optimization (PPO)-based RL method [20] within a unified framework.

III. PROBLEM DESCRIPTION

The goal is to assign customers to servers in real time to minimize waiting times while maximizing the total number of customers served over a day. The operation of the system is schematically shown in Fig. 1, and the main notations used in the paper are summarized in Table I.

The system is subject to a stationary customer arrival process, with servers that may pause in a stochastic, non-stationary, and unplanned manner. The environment includes multiple customer classes and multi-skilled, heterogeneous servers, differing in service rates. Furthermore, servers can serve only one customer at a time, and each customer can be served by only one server at a time. We also consider a non-preemptive setting. The entire system is modeled and evaluated using a continuous event-driven simulation.

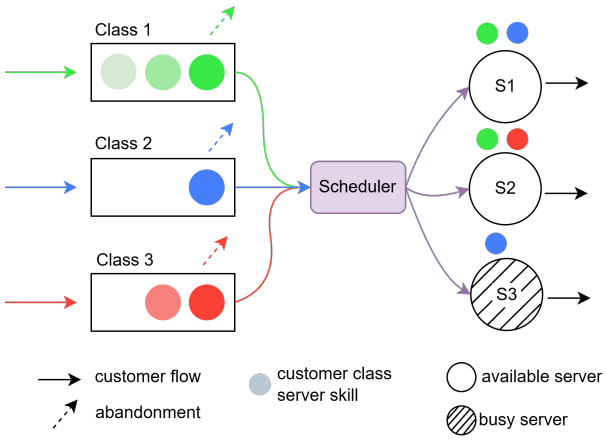

Fig. 1. Multi-class queue with heterogeneous servers.

TABLE I
NOTATIONS PART-I.

Global parameters	
$\mathscr{S}$	Set of servers, $\mathscr{S} = \{1,\ldots,c\}$
$\mathscr{L}$	Set of classes, $\mathscr{L} = \{1,\ldots,L\}$
$S_{j,l}$	Mean service time of server j for class l
	$S_{j,l} = 1/\mu_{j,l}$, $\forall j \in \mathscr{S}, l \in \mathscr{L}$
T	Maximum simulation time
T'	Maximum customer arrival time
W'_l	Target waiting time for customer class l, $\forall l \in \mathscr{L}$
p_{eligible}	Probability that a server can serve a given class
Rolling Window parameters for time step s	
t	Current simulation time
s	Current number of steps
$\mathscr{N}$	Set of customers in the queue at the observed instant, $\mathscr{N} = \{1,\ldots,N\}$
q_l	Set of customers of class l, $\forall l \in \mathscr{L}$
r_j	Release time of server j, $\forall j \in \mathscr{S}$
l_i	Class of customer i, $\forall l_i \in \mathscr{L}, i \in \mathscr{N}$
a_i	Arrival time of customer i, $\forall i \in \mathscr{N}$

A. Stochastic parametrization

1) Arrival law: In real-world settings, customer arrivals typically exhibit non-stationary behavior. For example, this may be due to a public transportation stop in front of the store, which can create an influx of customers. Although these arrivals are non-stationary in reality, we simulate them using stationary arrival laws that can still represent bursts. To do this, we use a batch Poisson. Customer batches arrive according to a Poisson process with rate λ. The batch size follows a geometric distribution. Each customer in the batch is assigned randomly to a class according to class-specific probabilities.

2) Stochastic abandonment: If customers wait too long, they may leave without being served. This has been modeled

using an abandonment time following an exponential distribution with mean W_l'. Abandonment is not modeled for all configurations defined in (V), but only for the setup Heavy Traffic with Abandonment (HTA).

3) Stochastic service times: Each customer class is associated with a specific average service time. To enable generalization of algorithms, particularly learning algorithms, the class-specific averages are drawn uniformly over the minimum and maximum service times.

Given that servers have varying skills, a binomial draw with probability p_{eligible} determines whether a server is eligible to serve a given class. Furthermore, servers exhibit different skills across classes: the average service time for a class is therefore perturbed per server according to a normal distribution centered on the class mean.

Finally, the actual service times are drawn from a truncated normal distribution with these server averages as the mean and randomized standard deviations, reflecting stochastic variability in service times.

4) Server idleness: To simulate realistic server behavior, including unpredictability, we introduce non-stationary, unexpected periods of server idleness. Each server has a 30% probability of opening late, closing early, or taking an unscheduled break.

For idleness occurring during the simulation, the start time is drawn from an exponential distribution with mean $T/2$. The duration of unavailability is drawn from a normal distribution, capturing variability in break length.

IV. OPTIMIZATION APPROACHES

Three types of optimization approaches have been developed. The first type serves as a baseline and consists of classic queueing-theory policies (IV-A). The second type is based on linear programming within a rolling horizon framework (IV-B). Finally, an RL policy has been developed (IV-C). These approaches are based on a continuous event simulation in which an assignment is made when an event allowing an assignment occurs. These events are:

- Release of a server and presence of a compatible customer in the queue.
- Arrival of a customer and availability of a compatible server.

This means that when an assignment is requested, a valid assignment must exist.

A. Baseline approaches

The baseline models are based on selecting the least-busy compatible and available server. Customers are organized into queues by class. Policies will then choose a queue. The customer assigned by the policy is the customer who has waited the longest in the queue. This assignment choice will be the same as the action studied in the MDP modeling (IV-C). 4 baselines have been studied:

- Random - uniform random selection.
- FCFS - First Come First Served, and fastest class upon equality.
- SJF - Shortest Job First.
- cμ-rule on the queue length, with linear costs.

B. Rolling horizon and linear programming formulation

We develop a myopic rolling horizon scheduling approach. At each event allowing an assignment, a linear program is solved over the entire horizon using the information available at that moment. The linear program contains all information on waiting customers and servers (availability, class compatibility, and service times). Note that once the linear program is solved, only the first assignment for each available server is kept, making the server unavailable for other assignments until completion.

1) Linear Programming formulation: The variables in Table II are introduced and optimized. The objective function and constraints are given below:

$$\min \sum_{j\in\mathscr{S}}\sum_{k\in\mathscr{N}} W_{jk} \tag{1}$$

subject to constraints:

$$\sum_{j\in\mathscr{S}}\sum_{k\in\mathscr{N}} x_{ijk} = 1 \qquad \forall i\in\mathscr{N} \tag{2}$$

$$\sum_{i\in\mathscr{N}} x_{ijk} \le 1 \qquad \forall j\in\mathscr{S},\ k\in\mathscr{N} \tag{3}$$

$$x_{ijk} \le S_{jl_i} \qquad \forall j\in\mathscr{S}, i,k\in\mathscr{N} \tag{4}$$

$$t_{jk} \ge t_{jk-1} + \sum_{i\in\mathscr{N}} S_{jl_i} x_{ijk-1} \qquad \forall j\in\mathscr{S}, k\in\mathscr{N}_{\setminus\{1\}} \tag{5}$$

$$r_j \le t_{j1} \qquad \forall j\in\mathscr{S} \tag{6}$$

$$t_{jk} \ge \sum_{i\in\mathscr{N}} a_i x_{ijk} \qquad \forall j\in\mathscr{S}, k\in\mathscr{N} \tag{7}$$

$$W_{jk} = t_{jk} + \sum_{i\in\mathscr{N}} (S_{jl_i} - a_i) x_{ijk} \qquad \forall j\in\mathscr{S}, k\in\mathscr{N} \tag{8}$$

$$t_{jk}, W_{jk} \ge 0 \qquad \forall j\in\mathscr{S}, k\in\mathscr{N} \tag{9}$$

$$x_{ijk} \in [0,1] \qquad \forall j\in\mathscr{S}, i,k\in\mathscr{N} \tag{10}$$

The objective (1) is to minimize the cumulative time each server position spends in the system, including waiting time and service time. Initially, we considered an objective based solely on waiting times. This worked fairly well under low load, as service times had little influence on congestion, but proved suboptimal under heavy traffic and overload conditions. Under these conditions, ignoring service times can lead to assignments that reduce immediate waiting while unintentionally increasing overall congestion due to inefficient server utilization. Integrating service time into the objective yields better system-wide performance, both in terms of cumulative waiting time and throughput, especially as load increases.

TABLE II
NOTATIONS PART-II.

Decision variables	
$x_{ijk} \in [0,1]$	equals 1 if customer i is assigned to position k on server j, 0 otherwise
Intermediate variables	
$t_{jk} \ge 0$	service start time at position k on server j
$W_{jk} \ge 0$	expected sojourn time of the customer assigned to position k on server j

Equations (2) ensure that a customer is only served by one server once. Constraints (3) ensure that a server serves only one customer at once. Constraints (4) ensure that a server only serves eligible classes. In a non-preemptive system, a service can only start after the previously scheduled service has finished (5). Since servers can be on break and we are not questioning previous server unavailability, the first service assigned to a server must take place after the server has been released (6). Similarly, customer service can only begin after the customer arrives (7). Constraints (8) calculate the time spent in the system on all server positions. This quantity is not equal to the time spent in the system by all customers and does not reflect actual assignments. Finally, expressions (9)-(10) give the domain of the variables.

We note that, under specific conditions, the LP yields integer-optimal solutions. In the following proposition, we prove that the LP relaxation yields integer optimal solutions and is equivalent to the mixed-integer linear programming (MILP) formulation.

Proposition IV.1. *For all $j \in \mathscr{S}$ and $i,k \in \mathscr{N}$, if $a_i \leq 0$, then the linear programming relaxation (1)–(10) is equivalent to its MILP formulation and admits an optimal solution satisfying $x_{ijk} \in \{0,1\}$.*

Proof: In the previous linear program, the decision variable x_{ijk} was relaxed. This relaxation is integral when the a_i are negative. Otherwise, the relaxation is fractional. We will prove relaxation integrality in the case of $a_i \leq 0$.

On one hand, since each t_{jk} appears with coefficient +1 in the objective function, the LP pushes each t_{jk} to its minimum feasible value. So (5) and (6) are active and can be formulated respectively as follows:

$$t_{jk} = t_{jk-1} + \sum_{i \in \mathscr{N}} S_{jl_i} x_{ijk-1} \qquad \forall j \in \mathscr{S}, k \in \mathscr{N}_{\setminus\{1\}} \tag{11}$$

$$t_{j1} = r_j \qquad \forall j \in \mathscr{S} \tag{12}$$

Then, we have a new formulation for t_{jk}:

$$t_{jk} = r_j + \sum_{m=1}^{k-1} \sum_{i \in \mathscr{N}} S_{jl_i} x_{ijm} \qquad \forall j \in \mathscr{S}, k \in \mathscr{N}_{\setminus\{1\}} \tag{13}$$

On the other hand, in our rolling-horizon case, we always consider customer arrivals before the current simulation time, set to zero, thus allowing negative a_i. In other words, if $a_i \leq 0$ and $x_{ijk} \geq 0$ $\forall i,k \in \mathscr{N}$, $\forall j \in \mathscr{S}$, we have $\sum_{i \in \mathscr{N}} a_i x_{ijk} \leq 0$. So (7) is redundant because always satisfied ($t_{jk} \geq 0$).

Finally, we can substitute t_{jk} in the objective function, and (1) becomes (14), which is linear for x_{ijk}:

$$\min \sum_{j \in \mathscr{S}} \sum_{k \in \mathscr{N}} \sum_{i \in \mathscr{N}} \left(S_{jl_i}(N-k+1) - a_i \right) \times x_{ijk} \tag{14}$$

If we remove the incompatible columns induced by (4), the objective function is only subject to (2) and (3). This is a bipartite assignment problem between: customer nodes i, and server-position nodes (j,k).

So this is a totally unimodular matrix. By the Hoffman–Kruskal theorem [21], every extreme point of the feasible polyhedron is integral. So the linear relaxation admits an optimal solution with $x_{ijk} \in \{0,1\}$. □

C. Reinforcement Learning

We model the system as a continuous event-based MDP. As for the baseline policies, the action is to choose the oldest customer in each queue to be assigned to the server that has done the least work. The RL policy can also choose to hold, which means no assigning a customer to that server. This is useful, for example, when one customer is in the queue and two servers are available. If the next server can serve the customer more quickly, the policy can choose to hold in order to reserve the assignment to the second server.

The observation extracted from the state is conditioned to chosen server $j^\star$ (15). The observation gives a representation of the queues, with the waiting time of the head of the queues and the queues length. There is also the service times of each available servers and the id of the current server to identify its service times in the matrix.

$$\begin{aligned} O_t = \Big(& \{\max_{i \in q_l}(t - a_i)\}_{l \in \mathscr{L}}, \{|q_l|\}_{l \in \mathscr{L}}, j^\star, \\ & \{S_{jl}\}_{j \in \mathscr{S}, l \in \mathscr{L}}, \text{ if } j \text{ available, 0 otherwise.} \Big) \end{aligned} \tag{15}$$

The reward definition (16) is conditioned to the chosen customer $i^\star$. The first reward component (17) penalizes the normalized waiting time. When the observed waiting time exceeds the target waiting time, the reward is set to zero, because the target waiting time coincides with the average abandonment time which should not be exceeded. The second reward component (18) rewards the service times closest to the smallest service time that could be obtained for this customer with all servers. This encourages the RL policy not to assign a customer if better service is possible on another server. The total reward (16) combines the two components described above. Although defined locally for each selected customer i^*, the reward serves global objectives: penalizing waiting times reduces cumulative waiting times, while faster service assignments reduce congestion and abandonment, thereby improving the overall throughput of the system.

$$R = R_w + R_s \tag{16}$$

$$R_w = \begin{cases} \dfrac{W'_{l_{i^\star}} - (t - a_{i^\star})}{W'_{c_{i^\star}}}, & \text{if } W'_{c_{i^\star}} > t - a_{i^\star} \\ 0, & \text{otherwise.} \end{cases} \tag{17}$$

$$R_s = \begin{cases} \dfrac{\max_{j \in \mathscr{S}}(S_{jl_{i^\star}}) - S_{j^\star l_{i^\star}}}{\max_{j \in \mathscr{S}}(S_{jl_{i^\star}}) - \min_{j \in \mathscr{S}}(S_{jl_{i^\star}})} \\ \quad \text{if } \max_{j \in \mathscr{S}} S_{jl_{i^\star}} \neq \min_{j \in \mathscr{S}} S_{jl_{i^\star}}, \\ 0, \quad \text{otherwise.} \end{cases} \tag{18}$$

We choose to use a Maskable PPO agent for its stability and proven performances [20], [22]. It is designed for environments where some actions are invalid in certain states. Instead of letting the agent choose from all actions and penalizing invalid ones afterward, it masks them out so the policy cannot select them. Training was limited to 10^6 steps

(∼17,000 instances) using the hyperparameters summarized in Table III, selected via random search.

V. EXPERIMENTAL SETUP

The approaches were implemented in Python 3.10. They were executed on a single core from a single node of a cluster. The node used is a Intel(R) Core(TM) Ultra 5 135H with 14 cores at 1.70 GHz and 16 GB RAM. The LP approach was solved using Gurobi. The simulation framework ensures full reproducibility. The approaches were tested on 500 identical instances for four different configurations. The RL approach has been trained specifically for the setups, but it did not see the tested instances during training. The purpose of the four evaluation settings is to test the approaches under different loads and constraints in order to identify their strengths and weaknesses. We consider a simplified ESII case study [23]. There are $c = 5$ servers and $L = 10$ classes. The average service time is $\mu = 1/25$. Configurations are defined as described in Table IV. They differ in arrival rate, which determines traffic intensity, and in server eligibility probability, which controls the chances that a server can serve a class. In addition, HTA introduces customer abandonment. Our framework simulates a 5.5-hour day in which customers can arrive within the first 5 hours. The following 30 minutes serve as buffers for finishing serving all customers.

VI. RESULTS

To compare the results, we focus on three key metrics: waiting time (WT) in Fig. 2, number of unserved customers (UC) in Fig. 3, and execution time (ET, in ms) in Table V. Our goals are to minimize WT for all customers (including unserved ones), maximize the number of customers served, and perform assignments in real time (less than 1 second).

Under Low Load (LL), FCFS, SJF, cμ-rule and PPO achieve similar WT, FCFS being slightly better, closely followed by PPO. The LP solution fails to optimize waiting times efficiently. Very few customers remain unserved, indicating that the system is stable and all policies handle this regime effectively.

Under Heavy Traffic (HT), the system is saturated. Poor policies can destabilize queues, leading to high WT and UC. SJF minimizes WT, while LP achieves better overall throughput. While the median UC under LP is higher than under SJF, the third quartile is slightly lower, indicating fewer extreme UC values. The cμ-rule provides a balanced alternative, achieving low WT and UC simultaneously.

When adding server ineligibility constraints (HTI), the relative ranking of policies remains similar: SJF continues to outperform others in WT and UC, with LP achieving very similar results, especially for UC.

With customer abandonment (HTA), queues are partially stabilized, slightly increasing UC compared to HT. As shown in Fig. 2, WT decreases compared to HT, which shows that the system is stabilizing. cμ-rule consistently performs best on WT and UC, with SJF close behind. PPO and FCFS achieve near-optimal WT, providing practical alternatives.

Regarding execution time, all policies meet the 1-second target. LP is the slowest, particularly under HTI, but still within limits. Even under extreme overload ($\rho = 2.5$), LP's average WT is around 5.5 ± 6.6 ms and its maximum execution time is 90 ms. While all policies satisfy the real-time requirement, LP is the least efficient computationally.

TABLE III

MASKABLE PPO HYPERPARAMETERS.

Hyperparameter	Value
Rollout length	512
Epochs per update	10
Batch size	256
Architecture	[128, 64]
Learning rate	3.8×10^{-5}
Clip range	0.26
Discount factor γ	0.97
Entropy coefficient	10^{-4}

TABLE IV

EXPERIMENTAL SETUP.

Instance settings	Eligibility $p_{eligible}$	Arrivals λ	Traffic intensity $\rho = \lambda/(\mu \times c)$
Low Load (LL)	0.7	1/7	0.7
Heavy Traffic (HT)	0.7	1/5	1
HT, Ineligibility (HTI)	0.2	1/5	1
HT, Abandonment (HTA)	0.7	1/5	1

Note: The eligibility constraint indicates the average percentage of classes that a server can serve. λ is the mean arrival rate. Traffic intensity [9]: $\rho \ll 1$: low load, $\rho \simeq 1$: heavy traffic, $\rho > 1$: unstable traffic. Mean abandonment time for HTA is 60 minutes.

TABLE V

EXECUTION TIME (MS) BY CONFIGURATION AND POLICY.

Config.	FCFS, Random, SJF, cμ-rule	LP	PPO
LL	0.0 ± 0.0	0.9 ± 1.0	0.5 ± 0.1
HT	0.0 ± 0.0	2.6 ± 5.1	0.5 ± 0.1
HTI	0.0 ± 0.0	12.7 ± 19.5	0.5 ± 0.1
HTA	0.0 ± 0.0	1.1 ± 1.0	0.5 ± 0.1

VII. CONCLUSIONS AND FUTURE WORK

We studied dynamic resource allocation in the context of multi-class queues with a pool of heterogeneous servers. We compared OR policies and a RL policy under different configurations and traffic intensities with different constraints.

Under LL, all policies perform well, although the LP approach shows weaknesses. But under HT, serving the fastest customer gives the best performance in terms of waiting time, and the LP approach has the highest customer throughput. Under HTA, the system stabilizes again and SJF becomes the best policy. The RL approach is close in terms of metrics, but does not match the performance of the other policies. Due to the training limitation, full convergence of the RL approach cannot be guaranteed. All approaches manage to meet the execution time target of sub-second.

Several directions for future research remain. First, improved LP formulations and a shift from a myopic to a lookahead framework could further enhance computational efficiency and modeling accuracy by extending the optimization horizon at each step. Second, further work on

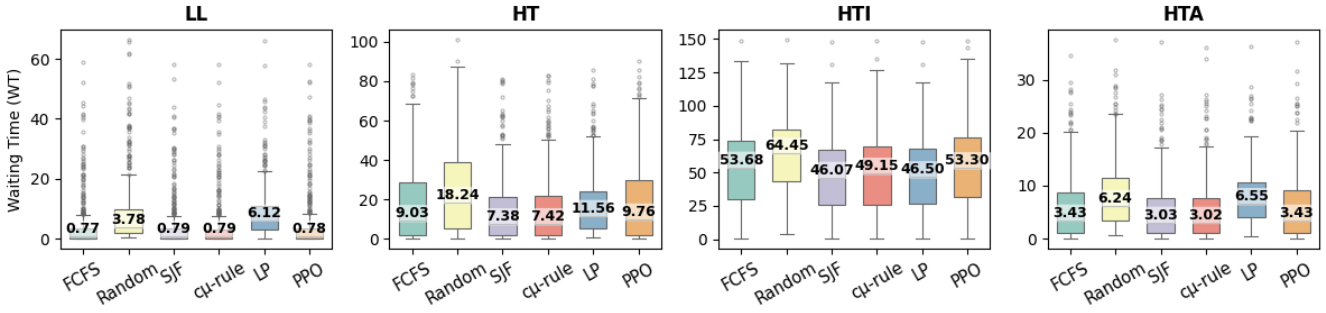

Fig. 2. Comparison of waiting times (WT) across various setups and policies.

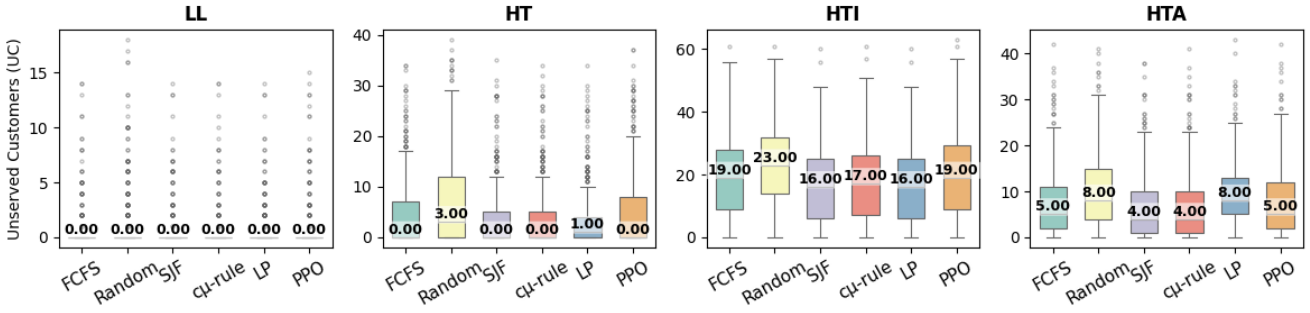

Fig. 3. Comparison of the number of unserved customers (UC) across various setups and policies.

the RL formulation and training strategy could improve its performance and convergence. Third, the current model captures only a subset of industrial constraints, and extending it to incorporate additional features such as customer appointments is a natural next step. Fourth, validating the approaches on real-world data and on more complex, non-stationary simulated data, is essential to assess its practical impact. Finally, combining scheduling methods and AI could improve performance. AI allows predicting customer arrivals and incorporating information on stochasticity into the scheduling approach to make more informed assignments.

References

[1] W. E. Smith, Various optimizers for single-stage production, Naval Research Logistics Quarterly 3 (1-2) (1956) 59–66.

[2] J. Dong, R. Ibrahim, Shortest-Job-First Scheduling in Many-Server Queues with Impatient Customers and Noisy Service-Time Estimates, Operations Research 73 (6) (2025) 3139–3155.

[3] L. Tassiulas, A. Ephremides, Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks, IEEE Transactions on Automatic Control 37 (12) (1992) 1936–1948.

[4] A. L. Stolyar, MaxWeight scheduling in a generalized switch: State space collapse and workload minimization in heavy traffic, The Annals of Applied Probability 14 (1) (2004) 1–53.

[5] C. Buyukkoc, P. Varaiya, J. Walrand, The $c\mu$ rule revisited, Advances in Applied Probability 17 (1) (1985) 237–238.

[6] J. A. van Mieghem, Dynamic Scheduling with Convex Delay Costs: The Generalized $c|mu$ Rule, The Annals of Applied Probability 5 (3) (1995) 809 – 833.

[7] R. Atar, C. Giat, N. Shimkin, The $c\mu/\theta$ rule for many-server queues with abandonment, Operations Research 58 (5) (2010) 1427–1439.

[8] J. G. Dai, W. Lin, Maximum Pressure Policies in Stochastic Processing Networks, Operations Research 53 (2) (2005) 197–218.

[9] J. F. Shortle, J. M. Thompson, D. Gross, C. M. Harris, Fundamentals of Queueing Theory, 1st Edition, Wiley Series in Probability and Statistics, Wiley, 2018.

[10] R. S. Sutton, A. Barto, Reinforcement Learning: An Introduction, second edition Edition, Adaptive Computation and Machine Learning, The MIT Press, Cambridge, Massachusetts London, England, 2020.

[11] H. Chen, A. Li, E. Che, J. Dong, T. Peng, H. Namkoong, Qgym: Scalable simulation and benchmarking of queuing network controllers, Advances in Neural Information Processing Systems 37 (2024) 92401–92419.

[12] B. Liu, Q. Xie, E. Modiano, Rl-qn: A reinforcement learning framework for optimal control of queueing systems 7 (1) (2022).

[13] J. Dai, M. Gluzman, Queueing Network Controls via Deep Reinforcement Learning, Stochastic Systems 12 (1) (2022) 30–67.

[14] Y. Zeiträg, J. R. Figueira, A novel machine-learning rolling horizon heuristic for dynamic lot-sizing and job shop scheduling problems, International Journal of Production Research 63 (12) (2025) 4563–4589.

[15] H. Chen, D. D. Yao, Dynamic Scheduling of a Multiclass Fluid Network, Operations Research (1993).

[16] S. K. Fuladi, C.-S. Kim, Dynamic events in the flexible job-shop scheduling problem: Rescheduling with a hybrid metaheuristic algorithm, Algorithms 17 (4) (2024).

[17] D. Johnson, G. Chen, Y. Lu, Multi-agent reinforcement learning for real-time dynamic production scheduling in a robot assembly cell, IEEE Robotics and Automation Letters 7 (3) (2022) 7684–7691.

[18] C. Destouet, H. Tlahig, B. Bettayeb, B. Mazari, Systematic review and future directions in dynamic flexible job shop scheduling: A decade of research (2025).

[19] J. S. H. van Leeuwaarden, E. Lefeber, Y. Nazarathy, J. E. Rooda, Model predictive control for the acquisition queue and related queueing networks, Association for Computing Machinery, New York, NY, USA, 2010.

[20] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms (2017).

[21] A. J. Hoffman, J. B. Kruskal, 13. Integral Boundary Points of Convex Polyhedra, Princeton University Press, Princeton, 2016, pp. 223–246.

[22] S. Huang, S. Ontañón, A Closer Look at Invalid Action Masking in Policy Gradient Algorithms, The International FLAIRS Conference Proceedings 35 (2022).

[23] ESII, Innovative solutions for customer journey and flow management to free your visitors from waiting. (2026).
URL https://www.esii.com/en/