TECHNISCHE UNIVERSITÄT DARMSTADT

SCALABLE PLANNING IN LARGE MULTI-AGENT QUEUING SYSTEMS

Dem Fachbereich Elektrotechnik und Informationstechnik der
TECHNISCHEN UNIVERSITÄT DARMSTADT

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Dissertation

von

ANAM TAHIR, M.SC.

Referent:	Prof. Dr. techn. Heinz Köppl
Korreferent I:	Prof. Dr.-Ing. Ralf Steinmetz
Korreferent II:	Prof. Dr.-Ing. Amr Rizk

Tag der Einreichung:	15. Januar 2024
Tag der mündlichen Prüfung:	2024-05-23

D17
Darmstadt 2024

Die Arbeit von Anam Tahir wurde von der Deutschen Forschungsgemeinschaft (DFG) innerhalb des Sonderforschungsbereiches (SFB) 1053 „MAKI — Multi-Mechanismen Adaption für das künftige Internet“ und der LOEWE Initiative (Hesse, Germany) innerhalb des „emergenCITY center — Die resiliente digitale Stadt“ gefördert.

Tahir, Anam: *Scalable Planning in Large Multi-Agent Queuing Systems*
Darmstadt, Technische Universität Darmstadt
Jahr der Veröffentlichung der Dissertation auf TUprints: 2024
Tag der mündlichen Prüfung: 2024-05-23

Dedicated to Saad Javed.

KURZFASSUNG

Diese Dissertation präsentiert Rahmenbedingungen zur Modellierungsmethoden großer Warteschlangensysteme, die integraler Bestandteil unseres täglichen Lebens sind und oftmals Verwendung in realen Prozessen und Systemen haben. Die Lastverteilung, ein wesentlicher Bestandteil von Warteschlangensystemen, umfasst die Verteilung eingehender Aufgaben auf verfügbare Ressourcen, um bestimmte Ziele wie die Minimierung von Wartezeiten oder das Vermeiden von Aufgabenverlusten zu erreichen. Trotz bestehender statischer Lastenausgleichsalgorithmen erfordert die wachsende Komplexität von Rechensystemen dynamische Ansätze, wie in dieser Arbeit argumentiert wird.

Große Warteschlangensysteme stehen vor Herausforderungen wie partieller Beobachtbarkeit und Netzwerkverzögerungen, was die Notwendigkeit skalierbarer Strategien betont, die im Fokus dieser Arbeit stehen.

Die Arbeit beginnt mit einem teilweise beobachtbaren Einzelagentensystem mit einer großen Anzahl von Warteschlangen, das als teilweise beobachtbarer Markov-Entscheidungsprozess modelliert und anschließend mithilfe des Monte Carlo Tree Search Algorithmus gelöst wird. Das Warteschlangensystem und der vorgeschlagene Lastenausgleich werden unter Verwendung verschiedener Verteilungen von Ankunfts- und Bedienzeiten sowie verschiedener Belohnungsfunktionen analysiert. Kaggle-Netzwerkverkehrsdaten wurden ebenfalls verwendet, um die zugrundeliegenden Verteilungen für Ankunfts- und Bedienprozesse für reale Systeme zu modellieren, und anschließend wurde die Leistung der vorgeschlagenen Strategien darauf analysiert.

Als Nächstes wurde dieses Einzelagentenmodell auf das Multi-Agenten Warteschlangensystem erweitert, wobei die Herausforderungen der Skalierbarkeit und Nichtstationarität durch die Modellierung dieses großen Warteschlangensystems als ein Mean-Field-Control-Problem mit beliebigen Synchronisationsverzögerungen bewältigt wurden. Die Lastausgleichsstrategie für den resultierenden Einzelagenten-Markov-Entscheidungsprozess wurde mithilfe des Proximal Policy Optimization Algorithmus aus dem Bereich des Reinforcement Learning erlernt. Dieser Beitrag betont die Notwendigkeit, eine Strategien für den Fall zu erlernen, wenn eine Synchronisationsverzögerungen weder niedrig sind (sodass Join-the-Shortest-Queue optimal ist) noch hoch (sodass die zufällige Zuweisung die beste Lastausgleichsstrategie ist). Der Beitrag bietet auch theoretische Garantien und weist empirisch nach, dass die im Mean-Field-System erlernten Strategien in großen endlichen Warteschlangensystemen gut funktionieren.

Der erfolgreiche Rahmen des Mean-Field-Control zur Modellierung eines großen Warteschlangensystems wurde dann weiterentwickelt, um die Lokalität der Interaktionen zwischen Agenten zu berücksichtigen, anstatt von einem vollständig verbunden Netzwerk auszugehen, in dem jeder Agent Zugriff auf jede andere Warteschlange haben kann. Um diese dezentralen Interaktionen zu modellieren, wurde die kürzlich entwickelte Sparse Mean-Field-Theorie verwendet und erweitert, um ein Sparse Mean-Field-Control-Rahmen

werk zu erhalten. Der Proximal Policy Optimization Algorithmus wurde dann auf den resultierenden Mean-Field-Control-Markov-Entscheidungsprozess angewendet, um eine skalierbare und dezentrale Lastausgleichsstrategie zu erlernen. In allen vorgenannten Beiträgen haben unsere vorgeschlagenen, gelernten Lastausgleichsstrategien eine gute Leistung im Vergleich zu bestehenden Arbeiten auf dem neuesten Stand der Technik erzielt, was zukünftige Arbeiten in diese Richtung motiviert.

ABSTRACT

This dissertation presents methods for modelling large queuing systems which are integral to our daily lives, impacting processes and systems significantly. Load balancing, is a vital component of queuing systems, which involves distributing incoming jobs among available resources to achieve specific objectives like minimizing waiting times or job drops. Despite existing static load balancing algorithms, the growing complexity of computational systems necessitates dynamic approaches, as argued in this thesis. Large queuing systems face challenges like partial observability and network delays, emphasizing the need for scalable policies, which is the primary focus of this thesis.

The thesis starts with a partially observable single-agent system with large number of queues which is modelled as a partially observable Markov decision process and then solved using Monte Carlo tree search algorithm. The queuing system and the proposed load balancing is analyzed using various inter-arrival and service time distributions as well different reward functions. Network traffic data from Kaggle was also used to infer the underlying distributions for the arrival and service processes for a real system, and then analyzed the performance of our proposed policy on it.

Next, this single-agent model was extended to the multi-agent queuing system, where the challenges of scalability and non-stationary were tackled by modelling this large queuing system as a mean-field control problem with arbitrary synchronization delays. The load balancing policy for the resulting single-agent Markov decision process was learned using the proximal policy optimization algorithm from reinforcement learning. This contribution highlights the need for learning a policy for when the synchronization delays is not too low, when join-the-shortest-queue is optimal, or not too high, when random allocation is the best load balancing policy. It also provides theoretical guarantees and empirically proves that the policy learned in the mean-field system performs well in large finite queuing systems as well.

The successful framework of mean-field control for modelling a large queuing system was then further extended to include the locality of interactions between agents, instead of assuming a fully connected network where every agent can have access to every other queue. To model these decentralized interactions, the recently developed sparse mean-field theory was used and extended to obtain a mean-field control framework. The proximal policy optimization algorithm was then used on the resulting sparse mean-field control Markov decision process to learn a scalable and decentralized load balancing policy. In all the above-mentioned contributions, our proposed learned load balancing policies were able to perform well when compared to the existing state-of-the-art work, thus motivating future works in this direction.

PUBLICATIONS

The following papers were published or are under review during the doctoral candidacy:

[1] A. Tahir, B. Alt, A. Rizk, and H. Koeppl, "Load balancing in compute clusters with delayed feedback", *IEEE Transactions on Computers*, 2022.

[2] A. Tahir, K. Cui, and H. Koeppl, "Learning mean-field control for delayed information load balancing in large queuing systems", in *Proceedings of the 51st International Conference on Parallel Processing*, 2022, pp. 1–11.

[3] A. Tahir, K. Cui and H. Koeppl, "Sparse mean field load balancing in large localized queueing systems", *arXiv preprint arXiv:2312.12973*, pp. 1–22, 2023.

[4] A. Tahir, K. Cui, B. Alt, A. Rizk, and H. Koeppl, "Collaborative Optimization of the Age of Information under Partial Observability", *IFIP Networking Conference*, pp. 1–7, 2024.

[5] A. Tahir, H. Al-Shatri, K. Kiekenap, and A. Klein, "Ultra-reliable low latency communication for consensus control in multi-agent systems", in *IEEE Wireless Communications and Networking Conference (WCNC)*, IEEE, 2019, pp. 1–7.

[6] S. Azem, A. Tahir, and H. Koeppl, "Dynamic time slot allocation algorithm for quadcopter swarms", in *IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, 2022, pp. 1–6.

[7] J. Jia, A. Tahir, and H. Koeppl, "Decentralized coordination in partially observable queueing networks", in *IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2022, pp. 1491–1496.

[8] K. Cui, A. Tahir, M. Sinzger, and H. Koeppl, "Discrete-time mean field control with environment states", in *60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 5239–5246.

[9] Y. Alkhalili, J. Weil, A. Tahir, T. Meuser, B. Koldehofe, A. Mauthe, H. Koeppl, R. Steinmetz, et al., "Towards QoE-driven optimization of multi-dimensional content streaming", in *Electronic Communications of the EASST*, vol. 80, 2021.

[10] J. Weil, Y. Alkhalili, A. Tahir, T. Gruczyk, T. Meuser, M. Mu, H. Koeppl, and A. Mauthe, "Modeling quality of experience for compressed point cloud sequences based on a subjective study", in *15th International Conference on Quality of Multimedia Experience (QoMEX)*, IEEE, 2023, pp. 135–140.

[11] K. Cui, M. B. Yilmaz, A. Tahir, A. Klein, and H. Koeppl, "Optimal offloading strategies for edge-computing via mean-field games and control", in *IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2022, pp. 976–981.

[12] K. Cui, A. Tahir, G. Ekinci, A. Elshamanhory, Y. Eich, M. Li, and H. Koeppl, "A survey on large-population systems and scalable multi-agent reinforcement learning", *arXiv preprint arXiv:2209.03859*, pp. 1–21, 2022.

[13] K. Cui, C. Fabian, A. Tahir, and H. Koeppl, "Major-Minor Mean Field Multi-Agent Reinforcement Learning", in *International Conference on Machine Learning (ICML)*, 2024, pp. 1-30.

ACKNOWLEDGMENTS

This dissertation has been a learning experience like no other, with good and bad times going hand in hand. I have had the pleasure of working with great people both at mind and at heart and have had the chance to make some special memories which will stay with me forever.

I would first and foremost like to thank my mentor and supervisor Prof. Heinz Koeppl without whose guidance and faith in me, I would not have been able to achieve the biggest milestone of my life.

Then I would like to thank my husband, Saad Javed, for always believing in me and being my cheerleader for the last five years and never letting me give up or slow down. A special thank you to my parents for all their sacrifices that enabled me to come this far and for their prayers that have brought me all my success in life. Also a big thank you to all my friends who were there for me.

I want to express my sincere gratitude to Prof. Amr Rizk for being my mentor and for all his guidance and support so far. I would like to thank the German Research Foundation (DFG) and the team of Collaborative Research Center (SFB) 1053 "MAKI — Multi-Mechanisms Adaptation for the Future Internet" and the LOEWE Initiative (Hesse, Germany) within the emergenCITY center for funding my work. Also, a special thanks to Prof. Ralf Steinmetz for inviting me to KOM and for always providing a collaborative working environment.

A special thank you to Bastian Alt and Kai Cui for not only all the work we did together, but also for always being so patient with me.

Lastly, I would like to thank the amazing people of SOS Lab that I got to work or spent time with, Ahmed Elshamanhory, Alina Kuzembayeva, Anja Engel, Christian Wildner, Christiane Hübner, Christine Cramer, Christian Fabian, Dominik Linzner, Erik Kubaczka, Eike Mentzendorff, Ekaterina Solyus, Fengyu Cai, Gamze Dogali, Gizem Ekinci, Hongfei Liu, Irem Ergenlioglu, Julia Detzer, Jérémie Marlhens, Jacob Christian Mejlsted, Kai Cui, Kilian Heck, Klaus-Dieter Voss, Matthias Schultheis, Maik Molderings, Maleen Hanst, Mark Sinzger, Markus Baier, Melanie Mikosch-Wersching, Mengguang Li, Maximilian Gehri, Nicolai Engelmann, Nikita Kruk, Özdemir Cetin, Philipp Fröhlich, Sascha Hauck, Sikun Yang, Sebastian Wirth, Sofia Startceva, Tim Prangemeier, Yannick Eich and Yujie Zhong.

Thank you, everyone !!

Darmstadt, January 15, 2024

Contents

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 MOTIVATION

Queuing systems have been an area of active research for decades. They play a crucial role in various aspects of our day-to-day lives and are essential for the efficient functioning of many processes and systems. Performance measures of queuing systems such as resource optimization, traffic flow management, waiting time analysis, congestion control, are some of the key features which motivate one to look into them from different aspects. In essence, analyzing queuing systems helps us understand, model, and improve processes across various domains, leading to more efficient and effective systems that positively impact our daily lives.

Load balancing is a very important part of any queuing system, where the incoming jobs (load) needs to be distributed among available queuing resources in a manner that a certain objective is achieved. Objectives in regard to queuing systems include but are not limited to minimizing waiting time of incoming jobs, avoiding job beings dropped, not letting queues be idle, etc. Even though many static state-of-the-art load balancing algorithms exist, such as join-the-shortest-queue (JSQ), shortest-expected-delay (SED), join-the-idle-queue (JIQ), we motivate with the work of this thesis that there is a need for dynamic load balancing algorithms since the computational systems are becoming more and more complex and scalable.

Partial observability and network delays are inherited by large queuing systems, since it is not realistic for the load-balancer to *instantaneously* know the state of *all* the queues at every decision epoch (when a job arrives) as is assumed in algorithms such as JSQ and SED. This further motivates the need to learn scalable policies which take into consideration these challenges while modelling the queuing system, as we have done in this thesis.

The reinforcement learning (RL) framework is a mixture of machine learning and optimal control, which is particularly useful to learn policies in systems where the mathematical model is unknown or infeasible. The RL agent (load-balancer) gathers model information by interacting with the system online and learns a policy by delicately balancing between

exploration and exploitation while optimizing some reward function. Due to the success of RL in various applications, we were also motivated to use it for modelling and solving our partially observable queuing system having network delays. Learning scalable policies in multi-agent systems is tricky because of challenges such as non-stationary, computational complexity, partial observability and credit assignment. Which is why we have used mean-field approximations, more concisely mean-field control since we have cooperative agent setting, to model queuing systems with multiple load-balancers and learn scalable load balancing policies by converting the otherwise hard to solve multi-agent system into a single-agent system with theoretical performance guarantees.

Our contributions though this thesis and the overview of its structure is given next.

1.2 CONTRIBUTIONS AND OVERVIEW

The main focus of this thesis is to model large queuing systems such that scalable load balancing solutions can be learned. The main contributions are as follows:

- We have modelled and analyzed large queuing systems, as single-agent and multi-agent Markov decision processes (MDPs).
- We have considered challenges of partial observability and network delays both in both single-agent and multi-agent models.
- We have considered the case of localized structures within multi-agent systems, since they exist more naturally as compared to systems where agents are fully connected.
- We have used the mean-field control to model large multi-agent queuing systems to learn scalable load balancing policies and provided theoretical guarantees.

The structure of this thesis is given next, with a short description of each chapter.

CHAPTER 2. This is the background chapter which explains the key concepts used throughout this thesis. We first explain the queuing system model used in all our mentioned works and then explain the frameworks of RL, MFC and Bayesian inference which have been used, in one or more of the chapters, to learn load balancing algorithms.

CHAPTER 3. This chapter presents a queuing system with a single load-balancer having delayed information of the queue states. This leads to partial observability of the system, which is modelled as a partially observable Markov decision process (POMDP) and then solved using the MCTS algorithm. Here we look at the performance of policies for different objectives and various distributions modelling the arrival and service rates. Along with a detailed analysis of our proposed load balancing policy, we also used trace data provided by Kaggle to infer the underlying distribution of the arrival and service times of a real system. This chapter is based on the published work

[1] A. Tahir, B. Alt, A. Rizk, and H. Koeppl, "Load balancing in compute clusters with delayed feedback", *IEEE Transactions on Computers*, 2022.

CHAPTER 4. In this chapter, we extend the single-agent queuing system from Chapter 2 to having multiple load-balancers. We have considered an arbitrary network delay, which is needed to ensure synchronization of the load-balancers. In order to obtain scalable learning policy, we have used mean-field control framework and have theoretically proven that the learned policy of the limiting system performs well in our finite queuing system as long as the system size is large enough. This contribution highlights the need for learning a policy for when the synchronization delays is not too low, when join-the-shortest-queue is optimal, or not too high, when random allocation is the best load balancing policy. This work is base on the publication

[2] A. Tahir, K. Cui, and H. Koeppl, "Learning mean-field control for delayed information load balancing in large queuing systems", in *Proceedings of the 51st International Conference on Parallel Processing*, 2022, pp. 1–11.

CHAPTER 5. In this chapter, we extend the multi-agent model of Chapter 4 to a more realistic setup where the agents have only localized interactions and queue accesses. We then leverage the recent advances in sparse mean-field theory to model this localized queuing system, while giving theoretical guarantees. We have used different kinds of graph structure including torus and cube connected cycles to model the queuing system and to learn load balancing policies. This work is based on the publication currently under peer review at the Performance Evaluation journal

[3] A. Tahir, K. Cui and H. Koeppl, "Sparse mean field load balancing in large localized queueing systems", *arXiv preprint arXiv:2312.12973*, pp. 1–22, 2023.

CHAPTER 6. This chapter gives a final discussion of our contributions through this thesis and also provides some interesting future directions.

APPENDICES. The appendices contain two types of topics. Firstly, the existing work on state-of-the-art MCTS algorithm, from which our work of Chapter 3 is adapted. And secondly it contains topics which are fundamental to understanding our work but were too detailed to be put into the background chapter, such as Markov chains, different probability distributions used in this thesis and explanation of the state-of-the-art RL algorithm used in Chapters 4 and 5, proximal policy optimization PPO.

2

BACKGROUND

In this chapter, we present the concepts which are used in this thesis. We first explain a queuing system that has been used in all our works, with some modifications, which are mentioned in the respective chapters. Then we mention reinforcement learning, which has been used by us to learn optimal load balancing strategies for queuing systems. We have also covered mean-field approximations, which is an attractive framework for scalable modeling of large multi-agent systems. Lastly, we mention Bayesian inference, which is needed in real-world queuing systems due to incomplete or delayed information.

2.1 QUEUING SYSTEMS

The basic structure of the queuing system used throughout this thesis is given in Fig. 2.1, and we begin by explaining its key components. On the left-hand side, we have N load-balancers, these are the routers that do job allocations such that the incoming load can be split without overloading any of the available servers. On the right-hand side are the M servers and queues. The queues are finite with some maximum buffer capacity B and

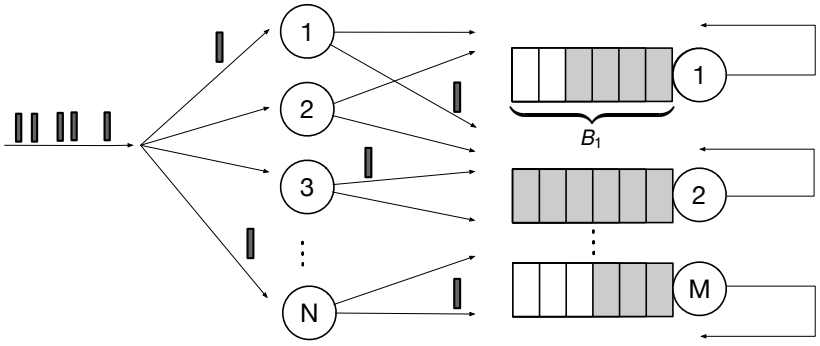

FIGURE 2.1: Queuing system model used in this thesis.

are shown by the slots, where the dark color indicates how many slots of each queue are occupied. For example, the queue 2 is full and cannot accept any more jobs. The servers take the jobs from the queue one at a time to process them. Note that we will use the terms job and packet interchangeably.

Next, we discuss the processes involved in a queuing system. First is the arrival process which defines the rate at which the jobs arrive at the load-balancers for allocation and is characterized by i.i.d inter-arrival times, $\mathbf{a}(n) = \{a(0), a(1), a(2), ...\}$, between the jobs. The average arrival rate is then given as $\lambda = \frac{1}{\bar{A}}$, where $\bar{A}$ is the average inter-arrival time. Second, we have the rate at which each server processes the jobs and is characterized by its service times, $\mathbf{v}(n) = \{v(0), v(1), v(2), ...\}$ and the rate is then given as $\mu = \frac{1}{\bar{V}}$, where $\bar{V}$ is the average service time. Additionally, we also assume that the servers send information about their filled queues to the load-balancers, which is indicated by the arrows going back from the servers. Note that many other variations of queuing systems have been developed and researched over the years. For instance, each server may not have its own designated queue, instead the whole system could have one queue. Or the queues could be at the load-balancer end and not attached to the server. Or each load-balancer could have its own designated arrival stream, etc. More details can be found in [14–16].

Queuing systems are interesting to model and solve since they often exhibit Markovian properties. This allows us to use continuous time Markov chain models to analyze queues, given that the inter-arrival times and service rate are exponentially distributed [15]. For details on Markov chains and their useful properties, see [17]. In Chapters 4 and 5 we have used a Markov modulated Poisson process (MMPP) to model the arrival rate. MMPP is a Poisson process with a variable rate that varies according to a Markov process [18]. It is a more realistic way of modelling time varying network traffic. The key assumptions we have for our queuing systems are (i) each job only takes one slot in the queue, (ii) the queues work in a first-in-first-out manner, (iii) once a job has been processed it leaves the system and (iv) if a job is dropped it is not resent, (v) each server has its own designated queue and (vi) no job has priority over the another.

Various state-of-the-art load balancing strategies, which have also been used in the following chapters, as baselines. These include join-the-shortest-queue (JSQ) [19, 20], shortest-expected-delay (SED) [21, 22] and their power-of-d variants [23–25]. JSQ, assigning the incoming job to the queue with the least amount of jobs, demonstrates high efficiency in reducing job response time in environments where servers are uniform, and service times adhere to independent and identically distributed exponential patterns. Conversely, SED, assigning the incoming job to the queue where it will have the least amount of expected delay, is tailored for heterogeneous servers and performs exceptionally well in situations characterized by heavy traffic. In their power-of-d algorithms, JSQ and SED is performed on only d out of total M queues. If $d = 1$ this can be seen as a random policy and if $d = M$ then it is the original JSQ or SED algorithm. Power-of-d algorithms need less system information and have proven to be optimal, especially in homogenous and large systems. Note that in this context of queuing systems, strategy refers to the decision (action) of the load-balancer of where to allocate the incoming jobs such that some goal is achieved. Throughout this thesis, one of our main goals is to devise scalable load balancing policies (strategies) such that overall jobs dropped in the system are minimized. And to achieve this

we have used RL to learn out-performing strategies while using different scalable modeling techniques since we have countable state spaces with Markovian assumptions.

2.2 REINFORCEMENT LEARNING

The key idea behind RL is for an agent (the load-balancer) to learn from interactions in which action (job allocation) optimizes the goal while exploring a possibly unknown environment (queuing system) [26]. The most commonly used mathematical framework for solving sequential RL problems is a Markov decision process (MDP).

2.2.1 *Markov Decision Process*

An MDP [27] is a discrete-time stochastic control process defined by the tuple $\langle \mathcal{X}, \mathcal{U}, \mathcal{T}, \mathcal{R} \rangle$, where $\mathcal{X}$ is the state space, $\mathcal{U}$ is the action space, $\mathcal{T} : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \to [0, 1]$ is the transition function, $\mathcal{R} : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the reward function. An MDP considers a process $X(t) \in \mathcal{X}$ which is controlled by actions $U(t) \in \mathcal{U}$. Throughout this work, we assume time homogeneous transition function $\mathcal{T}(x', x, u) \coloneqq \mathbb{P}(X(t+1) = x' \mid X(t) = x, U(t) = u)$, where $x', x \in \mathcal{X}$ and $u \in \mathcal{U}$ and countable spaces. It is assumed for an MDP that the agent can always observe the state of the environment. and at each discrete timestep t the agent selects an action, $u(t) \in \mathcal{U}$, based on the state, $x(t) \in \mathcal{X}$, the environment is in. This action comes from the policy, $\pi : \mathcal{X} \times \mathcal{U} \to [0, 1]$ and $\pi(u, x) = \mathbb{P}(U(t) = u \mid X(t) = x)$, which is a mapping from a state to the probability of choosing each possible action in that state. On taking this action the environment gives a reward $r(t+1) = \mathcal{R}(X(t+1), U(t+1))$, at the next timestep, to the agent and transitions to the next state $x(t+1)$ based on the transition function $\mathcal{T}$. The agents' action selection is a crucial balance between exploration of new actions and exploitation of already taken actions which yielded high rewards. The goal of an MDP agent is to learn/find an optimal policy, π^*, which maximizes, usually, the expected discounted sum of rewards, possibly over an infinite horizon, $\pi^* \coloneqq \arg\max_\pi \mathbb{E}[R \mid \pi]$, where $R \coloneqq \sum_{t=0}^{\infty} \gamma^t \mathcal{R}(X(t), U(t))$ is the return over time. $\gamma < 1$ is the discount factor indicating the importance of future rewards as compared to the immediate reward. Solutions to RL problems can be divided into two main approaches, value functions methods and policy search methods. If the model dynamics are accurately known, then both these methods fall under the dynamic programming principle (DPP) method from control theory. DPP is used to find optimal control/policies by breaking down a complex optimization task into smaller sub-problems [28]. It is based on Bellman's principle of optimality which says that as long as the system is time-consistent, its optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision [29].

VALUE FUNCTION: These methods estimate the value of being in a given state x [26]. And the state-value function is the expected return when an agent starts in state x and follows the policy π, given as $Q^\pi(x) \coloneqq \mathbb{E}[R \mid X(0) = x, \pi]$.

Hence, the value function is for a specific policy π from which actions u are sampled. Whereas, the optimal policy π^* is the one which has the optimal state-value function, $Q^*(x) \coloneqq \max_\pi Q^\pi(x)$, for all $x \in \mathcal{X}$. And the optimal policy for a certain state is given as $\pi(x) \coloneqq \arg\max_u \left[r(x,u) + \gamma \sum_{x' \in \mathcal{X}} \mathcal{T}(x', x, u) Q(x')\right]$. Q-learning, policy iteration and value iteration are some methods which can be used to find the optimal policy. For details, see [26].

In RL problems where the transition dynamics are not available, a state-action value function is used, $Q^\pi(x,u) \coloneqq \mathbb{E}\left[R \mid X(0) = x, U(0) = u, \pi\right]$, where now the initial action u for state x is also given, and the policy π is followed from the next state. The best policy can then be chosen, an action u greedily at every state x using: $\arg\max_u Q^\pi(x,u)$. And the state-value function can be retrieved as: $Q^\pi(x) \coloneqq \max_u Q^\pi(x,u)$. Iterative methods such as temporal difference learning, dynamic programming and Monte Carlo sampling can be used to solve for optimal value functions.

POLICY SEARCH: These methods do not use the value function, rather search for an optimal policy π^* directly. The policy is parameterized by some parameters π_θ which are then updated such that the expected return, $\mathbb{E}[R \mid \theta]$, is maximized [30]. The policy is stochastic and represented using a probability distribution from which actions can be sampled directly. Policy gradient and actor-critic methods are common approaches to learn the optimal policy directly [31, 32].

DEEP RL: It is a popular approach for learning the policy, by combining RL with artificial neural networks, allowing for scalable solutions [33]. In deep RL the neural network is trained to learn the optimal policy and/or the optimal value function. In this thesis, we have frequently used the state-of-the-art proximal policy optimization PPO algorithm to find the optimal policy, details of which are given in Appendix C.

MULTI-AGENT MARKOV DECISION PROCESS (MMDP): is the multi-agent extension of MDP, where in addition to the above stated MDP tuple we consider a set of agents $N = \{1, \ldots, n\}$ acting on the environment. Since it is still an MDP, each agent can fully observe the state of the environment. Now given the state, a joint action of all the agents, $\mathbf{u} = [u_1, \ldots, u_n]$, is considered. Note that throughout this thesis, we assume that the agents are working in a cooperative manner.

2.2.2 *Partially Observable Markov Decision Process*

A system where the agent is not able to fully observe the state of the environment can be modeled as a POMDP [34]. In addition to the MDP tuple defined in Section 2.2.1, this framework also considers an observation model defined by a time-homogeneous observation function $\Omega : \mathcal{Y} \times \mathcal{X} \times \mathcal{U} \to [0,1]$, where $Y, \mathcal{X}, \mathcal{U}$ are the observation space, state space and action space, respectively and the observation $\Omega(y,x,u) \coloneqq \mathbb{P}(Y(t) = y \mid X(t) = x, U(t) = u)$ for $y, Y \in \mathcal{Y}$, $x, X \in \mathcal{X}$ and $u, U \in \mathcal{U}$. Since the state is not observable, POMDP also considers the agents' belief, which at any time t is a probability distribution

over the states given the history of observations and actions, i.e., $\forall x(t) \in \mathcal{X}, \boldsymbol{\rho}(x,t) = \mathbb{P}(X(t) = x \mid \mathbf{h}(t))$ where $\mathbf{h}(t) = \{U(0), Y(1), U(1), Y(2), \ldots, U(t)\}$ is the history up till t and $Y \in \mathcal{Y}, U \in \mathcal{U}$. And the policy of the agent is then a mapping from its belief to actions. The agents' belief can be updated using Bayes rule, explained later in Section 2.4. The multi-agent extension of the POMDP framework for cooperative agent settings is known as decentralized partially observable Markov decision process (Dec-POMDP) [35–37]. Deep RL is also used to learn the optimal policy for partially observable systems, especially when the system size is large. And in order to consider the entire history, $\mathbf{h}(t)$, recurrent neural networks are used [38–42].

2.2.2.1 *Semi Markov Decision Process*

A semi Markov decision process (SMDP) is a discrete-time model similar to an MDP except that the time interval between decision epochs is random. Note that, for reference, in an MDP these decision epochs are defined using constant time intervals, e.g., $t = 0, 1, 2, \ldots$. In an SMDP the agent can only take actions at these decision epochs, which occur at times $(\tau_n)_{n\in\mathbb{N}}$. The sojourn time is then defined as the time interval $S(\tau_{n+1}) = \tau_{n+1} - \tau_n$ between transitions from state $x(\tau_n)$ at time $t = \tau_n$ to the next state $x(\tau_{n+1})$ at the next decision epoch time, $t = \tau_{n+1}$. This sojourn time is a positive random variable and given the current state $x(\tau_n)$ and the action $u(\tau_n)$ taken in this state, does not depend on the past states; following the Markov property. The SMDP framework can then be defined using the MDP tuple where now the transition function is defined jointly for the state and sojourn time, $\mathcal{T}(x', s, x, u) = \mathbb{P}(X(\tau_{n+1}) = x', S(\tau_{n+1}) = s \mid X(\tau_n) = x, U(\tau_n) = u)$. Furthermore, a partially observable semi Markov decision process (POSMDP) is an SMDP in which the agents cannot observe the state $x(\tau_n)$ of the environment and work with the received observations $y(\tau_n)$ and their belief over the state, $\rho(\tau_n)$. Note that after some transformations and assumptions, the state-of-the-art algorithms for solving MDPs can be used for SMDPs and POSMDPs, respectively. The key assumption which allows this is that the agent takes an action, receives an observation and updates its belief only at the decision epochs, τ_n. For details, see [43–47].

2.3 MEAN-FIELD APPROXIMATION

Scalability in multi-agent reinforcement learning (MARL) frameworks is a major problem due to its combinatorial nature and sampling complexity, such as Dec-POMDPs [48, 49]. This is why mean-field theory has recently become an attractive solution, where the idea is to let the number of agents go to infinity [50, 51]. The main assumption which allows this approximation is that the agents are permutation invariant, i.e., they are interchangeable and indistinguishable and have similar behaviors. This resulting limiting system, represented by the mean-field distribution, combined with RL has provided tractable control solutions, see [52] for details. Heterogeneous systems have also been recently considered [53–55]. From the MARL perspective, there exist two main branches: (i) mean-field games (MFG) for the non-cooperative agents using Nash equilibrium [50, 51, 56] and (ii) mean-field control MFC for cooperative agents using Pareto optimal strategy [57–60]. Both these mean-field

approximations are independent of the number of agent N and the learned solutions in the limiting systems have shown to provide good approximations in (large) finite systems, both empirically and analytically [56, 61, 62].

Since throughout this thesis we assume the agents to work collaboratively the MFC framework has been used in Chapters 4 and 5, and is explained next.

2.3.1 *Mean-Field Control*

MFC with learning, as compared to MFG is still a relatively uncharted research area, despite its wide range of applications in ride-sharing systems, software systems, traffic light controls, systemic risk assessment, central banking, etc [63–68]. In MFC framework, the agents are permutation invariant, which means they are identical, indistinguishable and interchangeable. The MFC framework is in principle non-Markovian and time-variant and in general the DPP method cannot be applied to it directly, see Section 5.4. However, this can be resolved by using an enlarged joint state-action space, known as *lifting* the space, and by aggregating the underlying dynamics and reward [69].

MATHEMATICAL FORMULATION OF MFC: Similar to an MDP, we first consider a finite system having N (homogeneous) agents which work in a cooperative manner. At every timestep t each agent has its own state $X_i(t) \in \mathcal{X}$ and action $U_i(t) \in \mathcal{U}$, for $i = \{1, \dots, N\}$. However, the agent state evolution now depends on the measurable transition function $\mathcal{T}\colon \mathcal{X} \times \mathcal{P}(\mathcal{X}) \times \mathcal{U} \times P(\mathcal{U}) \to \mathcal{P}(\mathcal{X})$ written as $\mathcal{T}(x, \sigma', u, u') \coloneqq \mathbb{P}(X_i(t) = x, \sigma^N(t) = \sigma', U_i(t) = u, \bar{u}^N(t) = u')$ where $x \in \mathcal{X}$, $\sigma' \in P(\mathcal{X})$, $u \in \mathcal{U}$, $u' \in P(\mathcal{U})$, $\sigma^N(t) \coloneqq \frac{1}{N}\sum_{i=1}^{N} \delta_{X_i(t)}$ is the empirical distribution over states and $\bar{u}^N(t) \coloneqq \frac{1}{N}\sum_{i=1}^{N} \delta_{U_i(t)}$ is the empirical distribution over actions. Similarly, each agent receives a reward based on the reward function $\mathcal{R}\colon \mathcal{X} \times \mathcal{P}(\mathcal{X}) \times \mathcal{U} \times \mathcal{P}(\mathcal{U}) \to \mathcal{P}(\mathbb{R})$ and is given as $r_i(t) \coloneqq \mathcal{R}(X_i(t), \sigma^N(t), U_i(t), \bar{u}^N(t))$.

For the mean-field control system, we take $N \to \infty$ and using law of large numbers we can model the above as MFC. And due to the agents being indistinguishable we need to consider only a single representative agent and the reward function then depends on the average reward of each agent. Now, using the $N \to \infty$ assumption, at every timestep t the state of the representative agent is given by $x \in \mathcal{X}$. And given the probability distribution over the states $\sigma \in \mathcal{P}(\mathcal{X})$, the representative agent takes action $u \sim \pi(x, \sigma)$, where the policy is a mapping from the current state and current state distribution to a distribution over the actions space, $\pi(t)\colon \mathcal{X} \times \mathcal{P}(\mathcal{X}) \to \mathcal{P}(\mathcal{U})$ is measurable. The state of the agent then transitions to the next state $x(t+1) \sim \mathcal{T}(x, \sigma', u, u')$ and the agent receives an *instantaneous* reward $r(t)$. As in Markov decision processes (MDPs), given initial state $x(0)$ and set of policies $\pi = (\pi(t))_{t=0}^{\infty}$ the value function can be written as $Q^{\pi}(x) \coloneqq \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r(t) \mid \mathcal{X}(0) = x, \pi]$. Note that, different from MDPs, in MFC the representative agent interacts with other agents only through the empirical distribution of their states, σ', and their actions u'.

Note that using the MFC framework with lifted state-action space a finite N-agent system is represented by a single representative agent, with similar formulation as that of an MDP, and hence referred to as MFC MDP in this work. For details, see [70, 71].

2.4 BAYESIAN INFERENCE

Bayesian Inference is a statistical inference method which can be used to update the belief when a new observation is received. It is based on the Bayes rules, which is given as follows:

$$\mathbb{P}(x(t) \mid y(1), \dots, y(t)) = \frac{\mathbb{P}(y(1), \dots, y(t) \mid x(t))\mathbb{P}(x(t))}{\sum_{x \in \mathcal{X}} \mathbb{P}(y(1), \dots, y(t) \mid x(t))\mathbb{P}(x(t))}$$

and consists of four components:

- posterior: $\mathbb{P}(x(t) \mid y(1), \dots, y(t))$ is the probability distribution representing our belief over the state x,
- likelihood: $\mathbb{P}(y(1), \dots, y(t) \mid x(t))$ is the probability of the observation $y(1), \dots, y(t)$ given systems' latent state is $x(t)$,
- prior: $\mathbb{P}(x(t))$ is the probability distribution of the state $x(t)$ independent of any observations.
- marginal: $\mathbb{P}(y(t)) = \sum_{x \in \mathcal{X}} \mathbb{P}(y(1), \dots, y(t) \mid x(t))\mathbb{P}(x(t))$ is the marginal probability distribution of the observation. Also referred to as the normalizing constant.

The main task in solving Bayesian models is to compute the expectation over the posterior distribution:

$$\mathbb{E}[g(x(t)) \mid y(1), \dots, y(t)] = \sum_{x \in \mathcal{X}} g(x(t))\mathbb{P}(x(t) \mid y(1), \dots, y(t)), \tag{2.4.1}$$

for a state $x(t)$, measurement $y(t)$ at timestep t and where $g(x(t))\colon \mathbb{R}^n \to \mathbb{R}^m$ is an arbitrary function. However, it is often not possible to compute this integral in closed form, which is where the Monte Carlo (approximation) methods come into play.

2.4.1 *Monte Carlo Approximations*

These are a general class of numerical methods to calculate normalizing constant such as given in Eq. 2.4.1 when they do not have closed-form solutions [72], due to lack of conjugate priors or intractable normalization constants. Another reason for using such numerical methods is when the state space is too large or depends on a large number of parameters, making it computationally expensive to calculate. The key idea is to draw i.i.d. set of samples from a target distribution, $q(\cdot)$ and take their average to estimate the exact posterior distribution.

$$\mathbb{P}(x(t)) \mid y(1), \ldots, y(t)) \approx \frac{1}{K} \sum_{i=1}^{K} x^i(t), \quad x^i(t) \sim q(x(t)) \mid y(1), \ldots, y(t))$$

where in a perfect Monte Carlo approximation $x^i(t)$ are sampled from the posterior distribution $q(x(t)) \mid y(1), \ldots, y(t)) = \mathbb{P}(x(t) \mid y(1), \ldots, y(t)$ itself.

MCMC ALGORITHMS: If it is not possible to sample directly from the posterior distribution $\mathbb{P}(x(t) \mid y(1), \ldots, y(t))$, Markov Chain Monte Carlo (MCMC) algorithms can be used instead [73]. These algorithms generate samples using the Markov chain principle, which is a sequence of dependent random variables that follows the Markov property. Metropolis-Hasting (MH) and Gibbs sampler are examples of the most commonly used MCMC algorithms. MH involves obtaining samples from the true distribution through a random walk within the parameter space, essentially forming a Markov chain. The acceptance or rejection of these samples is determined by evaluating an acceptance probability. Consequently, we generate samples representative of the true posterior distribution, facilitating computations such as empirical estimations of the posterior distribution. Whereas, the Gibbs sampler involves an iterative process that alternates between sampling from the full conditionals of the joint distribution across model parameters and data. See [72, 74, 75] for details on these two and more.

In this thesis, we have used the Hamiltonian Monte Carlo (HMC) in Chapter 3, which is more efficient for sampling from higher dimensional complex models [76, 77]. In HMC, the key idea is to introduce a dynamic auxiliary variable, often referred to as "momentum," which is coupled with the original variables of interest. The dynamics of this joint system are then governed by Hamiltonian mechanics, a concept borrowed from classical physics. The total energy of the system is the sum of the potential energy (associated with the target distribution) and the kinetic energy (associated with the momentum). The algorithm involves simulating trajectories of the joint system using Hamiltonian dynamics. These trajectories provide an efficient exploration of the parameter space, allowing the sampler to move rapidly through regions of high probability. The acceptance of proposed samples is based on a Metropolis-Hastings step that takes into account both the position and momentum of the system.

Another successful method for Bayesian inference is importance sampling, explained next.

SEQUENTIAL IMPORTANCE SAMPLING: Given a generic state space model such as:

$$\begin{aligned} x(t) &\sim \mathbb{P}(x(t) \mid x(t-1)), \\ y(t) &\sim \mathbb{P}(y(t) \mid x(t)), \end{aligned} \tag{2.4.2}$$

where, $x(t) \in \mathbb{R}^n$ is the state and $y(t) \in \mathbb{R}^m$ is the measurement at timestep t, sequential importance resampling (SIS) algorithm can be used to generate importance sampling

approximations for posterior distribution [78, 79]. This importance distribution is quite often assumed to have Markovian properties, which is why the entire history of states $x(0) \dots x(t-1)$ does not need to be used at timestep t. SIS is the sequential version of importance sampling [79].

The SIS algorithm consists of K weighted particles, $\{(w^i(t), x^i(t))\}$ for $i = 1, \dots, K$, where $x^i(t)$ are sampled from an importance distribution $\pi(x(t) \mid y(1), \dots, y(t))$ to represent the posterior (also known as filtering) distribution. Then at every timestep t the distribution in Eq. 2.4.1 can be calculated, based on Eq. 2.4.1, as the weighted average of the K samples:

$$\mathbb{P}(x(t) \mid y(1), \dots, y(t)) \approx \sum_{i=1}^{K} w^i(t)\delta(x(t) - x^i(t)).$$

The algorithm has the following steps:

1. Draw K samples from the prior distribution, $x^i(0) \sim \mathbb{P}(x(0))$, for $i = 1, \dots, K$, and assign them equal weights, $w^i(0) = \frac{1}{K}$.
2. For each timestep $t = 1, \dots, T$ do:
   a) Draw samples from the importance distribution, $x^i(t) \sim \pi(x(t) \mid x^i(t-1), y(1), \dots, y(t))$, for $i = 1, \dots, K$.
   b) Update the weights, $w^i(t) \propto w^i(t-1)\frac{\mathbb{P}(y(t)|x^i(t))\mathbb{P}(x(t)|x^i(t-1))}{\pi(x^i(t)|x^i(t-1),y(1),\dots,y(t))}$ and then normalize them.

Note that we do not need to consider the whole history of state x, since we assumed the importance distribution to have Markov properties.

¬

3

SINGLE-AGENT CONTROL FOR LARGE QUEUING SYSTEMS

In this chapter, we present a load balancing problem in a single-agent system having many parallel queues, based on the published work [1]. In recent years, with the growth rate of single-machine computation speeds stagnating, parallelism has emerged as an effective strategy for harnessing the computational power of multiple machines. Consequently, parallelism has become a critical component of compute cluster architectures [80, 81], primarily because it reduces the computational and storage burden on individual servers [82]. In addition to the capacity aggregation, another significant challenge in the operation of parallel servers lies in the optimization of low latency and minimizing data loss. The pivotal factor in achieving this optimization is the assignment of incoming tasks, referred to as *jobs*, to various serving machines, denoted as *servers*, which may have varying capacities and finite buffer capacity. This allocation of tasks, also referred to as load balancing, to servers is orchestrated by a decision-making entity known as the *load-balancer*. Load balancing poses a fundamental challenge that underpins the design and operation of numerous computing and communication systems, including tasks like job routing within data center clusters, managing multipath communication, handling Big Data processing, and optimizing queuing systems.

Essentially, the decision-making agent (load-balancer) in these scenarios assigns each incoming job to one of the servers, which may be of varying capabilities, with the objective of achieving goals such as load distribution, minimizing average delay, or reducing data loss. A significant challenge in devising optimal load balancing policies lies in the fact that the agent only has partial observability of the consequences of its decisions, often relying on delayed acknowledgments from the served jobs. In this chapter, we introduce a model that addresses this partial observability (PO) issue, specifically addressing load balancing decisions in parallel buffered systems when equipped with limited information derived from delayed acknowledgments.

The multi-agent extension of such a queuing system is studied in Chapter 4.

Related Work

Dynamic load balancing is essential for optimizing the performance of parallel systems, and it has led to the development of several state-of-the-art algorithms, including JSQ, SED, and Power-of-d policies [83–85]. JSQ is particularly effective in minimizing job response time when servers are homogeneous and service times follow independent and identically exponentially distributed patterns [19, 20]. SED, on the other hand, is designed for heterogeneous servers and excels in scenarios with heavy traffic [21, 22]. However, as the number of parallel systems M increases, the assumption that the load-balancer possesses full knowledge of the system state at every decision time becomes less realistic. This state may be the queue length or the required cumulative service times for the waiting jobs at each system. In some systems, such as those involving servers with randomly varying capacities, having prior knowledge of information like service times is not feasible. To address this challenge, researchers have explored alternative approaches, including control theory and emerging machine learning techniques. These methods have found extensive application in the analysis of stochastic queuing networks, aiming to enhance their performance as self-adaptive software systems [86, 87].

In practice, the load-balancer may only observe the consequences of its decisions after a non-deterministic feedback delay, which can result from factors like propagation delay or delayed job processing. Power-of-d policies offer a solution to this challenge. In this approach, a subset of servers, specifically a subset of size $d < M$, is repeatedly selected at random during each decision point. Consequently, JSQ(d) or SED(d) can be applied to this dynamically changing server subset of size d [23]. This policy is further enhanced by incorporating a short-term memory mechanism that retains information about the least filled servers from the previous decision instance [88, 89]. Consequently, instead of selecting a completely new set of d servers at random for each incoming job, the decision-making process combines the newly chosen random subset of servers with knowledge of the least filled servers obtained from the last decision. The underlying assumption of JSQ, SED, and Power-of-d policies is that the load-balancer has immediate and complete knowledge of the system state at each decision point. However, in practical distributed systems, this assumption often does not hold due to non-deterministic and heterogeneous feedback delays. These delays can significantly impact performance metrics like job response times and job drop rates, as only partial information is available to the load-balancer at the decision time. hence, this delay needs to be taken into consideration when modeling a real-time system.

From a control perspective, MDPs, have been extensively utilized as a modeling framework for queuing systems, enabling the achievement of optimal control strategies in both static and dynamic environments [90, 91]. In the context of MDPs, the current feedback, whether delayed or immediate, is typically assumed to be known to the agent [26]. In the study [20], the authors employed an MDP formulation in combination with a stochastic ordering argument to demonstrate that Join-the-shortest-queue (JSQ) maximizes the discounted number of jobs completed in a homogeneous server setting. Another notable work [92] focused on the allocation of customers to parallel queues. They modeled this problem as an MDP with the objective of minimizing the sojourn time for each customer. Their research resulted in the development of a 'separable rule,' which is a generalization of JSQ, designed for queues with heterogeneous servers, considering variations in both server

rates and numbers. It's important to note that their work assumed that the queue filling information was readily available to the decision-making agent without any delay. These studies demonstrate the effectiveness of MDPs as a modeling tool for queuing systems and highlight their application in optimizing system performance under various conditions, including homogeneous and heterogeneous server settings. However, they typically assume full and immediate knowledge of the queue state, which may not hold in practical scenarios with non-deterministic and delayed feedback.

In [93], the authors consider a scenario where the decision-making agent receives precise information about queue lengths, albeit with a delay of k time steps. They model this system as a Markov control model with what they refer to as perfect state information. To achieve this, they expand the state space by including the last known state (the exact queue lengths) and all the actions taken from the last known state up to the point of receiving the next known state. In their research, they focus on solving the flow control problem by regulating the arrivals to a single-server queue. Notably, for the case when the delay parameter k is set to 1, they find that the optimal policy adopts a threshold-based strategy that depends on the last action taken. A similar approach is presented in [94], where they address the single-server flow control problem and obtain results that align with those in [93], particularly when $k = 1$. In this context, the optimal policy for minimizing the discounted number of jobs in a system consisting of two parallel queues is shown to be Join-the-shortest-expected-length. In [95], they introduce a decision-making framework where the agent, at each time step n, possesses knowledge of the exact number of jobs present in each infinite queue at the preceding time step $n - 1$. This setup allows for deterministic decision-making with a one-time slot delay. The state space is augmented to incorporate the actual queue fillings at time $n - 1$, the action taken at time n, and information regarding arrivals at time n. These studies explore various aspects of queuing and flow control problems, considering different levels of information availability and incorporating delays in decision-making processes, leading to the development of specific policies and models tailored to these scenarios.

In this chapter, we consider a scenario where the acknowledgments available to the load-balancer, detailing the number of jobs processed, are subject to random delays. This situation characterizes the system as partially observable, introducing significant complexity akin to a POMDP. While several online and offline algorithms have been developed for solving POMDP models [96], there has been relatively little work on optimizing queuing systems using POMDP approaches. Standard solutions for POMDP models that rely on full-width planning, such as value iteration and policy iteration, tend to perform poorly when the state space becomes large, which is often the case in queuing systems due to the challenges associated with dimensionality and history [34, 97]. To address the scalability and complexity issues inherent in large state-space queuing systems, we employ the Partially Observable Monte Carlo Planning (POMCP) algorithm, which is an extension of the state-of-the-art MCTS algorithm to learn a scalable, optimal policy for a POMDP by constructing online a search tree of histories, h. Each node of the tree now keeps an estimate of a history, called belief state, using a set of particles. The state is then sampled from this belief state. The authors show that as long as the belief state is close to the actual state of the environment, POMCP will be able to learn an optimal policy for the POMDP. Monte Carlo simulations are used for the tree search and belief state

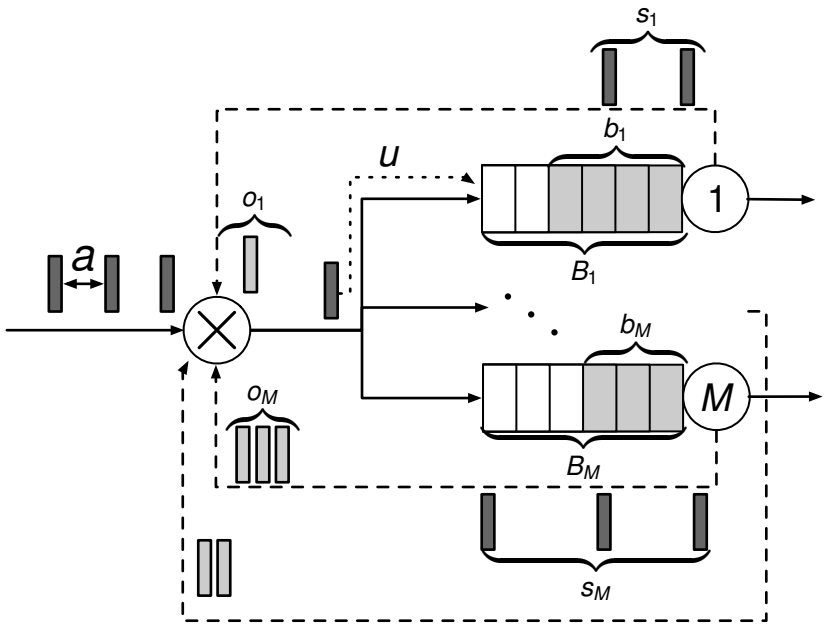

FIGURE 3.1: A parallel queuing system with a load-balancer that maps jobs to servers. The load-balancer observes the inter-arrival times u and the feedback, i.e., the number of acknowledgments, from each server o_i. The load-balancer *does not observe* the queue states b_i, the job service times, or the delayed feedback s_i, i.e., the number of acknowledgments on the way back.

updates. For further details, see [98]. Moreover, our algorithm utilizes a specialized MCTS implementation designed specifically for simulating parallel queuing systems with finite buffers. To handle the delayed feedback acknowledgments in the queuing system, we have developed a sequential importance resampling (SIR) particle filter, which is tailored to address this unique aspect of our problem. We also apply the MCTS approach to solve a POSMDP [46], which is a framework needed when the time intervals between decision epochs are no longer exponential.

3.1 QUEUING SYSTEM MODEL

In this chapter, we consider a system with M parallel servers, where each server has its own finite buffer first-in-first-out (FIFO) queue. The queue filling of each queue i is denoted by $b_i \in \mathcal{B}_i, i = 1, \ldots, M$, where $\mathcal{B}_i = \{0, \ldots, \bar{b}_i\}$ and $\bar{b}_i$ is the maximum buffer capacity for the i-th queue. The servers are assumed to be heterogeneous and the service times of consecutive jobs, $1, 2, \ldots$, at the i-th server, are denoted as $v_i(1), v_i(2) \ldots,$. The jobs are homogeneous and arrive at the load-balancer according to a renewal process which is described by the sequence $(\mathbf{a}(n))_{n\in\mathbb{N}}$, where the job inter-arrival time $a(j) := a(j+1)-a(j)$ is drawn i.i.d leading to an average arrival rate λ. The load-balancer maps each incoming job to exactly one server, and this job is lost if the buffer of the assigned servers' queue is already full. If the job is not lost, it is added to the queue of that server. The average service rate of the i-th server is denoted as μ_i and the service time for each job is random. Once processed, the job leaves the system and the corresponding server sends the load-balancer an acknowledgment. This acknowledgment informs the load-balancer about a slot getting

free in the queue of this server. At every decision time, the load-balancer uses this feedback acknowledgment information to calculate the current buffer fillings, b_i, of each queue and make a more informed decision.

A significant challenge in the job routing process arises when the load-balancer has to make decisions based on delayed acknowledgments from the server. This delay encompasses three key components:

1. The time jobs spend waiting in the queue to which they were assigned.
2. The time required for job processing.
3. The time it takes for the acknowledgments to propagate back to the load-balancer.

The third component introduces considerable complexity to the decision-making problem because decisions not only affect the current system state but also have repercussions on future states due to delayed feedback. *It is important to note that the load-balancer relies solely on these observed acknowledgments to make decisions, rendering the system state partially observable (PO).* In this context, we use the notation o_i to represent the number of acknowledgments from the i-th server that the load-balancer observes within one inter-arrival time. Additionally, we denote s_i as the delayed feedback, which signifies the number of acknowledgments that are currently en route back to the load-balancer but have not yet reached. Since the load-balancer operates under partial observability (PO), it does not have direct access to critical pieces of information, including (i) the queue states b_i, (ii) the job service times v_i, or (iii) the delayed feedback s_i. The lack of direct observation of this information complicates the decision-making process, requiring innovative approaches to address the challenges posed by partial observability and delayed feedback. See Fig. 3.1 for further visualization.

Markov Decision Process with Partial Observability

In order to find the optimal load balancing policy for our system having delayed acknowledgments, we model it as partially observable MDP which is a controlled Markov process with the exact state of the process being latent, see Section 2.2.2 for details. In this chapter, we consider countable state $\mathcal{X}$, action $\mathcal{U}$ and observation $\mathcal{Y}$ spaces. Additionally, we consider discrete decision-making epochs at time points $t \in \mathbb{N}_0$, where the clock given by t is an event clock and not a wall-clock time. Then, if the inter-arrival time, a, between the decision epochs t and the service times, v, are exponentially distributed, this partially observable Markov process can be modeled as a POMDP, since the system Markovian. While, for non-exponential distributed a and/or v it can be modeled as a POSMDP, under the condition that the decision-making is done only at these decision epochs t, [46], since the underlying process is now semi-Markov. For further details on POSMDP see Section 2.2.2.1.

At every decision epoch t, we consider a latent process $\mathbf{x}, \mathbf{X} \in \mathcal{X}$ that can be controlled by actions $u \in \mathcal{U}$. Since the state is latent, only observations $\mathbf{y}, \mathbf{Y} \in \mathcal{Y}$ are available to the agent (load-balancer). The state transition function $\mathcal{T}(\mathbf{x}', \mathbf{x}, u) := \mathbb{P}(\mathbf{X}(t+1) = \mathbf{x}' \mid \mathbf{X}(t) = \mathbf{x}, U(t) = u)$ is the conditional probability of moving from state $\mathbf{x}$ under action

u to a new state $\mathbf{x}'$ at the decision epoch $t+1$. The observation function $\Omega(\mathbf{y}, \mathbf{x}, u) := \mathbb{P}(\mathbf{Y}(t) = \mathbf{y} \mid \mathbf{X}(t) = \mathbf{x}, \mathbf{U}(t) = u)$ denotes the conditional probability of observing acknowledgment $\mathbf{y}$ under the latent state $\mathbf{x}$ and action u. On performing the action $u(t)$ the agent receives a reward $r(t) := \mathcal{R}(\mathbf{X}(t+1), \mathbf{X}(t), \mathbf{U}(t))$, which it tries to maximize over time using a policy $\pi(u, x) := \mathbb{P}(U(t) = u \mid \mathbf{X}(t) = \mathbf{x})$. We consider an infinite horizon objective, where the optimal policy π^* is found by maximizing the expected total discounted future reward

$$\pi^* := \arg\max_{\pi} \sum_{t=0}^{\infty} \mathbb{E}_{\pi}[\gamma^t r(t)], \tag{3.1.1}$$

with discount factor $\gamma < 1$.

Since the current state is not directly accessible by the agent, it has to rely on the action-observation history sequence, $\mathbf{h}(t) = \{U(0), \mathbf{Y}(1), U(1), \mathbf{Y}(2), \ldots, U(t)\}$, up to the current decision epoch t, where $\mathbf{Y} \in \mathcal{Y}, U \in \mathcal{U}$. The policy is then the conditional probability of choosing action u under action-observation history $\mathbf{h}$, $\pi(u, h) := \mathbb{P}(U(t) = u \mid \mathbf{h}(t))$. As the policy is defined as a function of the observation-action history of the agent, this makes it very challenging since keeping a record of an exponentially increasing history sequence over time, $\mathbf{h}$, is not feasible. One popular way is to represent this history in terms of the belief state, $\boldsymbol{\rho}(t) \in \Delta^{|X|}$, where $\Delta^{|\mathcal{X}|}$ is an $\mathcal{X}$ dimensional probability simplex, $\boldsymbol{\rho}(t) = [\rho_1(t), \ldots, \rho_{|X|}(t)]^\top$ and the components $\rho_X(t)$ are the filtering distribution $\rho_X(t) = \mathbb{P}(\mathbf{X}(t) = x \mid \mathbf{h}(t) = h)$. However, if the state space $|\mathcal{X}|$ is huge, this will be a very high-dimensional vector. In order to break the curse of *history* and the curse of *dimensionality*, a particle filter [99, 100] can be used to represent the belief state $\boldsymbol{\rho}(t)$ of the system at time t and is explained next.

Particle filter

In this chapter, we have used the state-of-the-art SIR particle filter, to represent the belief state $\boldsymbol{\rho}(t)$ of the system and update this belief using Monte Carlo simulations based on the action taken and observations received. A detailed explanation of a standard particle filter can be found in Section 2.4.1.

SIR improves on the SIS algorithm, explained in Section 2.4.1, by tackling the degeneracy problem of particles [101]. It does so by resampling K new particles from a discrete distribution that is defined by the normalized weights. This resampling does not need to be done at every timestep but only at certain timesteps or according to some method, such as adaptive resampling [102]. The SIR algorithm is given as:

The algorithm has the following steps:

1. Draw K samples from the prior distribution, $\mathbf{x}_i(0) \sim \mathbb{P}(\mathbf{x}(0))$, for $i = 1, \ldots, K$, and assign them equal weights, $w_i(0) = \frac{1}{K}$.
2. For each timestep $t = 1, \ldots, T$ do:
    a) Draw samples from the importance distribution, $\mathbf{x}_i(t) \sim \pi(\mathbf{x}(t) \mid \mathbf{x}_i(t-1), \mathbf{y}(1) \ldots \mathbf{y}(t)$, for $i = 1, \ldots, K$.

b) Update the weights, $w_i(t) \propto w_i(t-1)\frac{\mathbb{P}(\mathbf{y}(t)|\mathbf{x}_i(t))\mathbb{P}(\mathbf{x}(t)|\mathbf{x}_i(t-1))}{\pi(\mathbf{x}_i(t)|\mathbf{x}_i(t-1),\mathbf{y}(1)\ldots\mathbf{y}(t)}$ and then normalize to unity sum.

c) If effective number of particles n_e is significantly less than K do resampling, where n_e can be calculated as: $n_e \approx \frac{1}{\sum_{i=1}^{K}(w_i(t))^2}$ [102].

Using this particle filter, the posterior distribution can then be approximated as: $\mathbb{P}(\mathbf{x}(t) \mid \mathbf{y}(1)\ldots\mathbf{y}(t)) \approx \sum_{i=1}^{K} w_i(t)\delta(\mathbf{x}(t) - \mathbf{x}_i(t))$ and how good is this approximation will depend on the importance distribution chosen.

3.2 LOAD BALANCING WITH DELAYED ACKNOWLEDGMENTS

We now explain how we use the aforementioned Markov decision processes, POMDP and POSMDP, and the SIR particle filter to model our partially observable queuing system.

Our system consists of a load-balancer and M parallel finite queues, where each queue has its own servers (cf. Fig. 3.1). The complete state of the system can then be defined as $\mathbf{x} \in \mathcal{X}$, using $\mathbf{x} = [\mathbf{x}_1, \ldots, \mathbf{x}_M]^\top$. Here $\mathbf{x}_i$ is the augmented state of the i-th queue that has three components $\mathbf{x}_i = [b_i, s_i, o_i]^\top$, where $b_i \in \mathcal{B}_i$ denotes the current buffer filling at the queue i, $s_i \in \mathcal{B}_i$ denotes the number of delayed acknowledgments for the jobs executed by the server i but not observed by the load-balancer in the current epoch, and $o_i \in \mathcal{B}_i$ denotes the number of acknowledgments *observed* by the load-balancer in the current epoch. Note that an epoch corresponds here to one inter-arrival time. Hence, the state space is $\mathcal{X} \subseteq {\mathbb{N}_0}^{3M}$ and an action $u \in \mathcal{U}$, with $|\mathcal{U}| = M$ corresponds to sending a job to the u-th server. An observation $\mathbf{y} \in \mathcal{Y}$ is the vector of observed acknowledgments at the load-balancer, with the observation space being $\mathcal{Y} \subseteq {\mathbb{N}_0}^{M}$.

3.2.1 *Dynamical System Model*

The probabilistic graphical model for our delayed queuing model is depicted in Fig. 3.2. As mentioned earlier, in case of the POMDP model, the time between decision epochs t in Fig. 3.2 is exponentially distributed, while for POSMDP it can be non-exponential. As we are using Monte Carlo simulations to solve the PO system, the transition probabilities do not have to be defined explicitly. Therefore, we define the transition function indirectly as a generative process, which is explained later.

The load-balancer (our decision-making agent) makes an allocation decision at each job arrival, where the inter-arrival times of jobs $a(j)$ are i.i.d, with $j \in \mathbb{N}_0$ and the decision epochs are denoted by $t \in \mathbb{N}_0$. In order to characterize the stochastic dynamics, we first determine the random behavior of the number of jobs $\tilde{k}_i$ that leave the i-th queue during an inter-arrival time. Naturally, $\tilde{k}_i$ is constrained by how filled is queue i at that time, given by b_i. So, $\tilde{k}_i = \min(k_i, b_i)$, where k_i is the number of jobs that *can be served* in a decision epoch, which is determined by the inter-arrival time and the service time of the i-th server. Then the generative model for the queuing dynamics can be defined as:

$$\mathbf{b}' = \min(\max(\mathbf{b} - \mathbf{k}, \mathbf{0}) + \mathbf{e}_u, \bar{\mathbf{b}}), \tag{3.2.2}$$

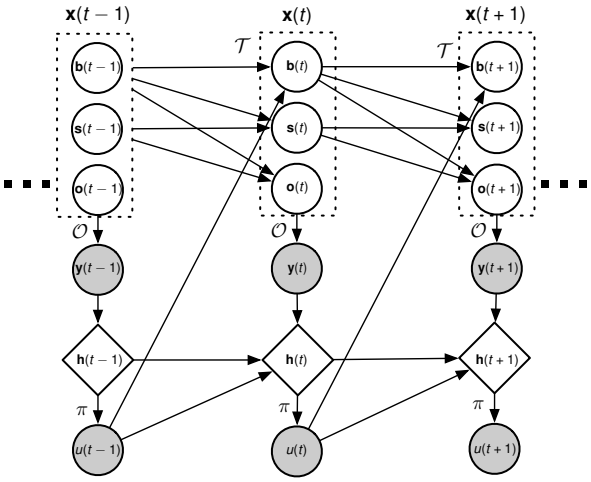

FIGURE 3.2: Probabilistic graphical model of the partially observable queuing system with delayed acknowledgments. Shown are three time slices, where gray nodes depict observed quantities and diamond-shaped nodes denote deterministic functions.

where $\mathbf{b}$ denotes the M size vector of the queuing system at some arrival time point, $\mathbf{b}'$ is the M size vector of the queuing system at the next epoch, $\mathbf{k}$ is the non-truncated vector of number of jobs that can be served at each queue and $\mathbf{e}_u$ is a M size vector of all zeros, except the u-th position is set to one to indicate a mapping of the incoming job to the u-th server. We use $\bar{\mathbf{b}}$ as the vector of maximum buffer sizes for all queues, and $\min(\cdot, \cdot)$ and $\max(\cdot, \cdot)$ denote the element-wise minimum and maximum operation.

As the *load-balancer only observes the job acknowledgments*, we use an augmented state space, $\mathbf{x}$, using the following stochastic update equation:

$$\mathbf{x}' = \begin{bmatrix} \mathbf{b}' \\ \mathbf{s}' \\ \mathbf{o}' \end{bmatrix} = \begin{bmatrix} \min(\max(\mathbf{b} - \mathbf{k}, \mathbf{0}) + \mathbf{e}_u, \bar{\mathbf{b}}) \\ \min(\mathbf{b}, \mathbf{k}) + \mathbf{s} - \mathbf{l} \\ \mathbf{l} \end{bmatrix}, \tag{3.2.3}$$

where $\mathbf{l}$ is the vector containing the number of jobs which are observed by the load-balancer for each queue at the current epoch, $\mathbf{s}$ is the number of unacknowledged jobs from the previous epoch which is updated by removing the observed jobs $\mathbf{l}$ and adding the newly generated acknowledgments of the served jobs given by $\min(\mathbf{b}, \mathbf{k})$. $\mathbf{l}$ is calculated using the delay model given next.

3.2.2 *Delay model*

We assume that the number of jobs that can be served in one inter-arrival time is distributed as $k_i \sim f(k_i)$ and we choose a delay model, where

$$L_i \mid b_i, k_i, s_i \sim \mathrm{Bin}\left(\min(b_i, k_i) + s_i, p_i\right). \tag{3.2.4}$$

Here, $\min(b_i, k_i)$ is the number of jobs leaving the i-th queue at the current epoch, s_i is the number of jobs from the i-th queue for which no acknowledgments have been previously observed by the load-balancer, p_i is the probability that an acknowledgment is received by the load-balancer in the current epoch and L_i is the distribution from which l_i is sampled for Eq. (3.2.3).

We have chosen the binomial distribution because it generally captures the fact that only a subset of the sent, $\min(b_i, k_i) + s_i$, acknowledgments are successfully observed at the load-balancer in the current epoch. At the cost of simplifying the usually correlated delays of jobs, this model helps to obtain tractable results. For example, $p_i = 0.6$ would mean that out of all the acknowledgments sent by queue i, only 60% are expected to be received by the load-balancer in that epoch while 40% are expected to be delayed to future epochs. Similarly, $p_i = 1$ would then represent the case of no delay. Note that any other distribution describing the delay model can be used here and numerically evaluated. The number of acknowledgments from all servers that are not observed in this epoch are accounted for in the next epoch in $\mathbf{s}'$. The previously introduced observations $\mathbf{z}$ for the load-balancer are essentially the received acknowledgments, i.e., $\mathbf{y} = \mathbf{o}' = \mathbf{l}$. Since only the vector $\mathbf{o}'$ of the state $\mathbf{s}'$ (Eq. (3.2.3)) is observed by the load-balancer, a partial observability is established. We note that one limitation of this model is due to the delay independence assumption that lies below the used Binomial distribution.

3.2.3 *Job acknowledgment Distribution*

Next, we discuss how to quantify the distribution of the number of jobs k_i that can be served at the i-th queue in one inter-arrival time. The marginal probability

$$f(k_i) = \int_0^\infty f(k_i \mid a) f(a) da, \tag{3.2.5}$$

can be computed by noting [103]

$$f(k_i \mid a) = \mathbb{P}\left(\bar{V}_i^{k_i} \leq a\right) - \mathbb{P}\left(\bar{V}_i^{k_i+1} \leq a\right), \tag{3.2.6}$$

with $\bar{v}_i^{k_i} = \sum_{m=1}^{k_i} v_i^m$ for the i-th queue. For the POMDP model, with exponentially distributed inter-arrival times $a \sim \mathrm{Exp}\,(\lambda)$, with rate parameter λ, and exponential service times for all servers $v_i \sim \mathrm{Exp}\,(\mu_i)$, with rate parameter μ_i, the distribution $f(k_i)$ can be calculated in closed form. Since the service process corresponds to a Poisson process, we find the conditional distribution $f(k_i \mid a)$ as $k_i \mid a \sim \mathrm{Pois}\,(\mu_i a)$. Carrying out the integral, in Eq. (3.2.5), we find the number of jobs that can be served in one inter-arrival time follow a Geometric distribution $k_i \sim \mathrm{Geom}\,(\frac{\lambda}{\mu_i+\lambda})$, with $\frac{\lambda}{\mu_i+\lambda}$ denoting success probability, [103].

For the POSMDP model with general inter-arrival and service time distributions, closed form expressions for the marginal probability above are often not available, since this would require closed form expressions for a k-fold convolution of the probability density function (pdf) of the service times. Note that some corresponding general expressions exist as Laplace transforms, where the difficulty is passed down to calculating the inverse

transform. Therefore, in such cases we will resort to a sampling based scheme for the marginal distribution $f(k_i)$, i.e.,

$$\begin{aligned} &A \sim f(a) \\ &v_i^m \sim f(v_i), \quad m = 1, 2, \dots \\ &\mathcal{K}_i = \left\{ j \in \mathbb{N}_0 : \sum_{m=1}^{j} v_i^m \leq A \right\} \\ &k_i = \max(\mathcal{K}_i). \end{aligned} \tag{3.2.7}$$

In this numerical solution, we draw a random inter-arrival time a and count the number k_i of service times that fit in this inter-arrival interval. Note that the number of jobs $k_i(t)$ at a decision epoch t is not necessarily independent of the number of jobs $k_i(t+1)$ in the next interval. The impact of this effect can be well demonstrated when the service time distribution is, e.g., heavy tailed. Also note that the exact modeling of this behavior would, in general, require an extended state space incorporating this memory effect. Therefore, the sampling scheme can be seen as an approximation to the exact system behavior.

3.2.4 *Inferring Arrivals and System Parameters*

For the load-balancer to be deployed in an unknown environment, we may require an estimate of the inter-arrival and/or service rate densities. For this purpose, we will resort to a Bayesian estimation approach to infer the densities $f(a)$ and $f(v_i)$. We select a likelihood model $f(\mathcal{D} \mid \theta)$ for the data generation process and a prior $f(\theta)$, with model parameters Θ. We assume we have access to data $\mathcal{D} = \{d(1), d(2), \dots, d(n)\}$, where $d(j)$ is the inter-arrival time between the j-th and $j+1$-st job arrival event that is observed by the load-balancer or the service times for each server. For inference-based load balancing, we use the inferred distribution of the inter-arrival times as in the posterior predictive $f(d^* \mid \mathcal{D}) = \int f(\theta \mid \mathcal{D}) f(d^* \mid \theta) d\theta$, of a new data point d^*, which is then used in the sampling simulator, see Eq. (3.2.7). The same can be done with the data for the service times. We will now describe some models of different complexities for data generation. Since, most models do not admit a closed form solution, we resort to a Monte Carlo sampling approach to sample from the posterior predictive [73, 74], see also Section 2.4.1.

EXPONENTIAL INTER-ARRIVAL TIMES: Here, we briefly show the calculation for the posterior distribution and posterior predictive distribution for renewal job arrivals with exponentially distributed inter-arrival times. For the likelihood model, we assume

$$D_j \mid m \sim \mathrm{Exp}\,(m), \quad j = 1, \dots, n$$

where m is the rate parameter of the exponential distribution. We use a conjugate Gamma prior $M \sim \mathrm{Gam}\,(\alpha_0, \beta_0)$. Hence, the posterior distribution is

$$M \mid \mathcal{D} \sim \mathrm{Gam}\,(\alpha_0 + n, \beta_0 + \sum_{j=1}^{n} d_j)$$

And the posterior predictive distribution is found as

$$D^* \mid \mathcal{D} \sim \mathrm{Par}\,(\alpha_0 + n, \beta_0 + \sum_{j=1}^{n} d_j),$$

where $\mathrm{Par}\,(\alpha, \beta)$ denotes the Pareto distribution.

GAMMA DISTRIBUTED INTER-ARRIVAL TIMES: In case of a gamma likelihood of the form

$$D_j \mid \alpha, \beta \sim \mathrm{Gam}\,(\alpha, \beta), \quad j = 1, \ldots, n$$

we use independent Gamma priors for the shape and the rate, with $A \sim \mathrm{Gam}\,(\alpha_0, \beta_0)$ and $B \sim \mathrm{Gam}\,(\alpha_1, \beta_1)$. Finally, we sample from the posterior predictive using Hamiltonian Monte Carlo (HMC) [76], which can be implemented using a probabilistic programming language, e.g. using PyMC3 [104].

SERVICE TIMES DISTRIBUTED AS AN INFINITE GAMMA MIXTURE: Here, we present a framework to non-parametrically infer the posterior distribution. We use an approximate Dirichlet process mixture model, which can be regarded as an infinite mixture model [105]. We use a gamma distribution for the observation model

$$\phi(d \mid \theta_i) \propto d^{\epsilon_i^1 - 1} e^{-\epsilon_i^2 d}, \tag{3.2.8}$$

with mixture parameters ϵ_i^1 and ϵ_i^2. For the base measure G_0, i.e., the prior distribution of the mixture parameters, we use $G_0 = F_{E_i^1} \times F_{E_i^2}$, with $E_i^1 \sim \mathrm{Gam}\,(1, 1)$ and $E_i^2 \sim \mathrm{Gam}\,(1, 1)$.

The truncated stick-breaking approximation is then given by

$$\begin{aligned}
&M \sim \mathrm{Gam}\,(1, 1), \quad E_i^2 \mid m \sim \mathrm{Beta}\,(1, m), \quad i = 1, \ldots, c-1 \\
&W_i = \beta_i \prod_{j=i-1}^{i} (1 - E_j^2), \qquad i = 1, \ldots, c-1 \\
&W_c = 1 - \sum_{j=1}^{c-1} W_j, \quad \Theta_i \sim G_0 \\
&D^{(j)} \mid w_1, \ldots, w_c, \theta_1, \ldots, \theta_c \sim \sum_{i=1}^{c} w_i \phi(d \mid \theta_i), \; j = 1, \ldots, m,
\end{aligned}$$

which corresponds to a mixture of Gamma pdfs. Here too, samples from the posterior predictive can be efficiently generated using HMC. For the truncation point, the number of components c in the formula above can be assessed using

$$c = \lceil 2 - \mathsf{E}[M] \log(\epsilon) \rceil = \lceil 2 - \log(\epsilon) \rceil, \tag{3.2.9}$$

where ϵ is an upper bound on the total variation distance between the exact and truncated approximation. For example, we can choose $\epsilon = 10^{-12}$, which corresponds to $c = 30$ components.

A detailed explanation of the above-mentioned distributions is given in Appendix A.1.

3.2.5 *Reward Function Design*

A main difference of our approach to *explicitly* defining a load balancing algorithm is that we provide the algorithm designer with the flexibility to set different optimization objectives for the load-balancer and correspondingly obtain the optimal policy by solving the POMDP or the POSMDP. This is carried out through the design of the reward function $\mathcal{R}$ as defined in Sections 2.2 and 3.1. The optimal policy π^* maximizes the expected discounted reward as $\pi^* \coloneqq \arg\max_\pi \sum_{t=0}^{\infty} \mathsf{E}_\pi[\gamma^t r(t)]$ where $r(t) = \mathcal{R}(\mathbf{X}(t+1) = \mathbf{x}', \mathbf{X}(t) = \mathbf{x}, U(t) = u)$, with $\gamma < 1$ and $\mathbf{x}', \mathbf{x}$ are defined by Eq. 3.2.3. In the following, we discuss several reward functions $\mathcal{R}$ in the context of mapping incoming jobs to the parallel finite queues, see Fig. 3.1.

Minimize queue lengths

A reward function which aims to minimize the overall number of jobs waiting in the system. This objective can be formalized as

$$\mathcal{R}(\mathbf{x}', \mathbf{x}, u) = -\sum_{i=1}^{M} b_i \tag{3.2.10}$$

as it takes the sum of all queue fillings. Similarly, a polynomial or an *exponential* reward function, such as

$$\mathcal{R}(\mathbf{x}', \mathbf{x}, u) = -\sum_{i=1}^{M} \chi^{b_i} \tag{3.2.11}$$

For a fixed overall number of jobs in the system and $\chi > 1$, this objective tends to balance queue lengths, e.g., if total jobs in the system are 10 and $M = 2$ then an allocation of $[5, 5]$ jobs will have much higher reward than $[9, 1]$ allocation. Using the *variance* amongst the current queue fillings also balances the load on queues. The reward function is then given as:

$$\mathcal{R}(\mathbf{x}', \mathbf{x}, u) = \mathrm{Var}(b_i, \ldots, b_M) \tag{3.2.12}$$

Note, however, that balancing queue lengths does not necessarily lead to lower delays if the servers are heterogeneous. Hence, *proportional allocation* provides more reward when jobs are mapped to the faster server as

$$\mathcal{R}(\mathbf{x}', \mathbf{x}, u) = -\sum_{i=1}^{M} \frac{b_i}{\mu_i} \tag{3.2.13}$$

Minimize loss events

To prevent job losses, we can also formulate a reward function that penalizes actions that lead to fully filled queues, i.e.,

$$\mathcal{R}(\mathbf{x}', \mathbf{x}, u) = -\sum_{i=1}^{M} \mathbb{1}_{b_i = \bar{b}_i} \tag{3.2.14}$$

The indicator function evaluates to one only when the corresponding queue is full.

Minimize idle events

One might also require that the parallel system remains work-conserving, i.e., no server is idling, as this essentially wastes capacity. Hence, in the simplest case we can formulate a reward function of the form

$$\mathcal{R}(\mathbf{x}', \mathbf{x}, u) = -\sum_i \mathbb{1}_{b_i=0} \tag{3.2.15}$$

Note that some of the reward functions above can be combined, e.g., in a weighted form such as the following.

$$\mathcal{R}(\mathbf{x}', \mathbf{x}, u) = -\left[\sum_{i=1}^{M} b_i + \kappa \mathbb{1}_{b_i=\bar{b}_i}\right], \tag{3.2.16}$$

where, b_i is the buffer state of the i-th queue at timestep t after taking action u and the constant weight $\kappa > 0$ is used to scale the impact of the events of job drops to the impact of the buffer filling on the reward.

3.3 PARTIAL OBSERVABILITY LOAD-BALANCER: A MONTE CARLO APPROACH FOR DELAYED ACKNOWLEDGMENTS

In this section, we outline our approach to solve the partially observable system for the job routing problem in parallel queuing systems with delayed acknowledgments. Our solution is an alternate technique to Dynamic Programming and is based on a combination of the MCTS algorithm [98] and SIR particle filter. A detailed explanation of how the MCTS algorithm works is given in Appendix B.1, whereas the SIR filter was explained previously in Section 3.1.

The reason for choosing an MCTS algorithm is that load balancing problems, like the one presented in this chapter, can span to very large state spaces. In these scenarios, solution methods based on dynamic programming [28] often break due to the *curse of dimensionality*. MCTS solves this problem by using a sampling based heuristic approach to construct a search tree to represent different states of the system, the possible actions in those states and the expected value of taking each action. In recent years, these techniques have been shown to yield exceptional results in solving very large decision-making problems [106, 107].

One main contribution of this chapter lies in the design of a simulator, $\mathcal{G}$, which incorporates the properties of the queuing model discussed above, into the algorithm. The simulator $\mathcal{G}$,

provides the next state $\mathbf{x}(t+1)$, the observation $\mathbf{y}(t+1)$ and the reward $r(t+1)$, when given the current state $\mathbf{x}(t)$ of the system and the taken action $u(t)$ as input,

$$\mathbf{x}(t+1), \mathbf{y}(t+1), r(t+1) \mid \mathbf{x}(t), u(t) \sim \mathcal{G}(\mathbf{x}, u). \tag{3.3.17}$$

This simulator $\mathcal{G}$, is used in the MCTS algorithm to rollout simulations of different possible trajectories in the search tree. Each trajectory is a path in the search tree starting from the current belief state of the system and expanding (using $\mathcal{G}$) to a certain depth. While traversing through the search tree, the trajectories (actions) are chosen using the upper confidence bounds for trees (UCT), which is an improvement over the greedy-action selection [108]. In UCT the upper confidence bounds guide the selection of the next action by trading off between exploiting the actions with the highest expected reward up till now and exploring the actions with unknown rewards.

At every decision epoch POL starts with a certain belief on the state of the system, $\boldsymbol{\rho}(t)$, see Section 3.1 for a formal definition, which is represented with particles and also used as the root of the search tree. Starting from the root, i.e., the current belief, the search tree uses UCT to simulate the system for a given depth, after which the action u with has the highest expected reward is chosen. The trajectories for all other actions are then pruned from the search tree since they are no longer possible. This is done to avoid letting the tree grow infinitely large.

Once the action u is taken and the job is allocated to a certain queue, the load-balancer receives real observations, $\mathbf{y} = \mathbf{l}$, from the system. These observations are the randomly delayed acknowledgments from the servers. The load-balancer then uses the received observations as an input to the SIR particle filter, in order to update its belief of the state the system is in now. The weights given to each particle (state), $w(s_i) = \mathbb{P}(\mathbf{y}_i|\mathbf{x}_i)$, while *resampling* in the SIR particle filter were designed to incorporate our queuing system and its delay model, Eq. (3.2.4), where the new samples $\mathbf{x}_i$ are drawn from the simulator $\mathcal{G}$. After applying the SIR filter, we will have the new set of particles representing the current belief state of the system, $\boldsymbol{\rho}(t)$. These particles are then used to sample the states for simulating the search tree and finding the optimal action at the next epoch. Note that for POL receiving observations, action selection and belief update all happen at each decision epoch, which is why it is possible to model the system as a POSMDP [46] as well. Note that it is shown in Theorem 1 of [98] for a POMDP and in Theorem 2 of [109] for a POSMDP, that the MCTS converges to an optimal policy.

To summarize, at every job arrival, POL simulates the tree from the root. The root contains the current set of belief particles. POL then acts on the real environment using the action which maximizes the expected value at the root. On taking the action, POL gets a real observation. This observation is the acknowledgment that is subject to delays. Using this observation and a SIR particle filter, POL updates its set of particles (belief state) of the system for the next arrival. The pseudo-code of the working of our proposed POL is given in Algorithm 3.3 and the code is also provided as open source[1] for reproducibility. In Algorithm 3.3, Q is the real world representation of the queuing network, T_m is the number of Monte Carlo simulations done, T_e is the number of jobs arriving in each Monte Carlo simulation, R_e collects the reward for each epoch, K_x and W_x are the set of particles and their corresponding weights, respectively.

1 https://github.com/AnamTahir7/Partially-Observable-Load-Balancer

input : $N, \lambda, \mu_1 \dots \mu_N, \mathcal{G}, \eta, \boldsymbol{\rho}(0), \mathcal{R}, \mathbf{x}(0), \kappa, b_1 \dots b_N, T_m, T_e, Q$
output : R_{avg}, average reward for each time step T_e

Initialize $R_m \rightarrow 0$

for $t = 0, 1, \dots, T_m$ **do**
  Initialize tree $\boldsymbol{\Psi}(\mathbf{0}), R_e, Q$
  for $t = 0, 1, \dots, T_e$ **do**
    $\boldsymbol{\Psi}(\boldsymbol{t+1})$ = SimulateTree($\boldsymbol{\Psi}(\boldsymbol{t}), \mathcal{G}$)
    $u(t+1) \rightarrow \arg\max_a \mathcal{R}(\mathbf{x}(t), u(t))$
    $\mathbf{y}(t+1), \mathbf{x}(t+1), r(t+1)$ = $Q(u(t+1))$
    $R_e \rightarrow R_e \cup r(t+1)$
    $\boldsymbol{\rho}(t+1)$ = UpdateBeliefandTree($\boldsymbol{\Psi}(\boldsymbol{t+1}), \mathbf{y}(t+1), u(t+1), \mathcal{G}$)
  end
  $R_m \rightarrow R_m + R_e$
end
$R_{\text{avg}} = \frac{R_m}{T_m}$
return R_{avg}

Function `UpdateBeliefandTree` $(\boldsymbol{\Psi}, \mathbf{y}, u, \mathcal{G})$**:**
  Initialize $K_x = \{\}, W_x = \{\}$
  repeat
    $\mathbf{x} \sim \boldsymbol{\Psi}$(root),
    $\mathbf{x}', \mathbf{y}', r' \sim \mathcal{G}(\mathbf{x}, u)$,
    $w_x = \text{Bin}\,(\mathbf{x}'|\mathbf{y}', u)$
    $K_x \rightarrow K_x \cup \mathbf{x}', W_x \rightarrow W_x \cup w_x$
  until *Timeout()*
  Resample particles K_x according to weights W_x
  Update root and prune tree $\boldsymbol{\Psi}$
return $\boldsymbol{\Psi}$

Function `SimulateTree` $(\boldsymbol{\Psi}(\boldsymbol{t}), \mathcal{G})$**:**
  See the pseudo-code in [98] and Appendix B.2
return $\boldsymbol{\Psi}(t+1)$

FIGURE 3.3: Pseudo-code for working of POL.

3.4 EVALUATION: SIMULATIONS AND REAL-WORLD EXPERIMENT

In the following, we show numerical evaluation results for the proposed Partial Observability Load-Balancer (POL), under randomly delayed acknowledgments. Recall that if the acknowledgment is not observed in the current inter-arrival time, it is not accumulated into the future observations. In order to evaluate the impact of delayed observations, we consider in our simulations a probability of $p_i = 0.6\ \forall i$ in Eq. (3.2.4), if not stated otherwise. This means that an acknowledgment is delayed until the next epochs, with probability: $1 - p_i = 0.4$. We set the buffer size, b_i, for all queues to 10 jobs. This value for b_i was chosen arbitrarily, and any other value can be used. For the UCT part of the B.2 algorithm, we used the exploration constant $c = R_{high} - R_{low}$, where $R_{high}(R_{low})$ is the highest(lowest) reward that can be achieved. And the depth of the tree was set to 10. Further, if not explicitly given, we use the *combined reward function* given in Eq. (3.2.16) with $\kappa = 100$, since we aim to avoid job drops in the system. We consider the system depicted in Fig. 3.1 for both cases of heterogeneous and homogeneous servers. In particular, we show numerical results comparing POL to different variants of load balancing strategies (with and without full system information) with respect to:

- the log complementary cumulative distribution function (CCDF) of the empirical job response time (measured from the time a job enters the queue until it completes service and leaves, lower is better). *This is done only for the jobs which are not dropped*,
- the empirical distribution of the job drop rate (measured over all simulation runs where for each run we track the number of jobs dropped out of all jobs received per run, lower is better),
- and the cumulative reward (higher is better).

The evaluation box plots are based on $T_m = 100$ independent runs of $T_e = 5 \cdot 10^3$ jobs with whiskers at $[0.5, 0.95]$ percentiles. The plotted results are an average of these 100 Monte Carlo simulations. For every independent run, a new set of inter-arrival and service times are sampled based on the chosen distributions. These sampled times are then used by all load balancing policies in that run in order to do variance reduction, according to the Common Random Numbers (CRN) technique [110].

The chosen inter-arrival and service time distributions are mentioned with the figures, with unit of measurement req/sec. The offered load ratio ($\eta := \lambda / \sum_i \mu_i$) is used to describe the ratio between arrival rate and the combined service rate. The higher the value of η, the higher the job load is on the system, for details on this see Section 2.1. We first mention the different type of load-balancers we have compared to our proposed POL.

3.4.1 *Overview of compared Load-Balancers*

The compared load-balancers can be divided into the following two categories:

Full information (FI) strategies

These strategies have access to the exact buffer length of queues at the time of each job arrival, and also know the arrival rate and the service rates of the servers.

- JSQ-FI: Join-the-Shortest-Queue assigns the incoming job to the server with the smallest buffer filling.
- DJSQ-FI: Join the shortest out of d randomly selected queues. If not stated otherwise, $d = 2$ has been used for our experiments.
- SED-FI: Shortest-Expected-Delay assigns the incoming job to the server with the minimum fraction of the current buffer filling divided by the average service rate.

Limited information (LI) strategies

These strategies, similar to POL, only have access to the randomly delayed acknowledgments.

- JMO: Join-the-Most-Observations maps an incoming job to the server that has generated the most observations, i.e., received acknowledgments, in the last inter-arrival epoch. This might lead to servers becoming and remaining idle (stale).
- JMO-E (with Exploration): with probability 0.2 randomly chooses an idle server and with probability 0.8 performs JMO.

For all strategies, ties are broken randomly.

3.4.2 *Numerical Results*

We first consider a system with $M = 2$ heterogeneous servers with exponentially distributed service times with rates $\mu_1 = 4$ and $\mu_2 = 2$. The inter-arrival times are also exponentially distributed with rate $\lambda = 5$. Fig. 3.4 shows the numerical comparison of POL with other load-balancers. Observe that, even though POL does not have access to the exact state of parallel systems and also the acknowledgments from the different systems are randomly delayed, it still achieves comparable results to *full information strategies*, while it outperforms the other *limited information strategies*. This is because in the other *limited information strategies*, the initially chosen queues play a key role. Since the queue to which more jobs are sent, will also give back more observations (acknowledgments), and JMO and JMO-E will keep sending to those queues, resulting in job drops. The overlap in response time indicates the similarity of the policies of the strategies, especially for high offered load when most of the finite queues will be full. The heatmap in, Fig. 3.4(d), represents the policy of POL at each buffer filling state. It can be seen that higher priority is given to the faster server, μ_1, having buffer filling b_1. The light (dark) regions in the heatmap corresponds to the state where jobs are allocated to server 1 (2). *This heatmap shows that for every possible state of the two queues $s = \{b_1, b_2\}$, even with limited and delayed information, POL is able to allocate more jobs to the queue with lower filling or*

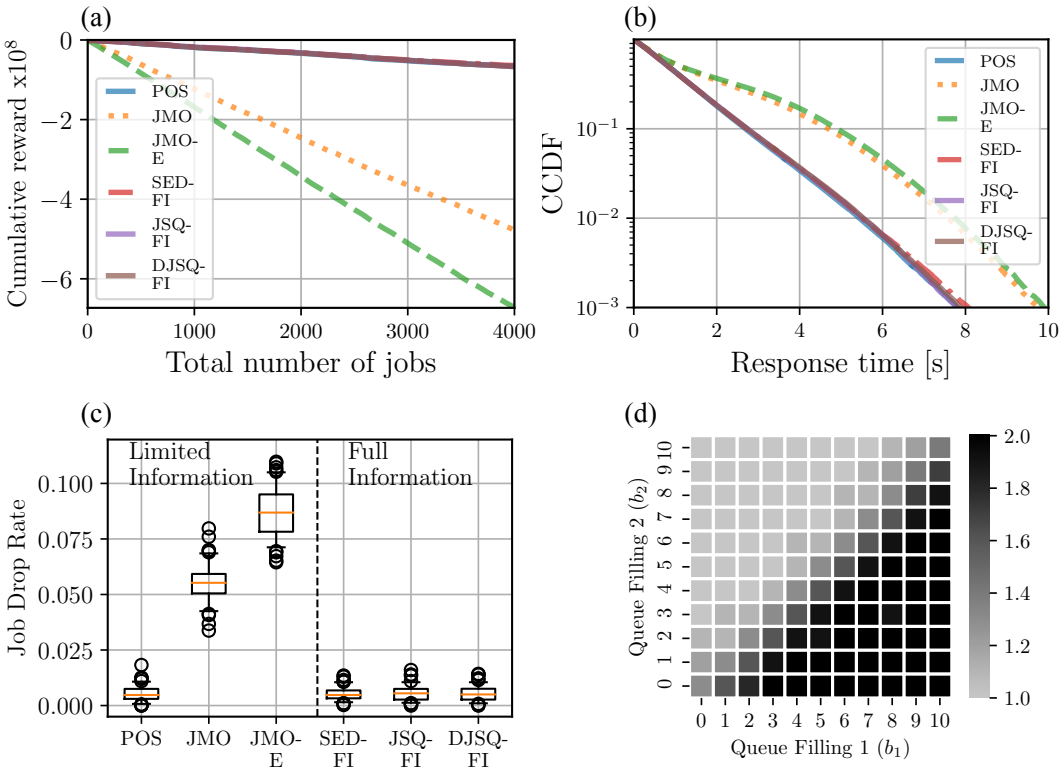

FIGURE 3.4: $M = 2$ heterogeneous servers with exponential service rates $\mu_1 = 4$, $\mu_2 = 2$. Job inter arrival times are exponentially distributed with rate $\lambda = 5$. POL outperforms the algorithms with limited information, JMO and JMO-E, in terms of (a) cumulative reward (higher is better), (b) response time (lower is better), and (c) job drops (lower is better). Although POL *only observes* the randomly delayed job acknowledgments, while SED-FI, DJSQ-FI and JSQ-FI know the **exact buffer fillings and service times / rates**, POL still has comparable performance. The heatmap (d) shows the allocation preference of POL based on how filled the queue is $b_1(b_2)$, i.e., the label bar $1.0(2.0)$ denotes the allocation to queue $1(2)$, respectively.

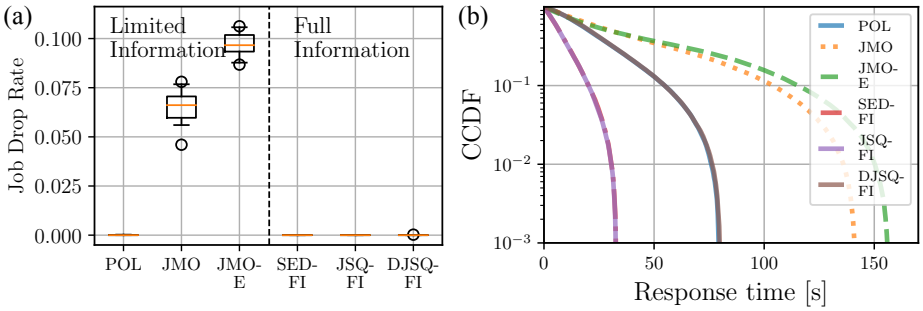

FIGURE 3.5: $M = 50$ heterogeneous servers with job inter-arrival times and service times described by an exponential distribution, with the offered load $(\eta \approx 1)$. POL outperforms the other algorithms with limited information, JMO and JMO-E, in terms of both (a) job drops and (b) response time. And has comparable performance to the full information strategies, SED-FI, DJSQ-FI and JSQ-FI.

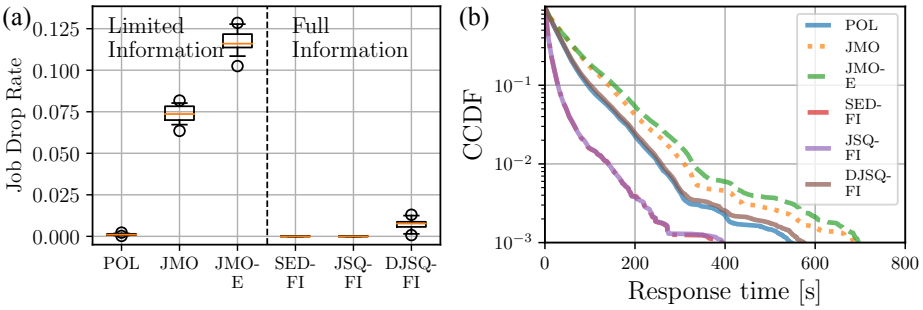

FIGURE 3.6: $M = 50$ heterogeneous servers with gamma arrivals and Pareto service times, with offered load $(\eta \approx 1)$. The effect of the heavy tailed Pareto distribution can be seen in the response plot (b). In terms of job drops, POL outperforms limited information (LI) strategies as well as FI strategy, DJSQ-FI.

faster servers, similar to JSQ and SED. Hence, it is able to perform almost as good as the FI strategies.

Fig. 3.5 shows the performance of POL for $M = 50$ heterogeneous servers. The service and arrivals rates of this setup are kept that the offered load is $(\eta \approx 1)$. This experiment shows that our load-balancer POL is scalable to perform well for large number of queues. Next, we remain with the case of $M = 50$ heterogeneous servers, however with inter-arrival times that are gamma distributed while the service times follow a heavy tailed Pareto distribution, with the offered load $(\eta \approx 1)$.

Fig. 3.6 shows that here too, POL is able to outperform both the LI strategies and the FI strategies, DJSQ-FI. The other two FI strategies have better performance because they always have timely and exact information of the queues, which is unrealistic. Note that as we consider heavy-tailed distributions in this example, the prediction of job acknowledgments by POL suffers, because of reasons discussed at the end of subsection 3.2.3.

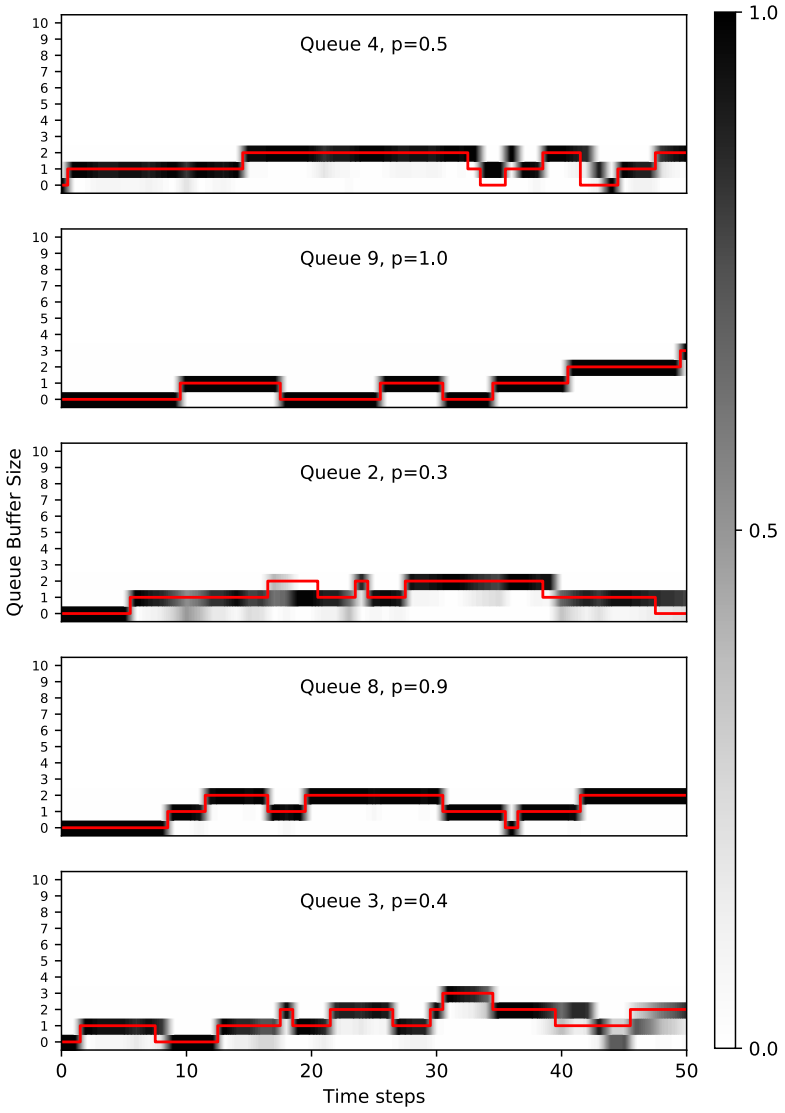

FIGURE 3.7: $M = 10$ homogeneous servers from which 5 were chosen at random to see their sample run for 50 time steps. The delay p for the acknowledgments from each of the server was also allocated randomly, ranging from 0.1 to 1.0. In each subplot is given the queue number and its delay, p. The solid red line trajectory is the true sample path for each queue (*not known to* POL), while the shaded region around it is the belief probability that POL has for each state at each time step. Exponentially distributed inter-arrival and service times were used, with the offered load $\eta \approx 1$. POL, with the help of SIR particle filter, is able to track the real state of the system, as long as the delay is not too high, which is why it is able to perform as good as FI strategies.

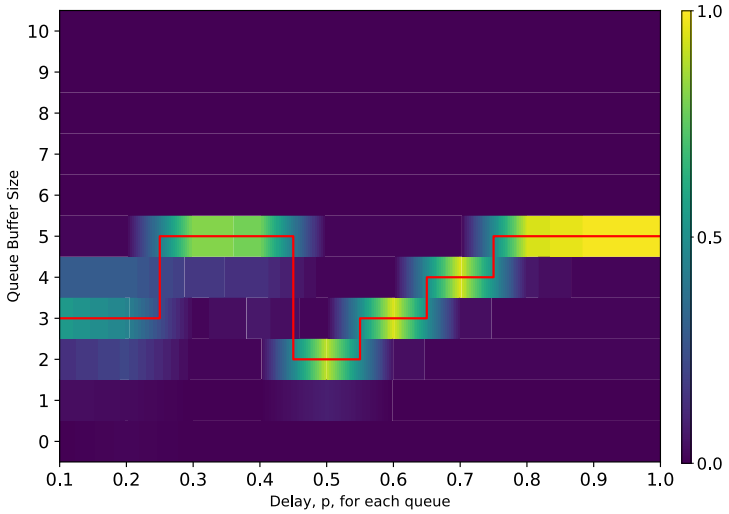

FIGURE 3.8: Visualization of belief state, based on the particles, of $M = 10$ queues after 1000 epochs. The x-axis gives the delay probability of each queue, ranging from $p = 0.1$ (worse delay) to $p = 1.0$ (no delay). The solid red line trajectory is the true state of each queue (not known to POL), while the shaded region around it is the belief probability that POL has for each queues' state after 1000 time steps. It can be seen that as the acknowledgments become less delayed (going from $p = 0.1$ to $p = 1.0$)), the belief of POL gets closer to the true state of the queue. Exponentially distributed inter-arrival and service times were used, with the offered load, $\eta \approx 1$.

3.4.3 *Sensitivity Analysis*

Next, we discuss the impact of the limited observations on POL under different acknowledgment delays, p_i. Recall, that POL is not able to observe the buffer fillings, but rather receives the randomly delayed acknowledgments of the served jobs. These delayed acknowledgments are used by the SIR filter of POL to keep its belief of the state of the environment updated.

Fig. 3.7 depicts sample runs showing the actual evolution of the job queue states (red solid line) and the belief (in shaded region) that POL has on each queue state at each time step, under different acknowledgment delays. Observe that increasing delays (i.e., lower p_i) increases the uncertainty in belief of each state.

However, POL is still able to track the system state for different delays for each server, which shows the efficiency of the SIR particle filter and also *justifies the performance of POL to be as good as FI strategies*. Having different delays in acknowledgments from each server reflects a distributed system, where network conditions may be different for each server and may lead to different delays in acknowledgments from different servers. Fig. 3.8 visualizes the belief of POL on the state of each queue after 1000 epochs, for different delays in acknowledgments in each queue.

In Fig. 3.9 we analyze the performance of POL under varying offered loads ranging from $\eta = 0.2$ to $\eta = 1.2$. It can be seen that POL has almost no job losses up to a load of $\eta = 1$. Note the qualitative change of the response time distribution as the offered load reaches $\eta = 1$ and beyond. For lower offered load, the response time distribution resembles an exponentially tailed distribution which changes with $\eta = 1$ and beyond. In Fig. 3.10 we

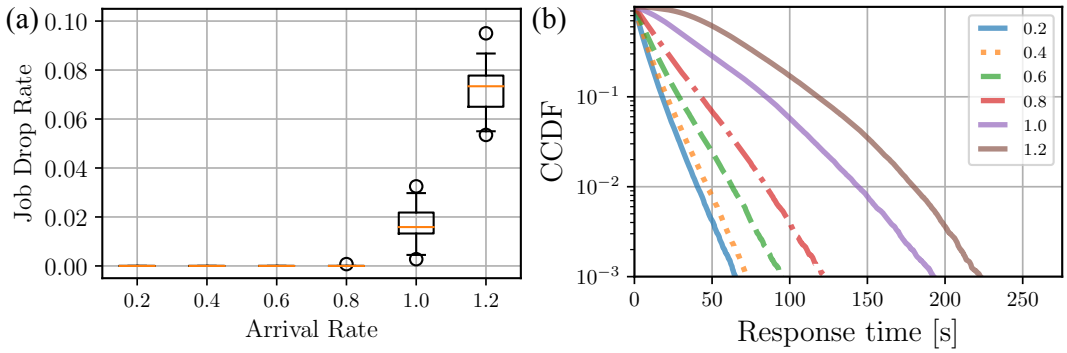

FIGURE 3.9: Varying offered load for a setup as in Fig. 3.5. As long as the offered load is $\eta < 1$, POL has no job losses. For loads, $\eta \geq 1$ we observe a load dependent exponential tail of the response time distribution.

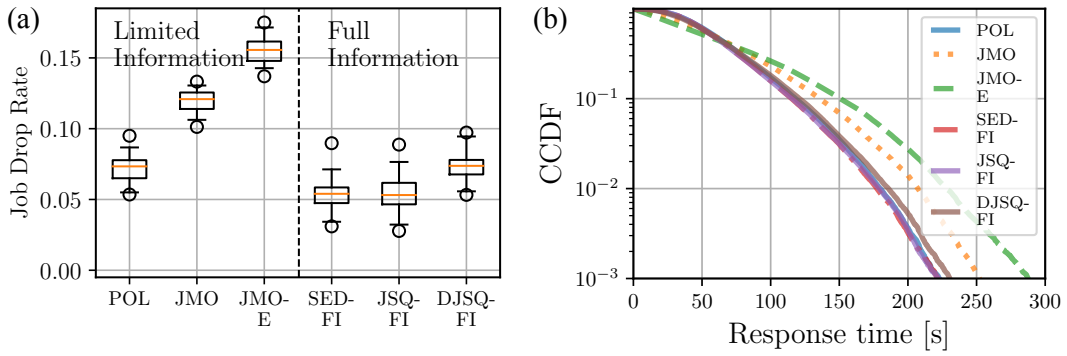

FIGURE 3.10: For the setup from Fig. 3.9 with an offered load $\eta = 1.2$: POL shows a comparable performance to full information (FI) strategies, while outperforming other limited information (LI) ones.

show the performance comparison of different LI and FI strategies for the high offered load case of $\eta = 1.2$. This is done to show that even though POL has high job drops and response times, it outperforms the LI strategies and has comparable performance to the FI strategy, especially DJSQ-FI.

For the sake of completeness, Fig. 3.11 compares the performance of POL using different reward functions from subsection 3.2.5, while keeping all the other parameters the same.

Time Analysis of POL: POL consists of two main components: (i) Tree simulator for action evaluation and (ii) SIR particle filter for belief update. Both steps need to be done at every decision epoch, i.e. on every job arrival. Since we assume no queue at the load-balancer, POL needs to allocate the incoming job to one of the queues, before the next arrival. Note that MCTS is a very successful online algorithm. Hence, POL first takes a portion of the time between arrivals to simulate the tree and take a decision for the current arrival. Then takes that action on the real environment and based on the received delayed acknowledgment performs the belief update using the SIR particle filter, until the next job arrives. As can be seen from the simulation results, POL is scalable in terms of the number of servers and is able to handle high offered loads, η. Note that the code used to run POL here in the system simulation is the same that would be used in a deployment scenario. Next, we investigate

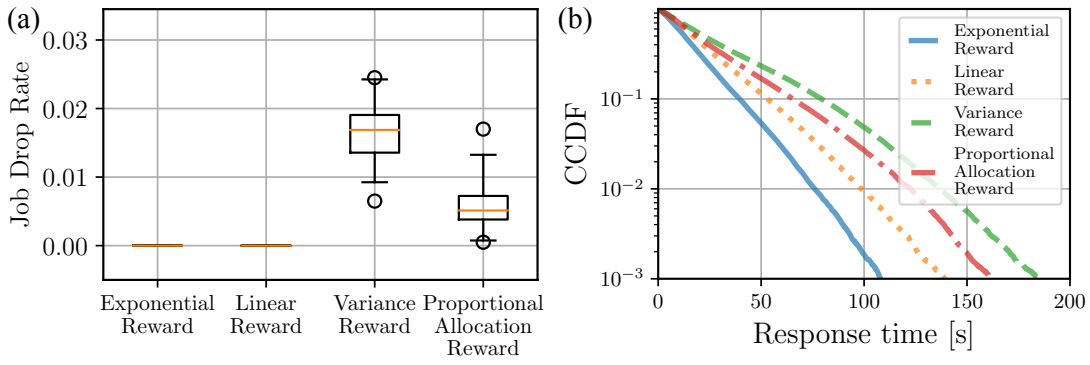

FIGURE 3.11: Comparison of different reward functions for $M = 50$ homogeneous servers, with exponentially distributed inter-arrival and service times and offered load $\eta \approx 1$.

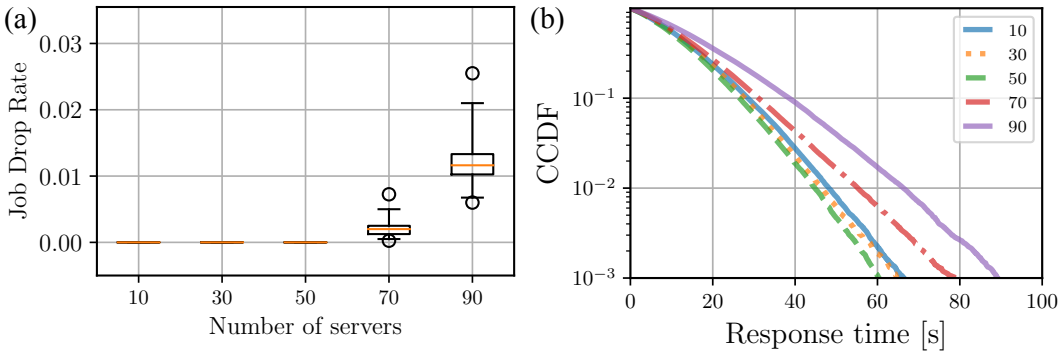

FIGURE 3.12: Homogeneous servers with exponentially distributed inter-arrival and service times. The number of servers is increased in intervals of 20 servers, while keeping the offered load always $\eta = 0.99$. With a higher number of servers, less time is available for POL to simulate the tree and do belief update, resulting in a slight deterioration in the performance.

the impact on the inter-arrival time between jobs on the load-balancer performance, as the inter-arrival time needs to be sufficient for POL to perform the above two steps at every job arrival.

In Fig. 3.12, the number of homogeneous servers M was increased from 10 to 90, while keeping the offered load fixed, i.e., $\eta = 0.99$. The average service rate μ_i of the servers is kept fixed in all experiments, i.e. the increase in M results in an increase in the sum of service rates, $\sum_i \mu_i$. Hence, to keep the offered load fixed with scale the job arrival rate accordingly. Firstly, this experiment demonstrates the scalability of POL in terms of number of servers. Although the state space of the system increases with the number of servers, hence, queues, POL manages to deal well with the increased state space. In POL we use MCTS adapted from POMCP [98], so instead of considering the entire state space we have a fixed set of particles to represent the state based on our belief of the state, thus tapering the curse of dimensionality and space complexity. As the inter-arrival time is the decision epoch, we observe that the time given to POL to simulate the tree and do the belief update reduces, the effect of which can be seen as the slight decrease in performance as M increases. It can be seen in Fig. 3.13, that for $M = 90$, lowering the load again, i.e. giving POL more time to decide, improves the performance of the system. This shows the trade-off

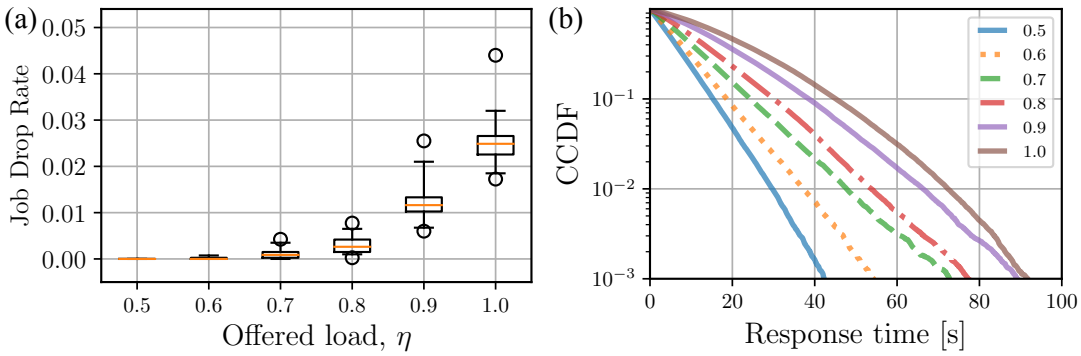

FIGURE 3.13: Performance of POL for $M = 90$ homogeneous servers for different loads.

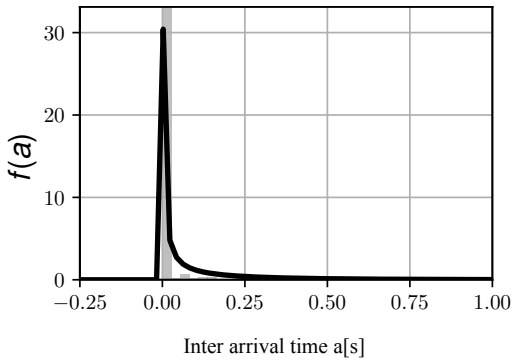

FIGURE 3.14: The density estimate of the job inter-arrival times indicates that it is exponentially distributed.

between the load balancing performance, e.g. in terms of the response time and drop rate vs. the load, which directly impacts the time provided to POL to make a decision.

We believe this to be the current limitation of POL, however step (i) can be further optimized using MCTS parallelization [111]. Note that the computational resources used also have a strong impact on the performance of POL. Here, we use a dedicated machine with an Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz for all our experiments.

In the next section, we discuss the scenario when some system parameters are not known and need to be inferred from the available data.

3.4.4 *Experiments with trace data*

For the results of this section, we make use of *Labeled Network Traffic Flow* data, provided by Kaggle in 2019 [112]. We used the frameworks given in Section 3.2.4 to infer the underlying distributions of the inter-arrival and service times provided in this data set. The inferred distribution based on data of the inter-arrival times of the chosen source is given in Fig. 3.14, it can be seen to follow an exponential distribution with high arrival rate. We then selected $M = 20$ heterogeneous servers from the available data such that they all followed the Gamma Mixture distribution. Gamma Mixture was selected to show the performance of POL with yet another type of service time distribution. The empirical distribution as a histogram as well as the posterior mean estimate for some of these selected

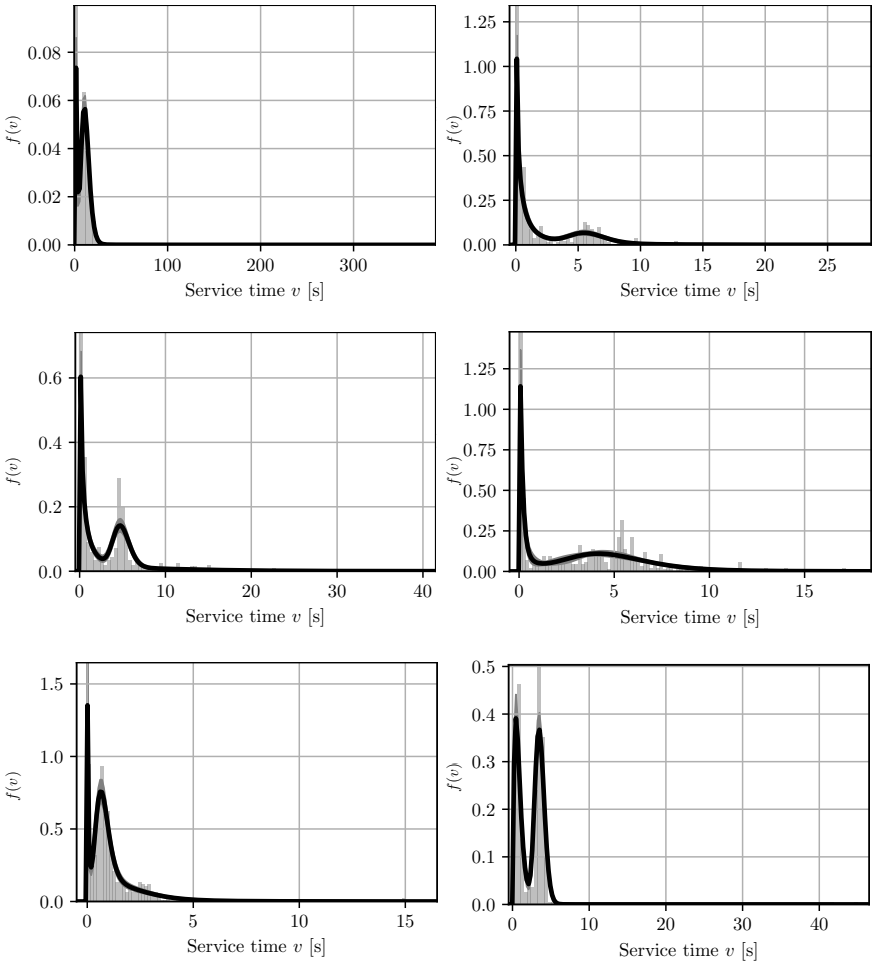

FIGURE 3.15: The density estimate of the job service times using a Gamma Mixture Likelihood model indicates that the servers are heterogeneous with high service times.

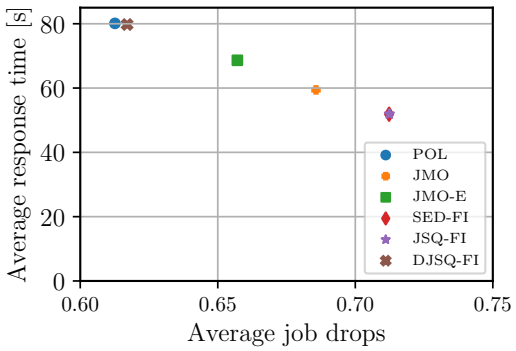

FIGURE 3.16: Trade-off between response time and job drop: In terms of job drops, POL has comparable performance to DJSQ-FI, while it outperforms the other FI and all LI strategies. The additionally allocated jobs increase the overall average response time.

servers is given in Fig. 3.15. The hyperparameters used are: $c = 3$, $\epsilon_i^1, \epsilon_i^2 = 1$, $m = 1$. POL makes use of the samples generated using the posterior predictions.

We assume that the servers have a finite buffer of size $B = 10$ (arbitrarily chosen) and a delay in acknowledgments of $p = 0.6$. Fig. 3.16 shows that even with limited information, POL has the lowest average job drops. However, due to the limited information available to POL and the reward function it is using, its average response time is as high as the full information strategy DJSQ-FI. POL is able to allocate more jobs to the servers (due to fewer drops), which can come at a cost of higher response time for some jobs, which will be allocated to the slower servers. Note that the reward function we used, (3.2.16), focuses on avoiding job drops and not on minimizing the response time.

3.5 SUMMARY

In this chapter, we analyzed online algorithms for mapping incoming jobs to parallel and heterogeneous processing systems under partial observability constraints. This partial observability is rooted in the assumption that the entity controlling this mapping, denoted load-balancer, takes decisions only based on *randomly delayed feedback* of the parallel systems. Unlike classical models that assume full knowledge of the parallel systems, e.g., knowing the queue lengths (JSQ) or additionally the job service times, SED this model is particularly suited for large distributed processing systems that only provide an acknowledgment-based feedback.

In addition to presenting a partially observable (semi-)Markov decision process model that captures the load balancing decisions in this parallel queuing system under delayed acknowledgments, we provide a Partial Observable Load-Balancer (POL) - to find near-optimal solutions online. A particular strength of POL is that it allows to *define the objectives of the system and lets it find the appropriate load balancing policy instead of manually defining a fixed one*. It can also be used for any kind of inter-arrival and service time distributions and is scalable to a large number of queues. We numerically show that the POL load balancing policies obtained under partial observability are comparable to fixed policies such as JSQ, JSQ(d), and SED which have full information. This is the case even though POL receives less, and in addition randomly delayed, informative feedback.

4

MULTI-AGENT CONTROL FOR LARGE QUEUING SYSTEMS

In this chapter, we extend the queuing system presented in Chapter 3 to having multiple decision-making agents (referred to as load-balancers), based on the published work [2]. This is a more realistic system since in a network you will have multiple decision makers trying to use the same limited resources. Hence, the multi-agent load balancing system considered in this chapter is crucial to effectively utilize distributed systems such as those present in data centers and cloud services. Our discrete-time system model now also incorporates an arbitrary synchronization delay under which the queue state information is synchronously broadcasted and updated at all load-balancers. This delay includes the network delay and can be assumed to be the maximum time till all the agents have received their acknowledgments, where each acknowledgment contains the desired queue state information. In order to obtain a tractable solution, we model this system as a mean-field control problem and apply policy gradient reinforcement learning algorithms to find an optimal load balancing solution. We also provide theoretical performance guarantees for our methodology in large systems, as well as a comparison to the state-of-the-art power-of-d variant of the join-the-shortest-queue (JSQ) and other policies in the presence of synchronization delays.

A natural extension to this model is to consider a localized queuing system and is presented in Chapter 5.

Related Work

Load balancing in large queuing systems has yielded many successful distributed algorithms such as Join-the-Shortest-Queue (JSQ), Shortest-Expected-Delay (SED) [20, 22, 85] and many others, see also [84] for a recent review. However, JSQ and SED have been designed for asynchronous systems with a central load-balancer (agent) assigning jobs (packets) to parallel servers (queues) under the assumption that the load-balancer can obtain instantaneous, accurate and synchronized information of the queue lengths at all times. In practice, both instant information and centralized decision-making are not realistic, especially if the number of queues $M \gg 1$ is large. In this chapter, we consider a multi-agent system of N load-balancers and M servers (queues) with $N \gg M \gg 1$ and communication delay.

To remedy this scalability issue, the power-of-d versions JSQ(d) and SED(d) of JSQ and SED [113] let the load-balancer sample only $d \leq M$ out of M servers randomly and then allocate the job to the sampled server with the shortest expected processing time. However, JSQ(d) and SED(d) nonetheless assume instant and accurate information of the state of those d servers, which remains unrealistic due to both the distributed nature of the system and computational overheads introducing latency. The problem is only exacerbated in a multi-agent scenario where all agents access simultaneously. Hence, to model a more realistic system, it is of importance to take communication delays Δt into account. In [114], it was shown that JSQ fails when $\Delta t > 0$ mainly due to a phenomenon known as 'herd behavior': Multiple agents assigning jobs at the same time would consider the same subset of servers with few jobs, and thus all agents will end up assigning to the same servers. This eventually leads to higher response times and, in the case of finite queues, job drops. Though JSQ(d) ameliorates this issue somewhat since it is highly unlikely for small d and large M that many agents will randomly choose the same servers, the technique nonetheless remains suboptimal under delayed information. Indeed, as $\Delta t \to \infty$, a completely random allocation to one of the servers becomes optimal [113]. However, when the delay Δt lies between 0 and ∞, the optimal policy must lie in-between, which will be the main focus of this chapter.

In order to scale to a great number of load-balancers and servers, we will apply mean-field theory, analogous to fluid limits $M \to \infty$, that is used to tractably model and assess systems with many queues. Fluid limits were used to study the performance of scheduling algorithms like JSQ and JSQ(d) in terms of sojourn time and average queue length [113, 115, 116]. However, models including delayed information still remain an open problem [117], in particular in the presence of many agents. One work with similar system model and synchronization delays is given in [118], though they instead consider finitely many servers with infinite buffer sizes where the multiple agents use their local, asynchronous estimates of queue lengths to perform scheduling. This idea of using local agent memory has also been proposed in [119, 120], however only for a single -agent.

More generally, the same tractability issue for large systems has led to the increasing popularity of general (competitive) mean-field games (MFG) [50, 51, 56] and their cooperative counterpart of mean-field control (MFC) [8, 57–60], wherein a system with large numbers of interchangeable and indistinguishable agents is converted into a system where one

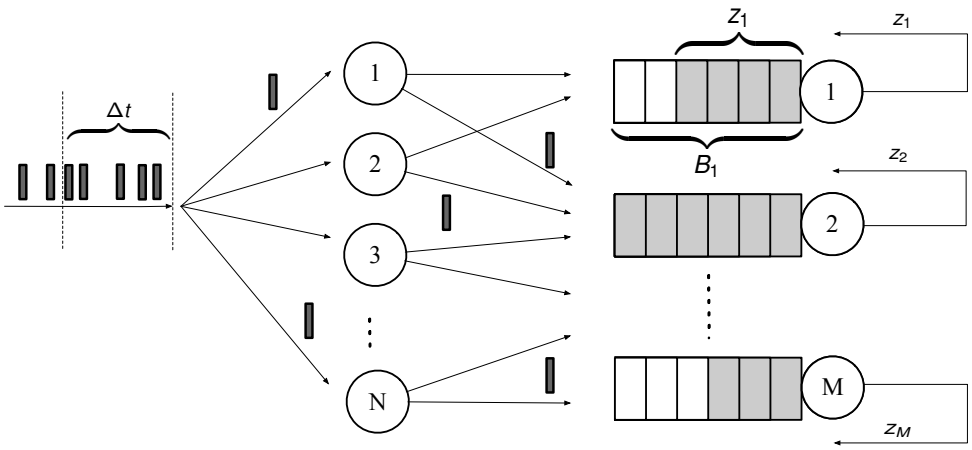

FIGURE 4.1: Our system model consists of N load-balancers and M parallel servers. Jobs arriving in a certain time interval Δt are assigned to the agent, which consequently assigns them to one of a few sampled servers based on some policy. Arrows from each agent indicate the $d = 2$ servers randomly sampled by each agent at the current epoch.

representative agent is interacting with the distribution (mean-field) of other agents. Here, there has been a recent focus on learning-based solution algorithms for MFGs [121–124] and MFC [53, 70, 125]. We will similarly apply the enlarged state-action space technique for MFC (see e.g. [70]), its associated dynamic programming principle as well as reinforcement learning in order to find optimal load balancing policies for an otherwise intractably large system. While RL [26], so-far has found great success e.g. in games [126, 127], robotics [128] or communication and queuing networks [124, 129], in the case of multiple agents, there still remain many challenges in MARL such as intractability for large numbers of agents [48]. RL itself has long since been used in numerous works – though not in the context of mean-field control – to find an optimal load balancing policy. For examples, see [20, 92, 130, 131] and references therein. The combination with mean-field control allows for a tractable solution of very large load balancing systems and shall be the subject of our studies. We will similarly formulate a synchronous system model with delay by assuming $N \gg M \to \infty$, which will allow us to apply reinforcement learning to the otherwise difficult to solve optimal load balancing problem. Although our model shares similarities in concept to MFC, it does not immediately fit into the framework of conventional MFC, as we not only derive the discrete-time mean-field model starting from an underlying continuous-time dynamic, but at the same time take a double limit of infinitely many queues and agents. While, existing MFC frameworks typically focus only on the limit of infinitely many agents without external dynamics of non-agent-bound (queue) states.

4.1 QUEUING SYSTEM MODEL WITH SYNCHRONIZATION DELAY

We consider N load-balancers and M servers, where each server has its own queue with limited buffer capacity. An overview of the considered load balancing system is given in Fig. 4.1. Jobs arrive randomly according to a Markov modulated Poisson process – modeling e.g. changing load factors throughout a day – with rate $\lambda(t)M$ and are divided uniformly among the load-balancers, which will allocate the jobs to servers for processing.

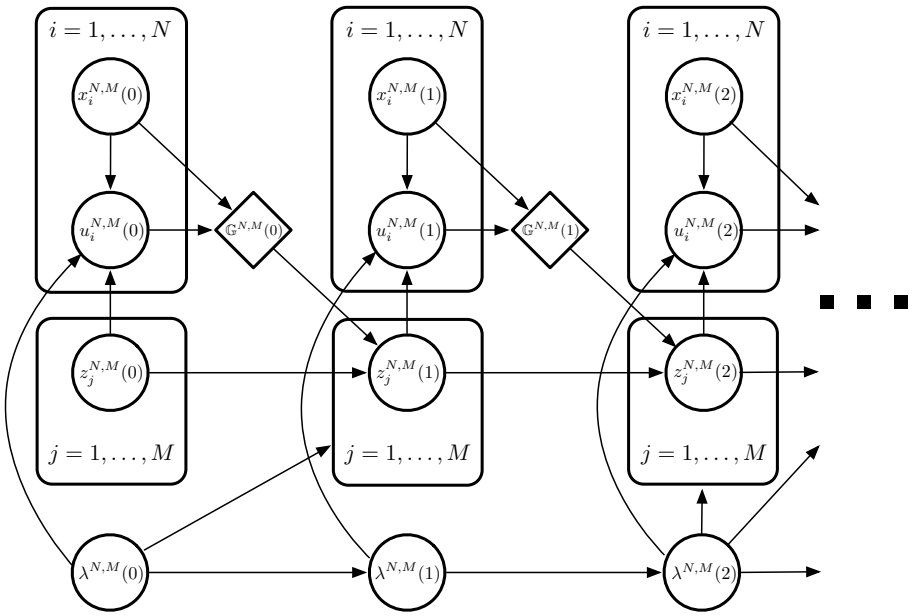

FIGURE 4.2: Probabilistic graphical model of our multi-agent multi-server scheduling system using plate notation, where diamonds and circles represent deterministic and stochastic nodes respectively [132]. Conditional on the random variables at time t, each agent's states as well as each queue's states at time $t + 1$ are i.i.d. random variables.

In accordance with the power-of-d technique, agents shall randomly select d out of M queues and – according to some policy to be optimized – send their jobs to a selection of these d queues, where $d \ll M$. On the queuing side of our system model, we have M parallel and homogeneous servers in the system with service rates μ. The queues are finite with a maximum buffer capacity B and the jobs in the queues are served in a first-in-first-out (FIFO) manner. Each server sends back its queue filling status, which is then used by the load-balancers to make their decision for the next incoming jobs. The number of jobs that are currently in each queue together makes up the state of the environment. Our goal is to *minimize overall job drops* under decentralized decision-making by each agent, e.g. like in edge computing scenarios.

We will assume that our system operates synchronously and broadcasts updates of sampled queue states to dispatchers only once every fixed time interval. Thus, in the following we will model our system at discrete decision epochs $\{0, \Delta t, 2 \cdot \Delta t, \ldots\}$ for some synchronization delay $\Delta t > 0$, after each of which the load-balancers will sample d new queues and keep this selection of d queues for the entire duration of that decision epoch. Not only will this allow us to incorporate communication delays, but it will also lead to significantly less sampling of server states by the agents, as each agent is only required to sample d servers in every decision epoch. Another advantage of this approach is that the resulting discretized Markov decision process will allow us to apply powerful and well-established reinforcement learning algorithms, which to this date have been extensively developed for discrete-time models.

4.2 FINITE-AGENT FINITE-QUEUE MODEL

Formally, the N-agent M-queue system could be considered a multi-agent Markov Decision Process (MMDP) for $N, M \in \mathbb{N}$, i.e. the cooperative and fully observable case, definition given in Section 2.2.1. See e.g. [37] for a review of possible multi-agent problem formulations. In principle, one could even consider competitive or partially observed cases. However, the resulting limiting mean-field systems will be significantly more complex. Instead, we will in the following consider a decentralized control setting where agents, due to the symmetry of our model, shall act depending on the current distribution of queue states.

Define $\mathcal{Z} \coloneqq \{0, \dots, B\}$ as the finite queue state space, i.e. each server can contain at most B jobs in its queue. The agent state space shall be denoted as $\mathcal{X} \coloneqq \{1, \dots, M\}^d$, i.e. a selection of d random queues. Although we could disallow repeated queue selections, it will make no difference in sufficiently large systems and add unnecessary notational complexity. Finally, each agent can choose as an action its choice of one of d randomly sampled accessible queues, i.e. the action space is defined as the d possible queue choices $\mathcal{U} \coloneqq \{1, \dots, d\}$. At any decision epoch $t = 0, 1, \dots$, the states and actions of agents $i = 1, \dots, N$, are random variables denoted by $x_i^{N,M}(t) \equiv (x_{i,1}^{N,M}(t), \dots, x_{i,d}^{N,M}(t)) \in \mathcal{X}$ and $u_i^{N,M}(t) \in \mathcal{U}$, and similarly the state of each queue $j = 1, \dots, M$ is denoted by $z_j^{N,M}(t) \in \mathcal{Z}$ with $z_j^{N,M}(0) \sim \nu(0) \in \mathcal{P}(\mathcal{Z})$ from some initial distribution $\nu(0)$. Additionally, $\lambda^{N,M}(t) > 0$ – the arrival rate parameter – will be modulated as an independent discrete-time Markov chain (DTMC) with state space Λ, i.e.

$$\lambda^{N,M}(t+1) \sim \mathcal{T}_\lambda(\lambda^{N,M}(t)) \tag{4.2.1}$$

for some arbitrary transition kernel $\mathcal{T}_\lambda$, see [15, 17] for details on DTMC.

Due to the symmetry of the problem, for sufficiently many agents, the information about each specific queue's state becomes irrelevant to the problem. Thus, we assume some common, shared policy of the form $\pi_t \colon \mathcal{P}(\mathcal{Z}) \times \mathcal{Z}^d \times \Lambda \to \mathcal{P}(\mathcal{U})$ for all agents, acting on the current $\mathcal{P}(\mathcal{Z})$-valued random empirical queue state distribution

$$\mathbb{H}^{N,M}(t) \coloneqq \frac{1}{M} \sum_{j=1}^{M} \delta_{z_j^{N,M}(t)} \tag{4.2.2}$$

with Dirac measure δ, the sampled queue states, and the current arrival rate. In practice, we may also drop dependence on the current arrival rate and empirical distribution, or estimate e.g. the empirical queue state distribution by sampling a subset of random queues, though both will complicate the theoretical analysis of the limiting MFC problem, as it would not be possible to formulate the limiting system as a standard, fully-observed Markov decision process. For details on MFC see Section 2.3.1.

The dynamics for each agent i are thus given by

$$x_i^{N,M}(t) \sim \otimes_{k=1}^{d} \operatorname{Unif}(\{1, \dots, M\}), \tag{4.2.3}$$

$$u_i^{N,M}(t) \sim \pi\left(t, \mathbb{H}^{N,M}(t), (z_{x_{t,1}^i}^{N,M}(t), \dots, z_{x_{t,d}^i}^{N,M}(t)), \lambda^{N,M}(t)\right), \tag{4.2.4}$$

i.e. at each decision epoch, the agents decide to which of their d randomly sampled, accessible queues they decide to send their jobs to. For simplicity of exposition, this choice of destination is deterministic, though in our experiments we shall allow randomization for each packet. As a result, starting with $z_j^{N,M}(0) \sim \nu(0) \in \mathcal{P}(\mathcal{Z})$ for each queue j and some initial queue state distribution $\nu(0)$, for any queue j, the next queue state $z_j^{N,M}(t+1)$ is obtained from the previous state $z_j^{N,M}(t)$ by simulating a $\mathcal{Z}$-valued continuous-time Markov chain (CTMC) for Δt time units, beginning with $z_j^{N,M}(t)$ and decrementing or incrementing by 1 at departure rate $\mu > 0$ and arrival rate

$$\lambda_j^{N,M}(t) = M\lambda^{N,M}(t) \cdot \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{d} \mathbb{1}_{x_{i,k}^{N,M}(t)=j} \mathbb{1}_{u_i^{N,M}(t)=k} \tag{4.2.5}$$

respectively, ignoring jumps above B or below 0. Any arrivals beyond B are counted in the average number of dropped packets

$$D^{N,M}(t) = \frac{1}{M} \sum_{j=1}^{M} D_j^{N,M}(t) \tag{4.2.6}$$

per queue j during each decision epoch t, which will constitute our objective through the discounted infinite-horizon objective

$$J^{N,M}(\pi) = \mathbb{E}\left[-\sum_{t=0}^{\infty} \gamma(t) D^{N,M}(t) \right] \tag{4.2.7}$$

to be maximized with discount factor $\gamma \in (0,1)$. See [15, 17] for details on CTMCs.

Note that we can rewrite (4.2.5) as

$$\lambda_j^{N,M}(t) = M\lambda^{N,M}(t) \int_{\mathcal{X}\times\mathcal{U}} \sum_{k=1}^{d} \mathbb{1}_{x_k=j} \mathbb{1}_{u=k} \, \mathbb{G}^{N,M}(t)(\mathrm{d}x, \mathrm{d}u) \tag{4.2.8}$$

with the $\mathcal{P}(\mathcal{X}\times\mathcal{U})$-valued empirical agent state-action distribution

$$\mathbb{G}^{N,M}(t) := \frac{1}{N} \sum_{i=1}^{N} \delta_{x_i^{N,M}(t), u_i^{N,M}(t)}. \tag{4.2.9}$$

The dynamical dependencies can be summarized in a probabilistic graphical model as shown in Fig. 4.2. Intuitively speaking, when $N \gg M \gg 1$, this empirical distribution becomes deterministic, and we need not track each queue state, but only their distribution. Similarly, only the overall distribution of all agent choices will matter, leading to the prospective limiting mean-field model derived in the sequel.

4.3 LIMITING SYSTEM MODEL

We now present the limiting system for the above-mentioned finite multi-agent queuing model. This is done in two steps, first we take the limit on the number of agents and then for the resulting model we assume infinite number of queues.

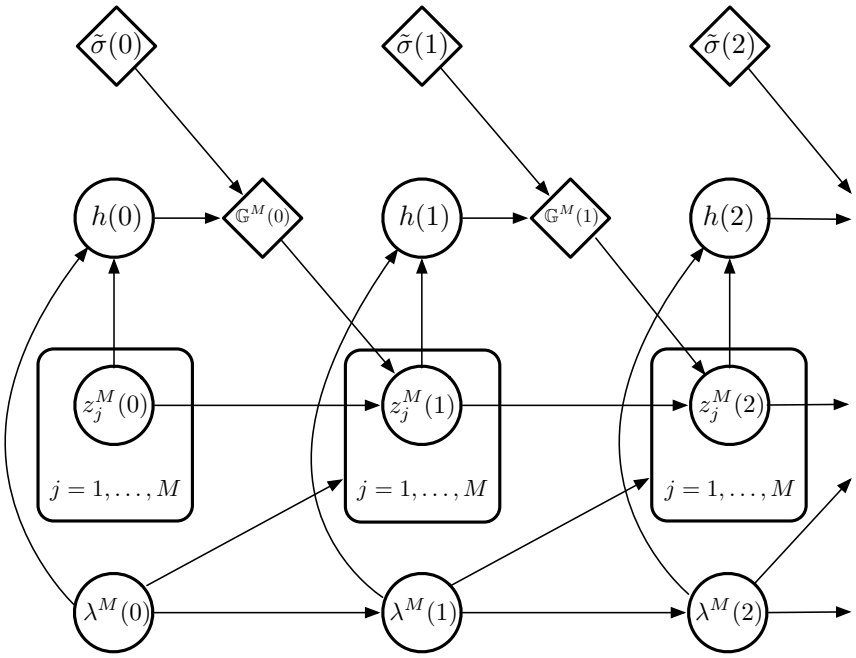

FIGURE 4.3: Probabilistic graphical model of our multi-server scheduling system in the limit of infinitely many agents, using plate notation as in Fig. 4.2. Compared to Fig. 4.2, the states and actions of the infinitely many agents are replaced by their deterministic distribution and conditional law, respectively.

4.3.1 *Infinite-agent finite-queue model*

In the infinite-agent limit where $N \to \infty$, we obtain a limiting control problem with random external states (queue states). Consider the evolution of the $\mathcal{P}(\mathcal{Z})$-valued empirical queue state distribution

$$\mathbb{H}^M(t) \coloneqq \frac{1}{M} \sum_{j=1}^{M} \delta_{z_j^M(t)} \tag{4.3.10}$$

as $N \to \infty$. Conditional on the queue states and arrival rate, $(x_i^M(t), u_i^M(t))_{i=1,\ldots,N}$ are i.i.d. Therefore, it will be sufficient to consider only the statistics of a representative agent. By the law of large numbers, we obtain the deterministic agent state distribution

$$\tilde{\sigma}(t) \coloneqq \otimes_{k=1}^{d} \operatorname{Unif}(\{1, \ldots, M\}) \in \mathcal{P}(\mathcal{X}) \tag{4.3.11}$$

of agents by (4.2.3). The $\mathcal{P}(\mathcal{X} \times \mathcal{U})$-valued agent state-action distribution

$$\mathbb{G}^M(t) \coloneqq \mathcal{G}^M(t, \tilde{\sigma}(t), h(t)) \tag{4.3.12}$$

thus depends on $h(t) \coloneqq \pi(t, \mathbb{H}^M(t), \cdot, \lambda^M(t))$, where we define

$$\mathcal{G}^M(t, \tilde{\sigma}, h)(x, u) \coloneqq \tilde{\sigma}(x) h(u \mid (z_{x_1}^M(t), \ldots, z_{x_d}^M(t))). \tag{4.3.13}$$

We observe that this state-action distribution is sufficient for characterizing system behavior: Conditional on fixed $\lambda^M(t)$ and $\{z_1^M(t), \dots, z_M^M(t)\}$, the arrival rate in (4.2.5) becomes,

$$\lambda_j^M(t) = M\lambda^M(t)\mathbb{E}\left[\sum_{k=1}^{d} \mathbb{1}_{x_{1,k}^M(t)=j}\mathbb{1}_{u_1^M(t)=k}\right] \tag{4.3.14}$$

$$= M\lambda^M(t)\int_{\mathcal{X}\times\mathcal{U}} \sum_{k=1}^{d} \mathbb{1}_{x_k=j}\mathbb{1}_{u=k}\,\mathbb{G}^M(t)(\mathrm{d}x, \mathrm{d}u) \tag{4.3.15}$$

by the law of large numbers, similar to (4.2.8). In other words, the empirical agent state-action distribution $\mathbb{G}^{N,M}(t)$ is replaced by the limiting distribution $\mathbb{G}^M(t)$. See Fig. 4.3 for the corresponding probabilistic graphical model.

4.3.2 *Infinite-agent infinite-queue model*

Finally, we derive the mean-field model in the limit as $M \to \infty$, i.e. formally $N \gg M \gg 1$. The random queue states are now replaced by the queue state distribution denoted by $\nu(t) \in \mathcal{P}(\mathcal{Z})$. Therefore, each agent state $x_i(t) \in \mathcal{X}$ is now also replaced by the anonymous queue state $\bar{z}_i(t) \in \mathcal{Z}^d$ instead of the actual queue index. The queue state distribution deterministically induces the agent state distribution

$$\sigma(t) \coloneqq \otimes_{k=1}^{d}\nu(t) \in \mathcal{P}(\mathcal{Z}^d) \tag{4.3.16}$$

by assigning the d-dimensional product measure $\sigma(t, \bar{z}) = \Pi_{k=1}^{d}\nu(t, \bar{z}_k)$ for any $\bar{z} \equiv (\bar{z}_1, \dots, \bar{z}_d) \in \mathcal{Z}^d$. For any decision rule $h(t) = \pi(t, \nu(t), \cdot, \lambda(t))$, this agent state distribution induces a state-action distribution

$$\mathbb{G}(t) \coloneqq \sigma(t) \otimes h(t) \in \mathcal{P}(\mathcal{Z}^d \times \mathcal{U}). \tag{4.3.17}$$

Now consider the random amount of arriving packets $P \sim \mathrm{Pois}\,(M\lambda(t)\Delta t)$ in a time slot Δt. Since $N \gg M$ implies $N \gg P$, the probability of any single-agent receiving more than one packet is negligible. This implies that almost all packets' destination queues will be i.i.d. random variables. As a result, since packets arrive with rate $M\lambda(t)$ and i.i.d. destinations, for any $z \in \mathcal{Z}$, packets will equivalently arrive with rate $M\lambda'_z(t)$ in queues with state $z \in \mathcal{Z}$ by Poisson thinning [133], where

$$\lambda'_z(t) = \lambda(t)\int_{\mathcal{Z}^d\times\mathcal{U}} \mathbb{1}_{\bar{z}_u=z}\,\mathbb{G}(t)(\mathrm{d}\bar{z}, \mathrm{d}u). \tag{4.3.18}$$

By symmetry, these packets arrive uniformly at random in any arbitrary specific fixed queue in state z. For any specific queue with state z, the probability of assigning such a packet to that queue is therefore $\frac{1}{M\nu_z(t)}$, which results in an equivalent queue packet arrival rate of

$$\lambda_z(t) \coloneqq \frac{M\lambda'_z(t)}{M\nu_z(t)} = \frac{\lambda'_z(t)}{\nu_z(t)}. \tag{4.3.19}$$

The informal derivation until now will be motivated more rigorously in Section 4.5 and numerically in Section 4.7.

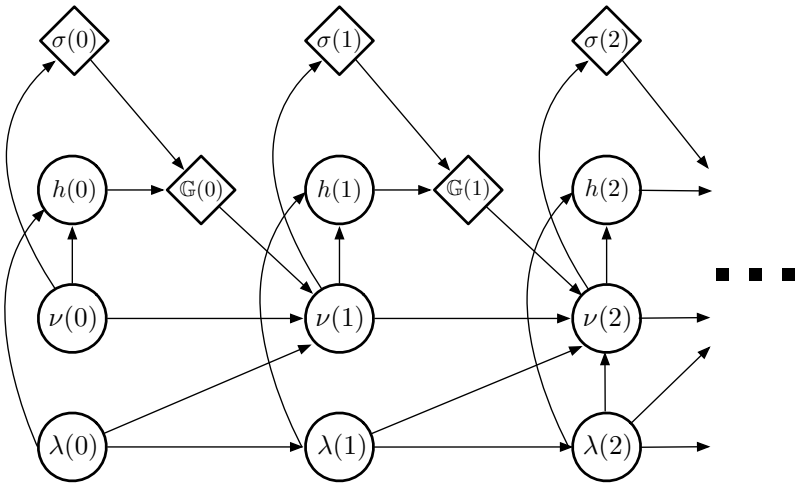

FIGURE 4.4: Probabilistic graphical model of our scheduling system in the limit of infinitely many agents and infinitely many servers. Note how this model can be considered a Markov decision process with states $\nu(t), \lambda(t)$ and actions $h(t)$. Compared to Fig. 4.3, the states of the infinitely many servers are replaced by their distribution. Additionally, the states of agents are now given by the anonymous state of accessible queues.

4.4 EXACT DISCRETIZATION OF THE LIMITING SYSTEM

The final step is to formulate a discrete-time optimal control problem from the delayed, synchronous system that allows for the application of standard optimal control techniques such as reinforcement learning. To discretize the mean-field system exactly at times $\{0, \Delta t, 2 \cdot \Delta t, \ldots\}$, we generate the master equations for the evolution of a single queue's state over time between each of the discretization time points. The procedure is done analogously for the pre-limit systems. Consider a queue in state $z \in \mathcal{Z}$ at the beginning of a decision epoch t. Then, for any $h(t)$, we define a $\mathcal{Z}$-valued CTMC y through $y(0) = z$ and formulate its Kolmogorov forward equations

$$\dot{\mathbf{P}}^z = \mathbf{Q}^z \mathbf{P}^z, \quad \mathbf{P}^z(0) = \mathbf{e}_z \tag{4.4.20}$$

for the vector of queue state transition probabilities $\mathbf{P}^z(\tau) \in [0,1]^{\mathcal{Z}}$ at times $\tau \in [0, \Delta t]$ with

$$P^z_{z'}(\tau) \equiv \mathbb{P}(y(\tau) = z'), \quad \forall z' \in \mathcal{Z} \tag{4.4.21}$$

and the transposed transition rate matrix $\mathbf{Q}^z \coloneqq \mathbf{Q}(\nu(t), z) \in \mathbb{R}^{\mathcal{Z} \times \mathcal{Z}}$ where $\mathbf{Q}(\nu, z)$ is defined by

$$\mathbf{Q}(\nu, z)_{i,i-1} = \lambda(t, \nu, z) \coloneqq \frac{1}{\nu_z(t)} \lambda(t) \int_{\mathcal{Z}^d \times \mathcal{U}} \mathbb{1}_{\bar{z}_u = z} \, (\otimes_{k=1}^d \nu \otimes h(t))(\mathrm{d}\bar{z}, \mathrm{d}u) \tag{4.4.22}$$

in accordance with (4.3.16) - (4.3.19), $\mathbf{Q}(\nu, z)_{i-1,i} = \mu_z$ for $i = 1, \ldots, B$, $\mathbf{Q}(\nu, z)_{i,i} = -\sum_j \mathbf{Q}(\nu, z)_{j,i}$ for $i = 0, \ldots, B$, and zero otherwise. Here, $\mathbf{e}_z$ denotes the z-unit vector.

Therefore, from the fraction $\nu_z(t)$ of queues in state $z \in \mathcal{Z}$ at time t, we will deterministically have the resulting fraction

$$\nu_{z,z'} = \nu_z(t) P^z_{z'}(\Delta t) \tag{4.4.23}$$

of queues with state $z \in \mathcal{Z}$ in resulting state $z' \in \mathcal{Z}$ at the end of the decision epoch Δt. In total, we therefore have

$$\nu_{z'}(t+1) = \sum_{z \in \mathcal{Z}} \nu_{z,z'} = \sum_{z \in \mathcal{Z}} \nu_z(t) P_{z'}^z(\Delta t), \quad \forall z' \in \mathcal{Z}. \tag{4.4.24}$$

Computing the expected packet drops $D_z(t)$ per queue with state $z \in \mathcal{Z}$ is done analogously by

$$\dot{D}_z(t) = \lambda_z(\nu, t) P_B^z, \quad D_z(0) = 0 \tag{4.4.25}$$

resulting in a total average packet loss of

$$D(t) = \sum_{z \in \mathcal{Z}} \nu_z(t) D_z(\Delta t). \tag{4.4.26}$$

For exact computation of the terms in (4.4.24) - (4.4.26), observe that we have the linear matrix differential equation

$$\begin{bmatrix} \dot{\mathbf{P}}_z \\ \dot{D}_z(t) \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{Q}_z & \mathbf{0} \\ \lambda_z(t) \cdot \mathbf{e}_B^T & 0 \end{bmatrix}}_{\bar{\mathbf{Q}}_z \equiv \bar{\mathbf{Q}}(\nu(t), z)} \cdot \begin{bmatrix} \mathbf{P}_z \\ D_z(t) \end{bmatrix} \tag{4.4.27}$$

where we define the extended rate matrices $\bar{\mathbf{Q}}(\nu(t), z)$ analogously to $\mathbf{Q}(\nu(t), z)$, and thus obtain exact discretization by

$$\begin{bmatrix} \mathbf{P}_z(\Delta t) \\ D_z(\Delta t) \end{bmatrix} = \operatorname{Exp}\left(\bar{\mathbf{Q}} \Delta t\right) \cdot \begin{bmatrix} \mathbf{e}_z \\ 0 \end{bmatrix} \tag{4.4.28}$$

where $\operatorname{Exp}(\cdot)$ denotes the matrix exponential.

4.4.1 *Upper-level decision process*

We can now obtain a MDP, see Section 2.2.1, with state space $\mathcal{P}(\mathcal{Z}) \times \Lambda$ and action space $\mathcal{H} := \{h \colon \mathcal{Z}^d \to \mathcal{P}(\mathcal{U})\}$, since we have states $(\lambda(t), \nu(t))$ and actions $h(t)$ following dynamics

$$(\lambda(t+1), \nu(t+1)) \sim P_\lambda(\lambda(t)) \otimes \delta_{T_\nu(\nu(t), \lambda(t), h(t))} \tag{4.4.29}$$

$$h(t) = \tilde{\pi}(t, \nu(t), \lambda(t)) \tag{4.4.30}$$

where the transition function T_ν deterministically maps to $\nu(t+1)$ according to (4.4.24), and the actions are given by a deterministic 'upper-level' policy $\tilde{\pi} = \{\tilde{\pi}(t)\}_{t \geq 0}$, where $\tilde{\pi}(t) \colon \mathcal{P}(\mathcal{Z}) \times \Lambda \to \mathcal{H}$. See also Fig. 4.4 for a visualization of the Markov property through a probabilistic graphical model. Here, the randomness of the system stems from the random packet arrival rate $\lambda(t)$. Finally, by (4.4.26), the objective becomes

$$J(\tilde{\pi}) = \mathbb{E}\left[-\sum_{t=0}^{\infty} \gamma^t D(t) \right]. \tag{4.4.31}$$

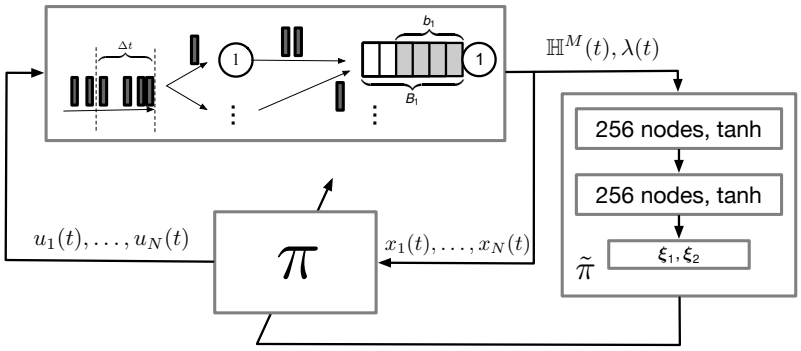

FIGURE 4.5: A schematic overview of the application of the upper-level mean-field control policy to the finite-agent finite-server system. The upper-level policy $\tilde{\pi}$ returns a lower-level policy π for a given distribution of server states $\mathbb{H}^M(t)$ and current arrival rate $\lambda(t)$. The lower-level policy is then applied separately to each agent state $x_i(t)$ to obtain an action $u_i(t)$.

The application of $\tilde{\pi}$ to the N-agent, M-queue case is visualized in Fig. 4.5, i.e. each of the agents $i = 1, \ldots, N$ first computes the decision rule $h(t) = \tilde{\pi}(t, \mathbb{H}^M(t), \lambda(t))$ according to the upper-level policy, and then samples its action $u_i(t) \sim h(t, x_i(t))$.

For the obtained MDPs, since the expected cost function and the dynamics are continuous in the states and actions of the MFC MDP, it is known that the typical dynamic programming principle (i.e. Bellman equation) holds, and an optimal stationary deterministic policy will exist.

Proposition 1 ([134], Theorem 4.2.3)**.** *There exists a stationary deterministic optimal policy $\tilde{\pi}$ that maximizes $J(\tilde{\pi})$.*

To find such a deterministic policy, an exact, closed-form solution is difficult due to the complexity of the associated transition model and continuous state and action spaces. Instead, we shall in the following employ well-established reinforcement learning techniques by exploring over stochastic policies $\tilde{\pi}(t) \colon \mathcal{P}(\mathcal{Z}) \times \Lambda \to \mathcal{P}(\mathcal{H})$, with the random decision rules $h(t) \sim \tilde{\pi}(t, \nu(t), \lambda(t))$ as actions of the MFC MDP, to find the desired optimal stationary deterministic policy.

4.5 THEORETICAL ANALYSIS

Although our formulated mean-field model is intuitively a good approximation of the finite system, in this section we shall make this connection rigorous. Note that our model does not immediately fit into standard MFC frameworks introduced in [53, 70], since we perform a double limit argument and continuous-to-discrete-time modeling. To verify the mean-field model, we shall show that performance in the finite system becomes arbitrarily close to the performance in the MFC system as long as the system is sufficiently large. For the following theoretical analysis, we shall consider the sequence of arrival rates $(\lambda_1, \lambda_2, \ldots)$ given a-priori by conditioning on them, i.e. non-random $\lambda^{N,M}(t) = \lambda^M(t) = \lambda(t)$.

Theorem 1. *The performance of the N, M system converges to the performance of the mean-field system under any stationary deterministic policy $\hat{\pi}$ as the system size becomes sufficiently large, i.e. for any $\varepsilon > 0$ there exists $N', M'(N') \in \mathbb{N}$ such that,*

$$\left|J(\hat{\pi}) - J^{N,M}(\hat{\pi})\right| < \varepsilon$$

for all $N > N', M > M'(N')$.

Proof. We will analyze

$$\begin{aligned}\left|J(\hat{\pi}) - J^{N,M}(\hat{\pi})\right| &\leq \left|J(\hat{\pi}) - J^{M}(\hat{\pi})\right| + \left|J^{M}(\hat{\pi}) - J^{N,M}(\hat{\pi})\right| \\ &\leq \sum_{t=0}^{\infty} \gamma(t) \left(\left|\mathbb{E}\left[D(t) - D^{M}(t)\right]\right| + \left|\mathbb{E}\left[D^{M}(t) - D^{N,M}(t)\right]\right|\right)\end{aligned}$$

where $D^M(t)$ denotes the random loss of packets in the infinite-agent finite-queue system.

For the first term, consider $M \to \infty$ and observe that

$$\begin{aligned}\mathbb{E}\left[D(t)\right] &= \mathbb{E}\left[\int \left(\text{Exp}\ (\bar{\mathbf{Q}}(\nu(t), z)\Delta t) \cdot \begin{bmatrix}\mathbf{e}_z \\ 0\end{bmatrix}\right)_{B+1} \nu(t)(\mathrm{d}z)\right], \\ \mathbb{E}\left[D^M(t)\right] &= \mathbb{E}\left[\frac{1}{M}\sum_j \left(\text{Exp}\ (\bar{\mathbf{Q}}^{M,j}\Delta t) \cdot \begin{bmatrix}\mathbf{e}_{z_j^M(t)} \\ 0\end{bmatrix}\right)_{B+1}\right] \\ &= \mathbb{E}\left[\int \left(\text{Exp}\ (\bar{\mathbf{Q}}(\mathbb{H}^M(t), z)\Delta t) \cdot \begin{bmatrix}\mathbf{e}_z \\ 0\end{bmatrix}\right)_{B+1} \mathbb{H}^M(t)(\mathrm{d}z)\right],\end{aligned}$$

with the rate matrices $\bar{\mathbf{Q}}^{M,j}$ of the infinite-agent finite-queue system, where the last equality follows since the rates in the M-queue case for each queue j are indeed given by

$$\begin{aligned}\lambda^{M,j}(t) &= M\lambda \int_{\mathcal{X}\times\mathcal{U}} \sum_{k=1}^{d} \mathbb{1}_{x_k=j\wedge u=k}\, \mathbb{G}^M(t)(\mathrm{d}x, \mathrm{d}u) \\ &= M\lambda \sum_{k=1}^{d} \sum_{x\in\mathcal{X}} \sum_{u\in\mathcal{U}} \mathbb{1}_{x_k=j\wedge u=k} \frac{1}{M^d} h(t, u \mid z_t^{M,x_1}, \ldots, z_t^{M,x_d}) \\ &= \lambda \sum_{k=1}^{d} \sum_{x\in\mathcal{X}} \sum_{u\in\mathcal{U}} \mathbb{1}_{x_k=j\wedge u=k} \frac{1}{M^{d-1}} h(t, u \mid z_{x_1}^{M}(t), \ldots, z_{x_d}^{M}(t)) \\ &= \lambda \sum_{k=1}^{d} \sum_{x_k\in\{1,\ldots,M\}} \sum_{x_{-k}\in\{1,\ldots,M\}^{d-1}} \sum_{u\in\mathcal{U}} \mathbb{1}_{x_k=j\wedge u=k} \frac{1}{M^{d-1}} h(t, u \mid z_{x_1}^{M}(t), \ldots, z_{x_d}^{M}(t)) \\ &= \lambda \sum_{k=1}^{d} \sum_{x_k\in\{1,\ldots,M\}} \sum_{x_{-k}\in\{1,\ldots,M\}^{d-1}} \sum_{u\in\mathcal{U}} \\ &\qquad \sum_{\bar{z}_k\in\mathcal{Z}} \sum_{\bar{z}_{-k}\in\mathcal{Z}^{d-1}} \mathbb{1}_{x_k=j\wedge u=k} \frac{1}{M^{d-1}} h(t, u \mid (\bar{z}_k, \bar{z}_{-k})) \mathbb{1}_{\bigwedge_{i=1}^{d} z_{x_i}^{M}(t)=\bar{z}_i}\end{aligned}$$

$$
\begin{aligned}
&= \lambda \sum_{k=1}^{d} \sum_{\bar{z}_k \in \mathcal{Z}} \sum_{\bar{z}_{-k} \in \mathcal{Z}^{d-1}} \sum_{u \in \mathcal{U}} \mathbb{1}_{\bar{z}_k = z_j^M(t) \wedge u = k} \cdot \underbrace{\frac{\sum_{x_{-k} \in \{1,\dots,M\}^{d-1}} \mathbb{1}_{\bigwedge_{i \neq k} z_{x_i}^M(t) = \bar{z}_i}}{M^{d-1}}}_{\prod_{i \neq k} \mathbb{H}^M(t, \bar{z}_i)} h(t, u \mid (\bar{z}_k, \bar{z}_{-k})) \\
&= \lambda \sum_{k=1}^{d} \sum_{\bar{z} \in \mathcal{Z}^d} \sum_{u \in \mathcal{U}} \mathbb{1}_{\bar{z}_k = z_j^M(t) \wedge u = k} \prod_{i \neq k} \mathbb{H}^M(t, \bar{z}_i) h(t, u \mid \bar{z}) \\
&= \lambda \sum_{\bar{z} \in \mathcal{Z}^d} \sum_{u \in \mathcal{U}} \mathbb{1}_{\bar{z}_u = z_j^M(t)} \prod_{i \neq u} \mathbb{H}^M(t, \bar{z}_i) h(t, u \mid \bar{z}) \\
&= \lambda \sum_{\bar{z} \in \mathcal{Z}^d} \sum_{u \in \mathcal{U}} \mathbb{1}_{\bar{z}_u = z_j^M(t)} \frac{\prod_{i=1}^{d} \mathbb{H}^M(t, \bar{z}_i)}{\mathbb{H}^M(t, z_j^M(t))} h(t, u \mid \bar{z}) \\
&= \frac{\lambda \int_{\mathcal{Z}^d \times \mathcal{U}} \mathbb{1}_{\bar{z}_u = z_j^M(t)} \left(\otimes_{k=1}^{d} \mathbb{H}^M(t) \otimes h(t)\right)(\mathrm{d}\bar{z}, \mathrm{d}u)}{\mathbb{H}^M(t, z_j^M(t))} \\
&= \lambda(t, \mathbb{H}^M(t), z_j^M(t))
\end{aligned}
$$

where the indices $-k$ denote all dimensions (indices) other than k.

Therefore, as long as $\mathbb{H}^M(t) \xrightarrow{d} \nu(t)$ (convergence in distribution), we find $\mathbb{E}\left[D(t) - D^M(t)\right] \to 0$ by the continuous mapping theorem. In particular, this holds true if $\mathbb{H}^M(t) \xrightarrow{p} \nu(t)$, i.e. for any $\delta > 0$ as $M \to \infty$,

$$\mathbb{P}\left(\left\|\mathbb{H}^M(t) - \nu(t)\right\| > \delta\right) \to 0.$$

We show this by induction: At $t = 0$ the statement holds by the law of large numbers. Now assume that the statement holds for t, then for $t + 1$ we first show that for any $\varepsilon, \delta > 0$ there exists $M', \delta' > 0$ such that for all $M > M'$ we have

$$\mathbb{P}\left(\left\|\mathbb{H}^M(t+1) - \nu(t+1)\right\| > \delta \;\middle|\; \left\|\mathbb{H}^M(t) - \nu(t)\right\| \leq \delta'\right) < \varepsilon.$$

Note that

$$
\begin{aligned}
&\mathbb{P}\left(\left\|\mathbb{H}^M(t+1) - \nu(t+1)\right\| > \delta \;\middle|\; \left\|\mathbb{H}^M(t) - \nu(t)\right\| \leq \delta'\right) \\
&\leq \sum_{z \in \mathcal{Z}} \mathbb{P}\left(\left|\mathbb{H}^M(t+1, z) - \nu(t+1, z)\right| > \delta \;\middle|\; \left\|\mathbb{H}^M(t) - \nu(t)\right\| \leq \delta'\right) \\
&\leq \sum_{z \in \mathcal{Z}} \mathbb{P}\left(\left|\mathbb{H}^M(t+1, z) - \mathbb{E}\left[\mathbb{H}^M(t+1, z) \mid \mathbb{H}^M(t)\right]\right| > \frac{\delta}{2} \;\middle|\; \left\|\mathbb{H}^M(t) - \nu(t)\right\| \leq \delta'\right) \\
&\quad + \sum_{z \in \mathcal{Z}} \mathbb{P}\left(\left|\mathbb{E}\left[\mathbb{H}^M(t+1, z) \mid \mathbb{H}^M(t)\right] - \nu(t+1, z)\right| > \frac{\delta}{2} \;\middle|\; \left\|\mathbb{H}^M(t) - \nu(t)\right\| \leq \delta'\right)
\end{aligned}
$$

and we shall bound the former term as follows: Define

$$\Delta_{z_j^M(t+1) | z_j^M(t)} f \coloneqq f(z_j^M(t+1)) - \mathbb{E}\left[f(z_j^M(t+1)) \mid f(z_j^M(t))\right]$$

and let $f \colon \mathcal{Z} \to \mathbb{R}$, then we have

$$\mathbb{P}\left(\left|\mathbb{H}^M(t+1, f) - \mathbb{E}\left[\mathbb{H}^M(t+1, f) \mid \mathbb{H}^M(t)\right]\right| > \frac{\delta}{2} \;\middle|\; \left\|\mathbb{H}^M(t) - \nu(t)\right\| \leq \delta'\right)$$

$$
\begin{aligned}
&= \mathbb{P}\left(\left|\frac{1}{M}\sum_{j=1}^{M}\Delta_{z_j^M(t+1)|z_j^M(t)}f\right| > \frac{\delta}{2} \;\middle|\; \left\|\mathbb{H}^M(t)-\nu(t)\right\| \le \delta'\right)\\
&\le \frac{4}{\delta^2}\mathbb{E}\left[\left(\frac{1}{M}\sum_{j=1}^{M}\left(\Delta_{z_j^M(t+1)|z_j^M(t)}f\right)\right)^2 \;\middle|\; \left\|\mathbb{H}^M(t)-\nu(t)\right\| \le \delta'\right]\\
&= \frac{4}{\delta^2 M^2}\sum_{j=1}^{M}\mathbb{E}\left[\left(\Delta_{z_j^M(t+1)|z_j^M(t)}f\right)^2 \;\middle|\; \left\|\mathbb{H}^M(t)-\nu(t)\right\| \le \delta'\right]\\
&\le \frac{16\max_z f(z)^2}{\delta^2 M} \to 0
\end{aligned}
$$

as $M \to \infty$ by conditional independence of $(z_1^M(t+1),\dots,z_M^M(t+1)$ given $z^M(t) = (z_1^M(t),\dots,z_M^M(t))$, the Chebyshev inequality and tower property. In particular, this holds for $f_z \equiv \mathbb{1}_{\{z\}}$, $z \in \mathcal{Z}$. Therefore,

$$
\sum_{z\in\mathcal{Z}}\mathbb{P}\left(\left|\mathbb{H}^M(t+1,z) - \mathbb{E}\left[\mathbb{H}^M(t+1,z) \mid \mathbb{H}^M(t)\right]\right| > \frac{\delta}{2} \;\middle|\; \left\|\mathbb{H}^M(t)-\nu(t)\right\| \le \delta'\right) \to 0
$$

as $M \to \infty$. For the latter term, note that analogously

$$
\begin{aligned}
&\left|\mathbb{E}\left[\mathbb{H}^M(t+1,f) \mid \mathbb{H}^M(t)\right] - \nu(t+1,f)\right|\\
&\quad\le \left|\sum_{z\in\mathcal{Z}} f(z)\sum_{z'\in\mathcal{Z}}\left(\mathbb{H}^M(t,z)-\nu(t,z)\right)\right.\\
&\qquad\qquad \left.\cdot\left(\operatorname{Exp}\left(\bar{\mathbf{Q}}(\mathbb{H}^M(t),z')\Delta t\right)\cdot\begin{bmatrix}\mathbf{e}_{z'}\\0\end{bmatrix}\right)_z\right|\\
&\qquad + \left|\sum_{z\in\mathcal{Z}} f(z)\sum_{z'\in\mathcal{Z}}\nu(t,z)\cdot\left(\operatorname{Exp}\left(\bar{\mathbf{Q}}(\mathbb{H}^M(t),z')\Delta t\right)\cdot\begin{bmatrix}\mathbf{e}_{z'}\\0\end{bmatrix}\right.\right.\\
&\qquad\qquad \left.\left.-\operatorname{Exp}\left(\bar{\mathbf{Q}}(\nu(t),z')\Delta t\right)\cdot\begin{bmatrix}\mathbf{e}_{z'}\\0\end{bmatrix}\right)_z\right|
\end{aligned}
$$

and by boundedness ($\lambda_{(\nu,z)}(t) \le d\lambda(t)$) and continuity in $\mathbb{H}^M(t), \nu(t)$, for any $\varepsilon > 0$ there exists $\delta' > 0$ such that $\left\|\mathbb{H}^M(t)-\nu(t)\right\| \le \delta'$ implies $\left|\mathbb{E}\left[\mathbb{H}^M(t+1,f) \mid \mathbb{H}^M(t)\right] - \nu(t+1,f)\right| < \varepsilon$. As a result, by the law of total probability

$$
\begin{aligned}
&\mathbb{P}\left(\left\|\mathbb{H}^M(t+1)-\nu(t+1)\right\| > \delta\right)\\
&= \mathbb{P}\left(\left\|\mathbb{H}^M(t+1)-\nu(t+1)\right\| > \delta \mid \left\|\mathbb{H}^M(t)-\nu(t)\right\| \le \delta'\right)\cdot\mathbb{P}\left(\left\|\mathbb{H}^M(t)-\nu(t)\right\| \le \delta'\right)\\
&\quad + \mathbb{P}\left(\left\|\mathbb{H}^M(t+1)-\nu(t+1)\right\| > \delta \mid \left\|\mathbb{H}^M(t)-\nu(t)\right\| > \delta'\right)\cdot\mathbb{P}\left(\left\|\mathbb{H}^M(t)-\nu(t)\right\| > \delta'\right)\\
&\le \mathbb{P}\left(\left\|\mathbb{H}^M(t+1)-\nu(t+1)\right\| > \delta \mid \left\|\mathbb{H}^M(t)-\nu(t)\right\| \le \delta'\right) + \mathbb{P}\left(\left\|\mathbb{H}^M(t)-\nu(t)\right\| > \delta'\right)\\
&\to 0
\end{aligned}
$$

since we can choose M', δ' according to the former analysis and the induction assumption, completing the induction step. It then follows at all times t by the continuous mapping theorem that

$$
\mathbb{E}\left[D(t) - D^M(t)\right] \to 0.
$$

For the second term, fix M and let $N \to \infty$. We find that

$$\mathbb{E}\left[D^M(t)\right] = \frac{1}{M}\sum_j \mathbb{E}\left[\left(\text{Exp}\ (\bar{\mathbf{Q}}_j^M \Delta t) \cdot \begin{bmatrix}\mathbf{e}_{z_j^M(t)} \\ 0\end{bmatrix}\right)_{B+1}\right],$$
$$\mathbb{E}\left[D^{N,M}(t)\right] = \frac{1}{M}\sum_j \mathbb{E}\left[\left(\text{Exp}\ (\bar{\mathbf{Q}}_j^{N,M} \Delta t) \cdot \begin{bmatrix}\mathbf{e}_{z_j^{N,M}(t)} \\ 0\end{bmatrix}\right)_{B+1}\right]$$

where $\bar{\mathbf{Q}}_j^{N,M}$ and $\bar{\mathbf{Q}}_j^M$ are continuous functions of

$$\lambda_j^{N,M}(t) = \lambda(t)\frac{M}{N}\sum_{i=1}^N \sum_{k=1}^d \mathbb{1}_{x_{i,k}(t)=j}\mathbb{1}_{u_i(t)=k},$$
$$\lambda_j^M(t) = \lambda(t) M \int_{\mathcal{X}\times\mathcal{U}} \sum_{k=1}^d \mathbb{1}_{x_k=j\wedge u=k}\, \mathbb{G}^M(t)(\mathrm{d}x, \mathrm{d}u),$$

and as $N \to \infty$, by the conditional law of large numbers [135, Theorem 3.5]

$$\lambda_j^{N,M}(t) \to \lambda_j^M(t)$$

a.s. conditional on $z_j^{N,M}(t) = z_j^M(t) = z$ for any $z \in \mathcal{Z}$. Therefore, again by the continuous mapping theorem, for all $j = 1, \ldots, M$ a.s.

$$\mathbb{E}\left[\text{Exp}\ (\bar{\mathbf{Q}}_j^{N,M}\Delta t) \;\middle|\; z_j^{N,M}(t) = z\right] \to \mathbb{E}\left[\text{Exp}\ (\bar{\mathbf{Q}}_j^M \Delta t) \mid z_j^M(t) = z\right].$$

At the same time, $z^{N,M}(t) \xrightarrow{d} z^M(t)$ at all times t as $N \to \infty$ via induction: For $t = 0$ trivially $\mathcal{L}(z^{N,M}(t)) = \nu(0) = \mathcal{L}(z^M(t))$. For $t+1$

$$\begin{aligned}
&\left|\mathbb{P}(z^{N,M}(t+1) = z) - \mathbb{P}(z^M(t+1) = z)\right| \\
&\leq \sum_{z'\in\mathcal{Z}} \left|\mathbb{P}(z^{N,M}(t) = z') - \mathbb{P}(z^M(t) = z')\right| \cdot \mathbb{P}(z^{N,M}(t+1) = z \mid z^{N,M}(t) = z') \\
&\quad + \sum_{z'\in\mathcal{Z}} \mathbb{P}(z^M(t) = z') \\
&\qquad \cdot \left|\mathbb{P}(z^{N,M}(t+1) = z \mid z^{N,M}(t) = z') - \mathbb{P}(z^M(t+1) = z \mid z^M(t) = z')\right|
\end{aligned}$$

where the former tends to zero by induction assumption, while for the latter we have

$$\begin{aligned}
&\left|\mathbb{P}(z^{N,M}(t+1) = z \mid z^{N,M}(t) = z') - \mathbb{P}(z^M(t+1) = z \mid z^M(t) = z')\right| \\
&\qquad = \left|\prod_{j=1}^M \mathbb{E}\left[\left(\text{Exp}\ (\bar{\mathbf{Q}}_j^{N,M}\Delta t) \cdot \begin{bmatrix}\mathbf{e}_{z'_j} \\ 0\end{bmatrix}\right)_{z_j} \;\middle|\; z^{N,M}(t) = z'\right]\right. \\
&\qquad\qquad \left. - \prod_{j=1}^M \mathbb{E}\left[\left(\text{Exp}\ (\bar{\mathbf{Q}}_j^M \Delta t) \cdot \begin{bmatrix}\mathbf{e}_{z'_j} \\ 0\end{bmatrix}\right)_{z_j} \;\middle|\; z^M(t) = z'\right]\right| \to 0
\end{aligned}$$

as $N \to \infty$ again as $\bar{\mathbf{Q}}_j^{N,M} \to \bar{\mathbf{Q}}_j^M$ conditionally a.s. for each j.

By Slutzky's theorem (on the conditional probability spaces given $z_j^{N,M}(t) = z_j^M(t) = z$), we have

$$\mathbb{E}\left[\left(\text{Exp }(\bar{\mathbf{Q}}_j^{N,M}\Delta t)\cdot\begin{bmatrix}\mathbf{e}_{z_j^{N,M}(t)}\\0\end{bmatrix}\right)_{B+1}\middle|\, z_j^{N,M}(t)=z\right]$$
$$\to\mathbb{E}\left[\left(\text{Exp }(\bar{\mathbf{Q}}_j^{M}\Delta t)\cdot\begin{bmatrix}\mathbf{e}_{z_j^{M}(t)}\\0\end{bmatrix}\right)_{B+1}\middle|\, z_j^{M}(t)=z\right]$$

for any $z \in \mathcal{Z}$, such that,

$$\begin{aligned}
&\mathbb{E}\left[\left(\text{Exp }(\bar{\mathbf{Q}}_j^{N,M}\Delta t)\cdot\begin{bmatrix}\mathbf{e}_{z_j^{N,M}(t)}\\0\end{bmatrix}\right)_{B+1}\right]\\
&=\sum_{z\in\mathcal{Z}}\mathbb{E}\left[\left(\text{Exp }(\bar{\mathbf{Q}}_j^{N,M}\Delta t)\cdot\begin{bmatrix}\mathbf{e}_{z_j^{N,M}(t)}\\0\end{bmatrix}\right)_{B+1}\middle|\, z_j^{N,M}(t)=z\right]\\
&\qquad\qquad\cdot\mathbb{P}\left(z_j^{N,M}(t)=z\right)\\
&\to\sum_{z\in\mathcal{Z}}\mathbb{E}\left[\left(\text{Exp }(\bar{\mathbf{Q}}_j^{M}\Delta t)\cdot\begin{bmatrix}\mathbf{e}_{z_j^{M}(t)}\\0\end{bmatrix}\right)_{B+1}\middle|\, z_j^{M}(t)=z\right]\cdot\mathbb{P}\left(z_j^{M}(t)=z\right)\\
&=\mathbb{E}\left[\left(\text{Exp }(\bar{\mathbf{Q}}_j^{M}\Delta t)\cdot\begin{bmatrix}\mathbf{e}_{z_j^{M}(t)}\\0\end{bmatrix}\right)_{B+1}\right]
\end{aligned}$$

which shows that $\mathbb{E}\left[D^{N,M}(t)\right] \to \mathbb{E}\left[D^M(t)\right]$ at all times t.

Now note that the terms $D(t), D^M(t), D^{N,M}(t)$ are uniformly bounded by the maximum expected average number of lost packets by dropping all packets, given by the expectation of the Poisson-distributed number of arriving packets $\lambda(t)\cdot\Delta t$. Therefore, for any $\varepsilon > 0$ we can choose T such that

$$\sum_{t=T}^{\infty}\gamma(t)\left(\left|\mathbb{E}\left[D(t)-D^M(t)\right]\right|+\left|\mathbb{E}\left[D^M(t)-D^{N,M}(t)\right]\right|\right)<\frac{\varepsilon}{3}.$$

Consequently choose M sufficiently large such that

$$\left|\mathbb{E}\left[D(t)-D^M(t)\right]\right|<\frac{\varepsilon}{3T},\quad\forall t\in\{0,1,\ldots,T-1\}$$

and similarly choose N sufficiently large to obtain

$$\left|\mathbb{E}\left[D^M(t)-D^{N,M}(t)\right]\right|<\frac{\varepsilon}{3T},\quad\forall t\in\{0,1,\ldots,T-1\}$$

according to the prequel, such that $\left|J(\hat{\pi})-J^{N,M}(\hat{\pi})\right|<\varepsilon$. □

Therefore, our mean-field model is well-motivated for sufficiently large systems, as we will also verify numerically.

4.6 SYSTEM MODEL EXTENSIONS

For the sake of completeness, we also represent two extensions to this model, one for heterogeneous servers and the other of infinite capacity queues.

4.6.1 *Heterogeneous servers*

Consider again M servers in the system, each server with its own queue. Each server j is defined by its serving rate $\mu_j^{N,M} \in \Omega$ sampled i.i.d. from some initial distribution, where for example $\Omega = \{\omega, 2\omega\}$, i.e. here we consider the example where some servers work at double the rate, 2ω, compared to other servers with rate ω. The speed assigned to the servers at the beginning shall not change. To reconcile with our approach, we replace the (queue) state of a server $z_j^{N,M}(t)$ by both its queue state and its speed $(z_j^{N,M}(t), \mu_j^{N,M})$. Let $\mathbb{H}^M(t)$ now be the empirical state distribution over both server rates and fillings

$$\mathbb{H}^{N,M}(t) = \frac{1}{M} \sum_{j=1}^{M} \delta_{z_j^{N,M}(t), \mu_j^{N,M}}. \tag{4.6.32}$$

Accordingly, the policy shall now depend on the joint distribution of queue states and speeds, i.e.

$$\begin{aligned} &\pi_i(t)(((z_i^{N,M}(t), \mu_1^{N,M}), \dots, (z_M^{N,M}(t), \mu_M^{M,M}(t))), x_i^{N,M}(t), \lambda^{N,M}(t)) \\ &\quad \equiv \hat{\pi}_t \left(\mathbb{H}_i^{N,M}(t), ((z_{x_i^1(t)}^{N,M}(t), \mu_{x_i^1(t)}^{N,M}), \dots, ((z_{x_i^d(t)}^{N,M}(t), \mu_{x_i^d(t)}^{N,M}), \lambda^{N,M}(t) \right) \end{aligned} \tag{4.6.33}$$

for all $i = 1, \dots, N$. The rate of the server affects the departure from that queue, since a higher rate will have relatively faster departures and vice versa. This will also affect the optimal policy, since in order to avoid job drops, allocating more jobs to faster queues will result in less downtime. The arrival rates shall remain unaffected.

The rest of the derivation proceeds analogously to the homogeneous case. As a result, the limiting mean-field state distribution shall also contain not only the buffer filling but also the speed of the server, $\nu(t) \in \mathcal{P}(\mathcal{Z} \times \Omega)$. This will be represented by a vector of length $|\Omega|(B+1)$, since there are $|\Omega|$ types of servers available. The speed of the servers is assumed to remain fixed, such that it does not evolve. However, the buffer filling will now evolve according to

$$\nu_{z',\mu}(t+1) = \sum_{z \in \mathcal{Z}} \nu_{z,\mu}(t) P_{z',\mu}^{z,\mu}(\Delta t), \quad \forall z' \in \mathcal{Z}, \forall \mu \in \Omega \tag{4.6.34}$$

where $P_{z',\mu}^{z,\mu}$ is the probability of a queue in state z to change to state z' when the serving rate of the server is $\mu \in \Omega$. The expected packet drops per queue with state $z \in \mathcal{Z}$ are calculated via

$$\dot{D}^{z,\mu}(t) = \lambda_{z,\mu}(t) P_B^{z,\mu}, \quad D^{z,\mu}(\tau) = 0 \tag{4.6.35}$$

resulting in a per-queue average packet loss of

$$D(t) = \sum_{\mu \in \Omega} \sum_{z \in \mathcal{Z}} \nu_{z,\mu}(t) D^{z,\mu}(\Delta t) \tag{4.6.36}$$

where $\lambda_{z,\mu}(t)$ is the arrival rate to queues with filling z and speed $\mu \in \Omega$, which is given analogously to (4.3.19). The objective function for this model stays the same as given in (4.4.31), since we still have finite capacity queues, and we want to reduce packet drops. The exact discretization is performed as for the homogeneous system.

4.6.2 *Infinite buffer capacity*

For infinite capacity queues, i.e., $B = \infty$, our queue state distribution is defined as $\nu(t) \in \mathcal{P}(\mathcal{Z})$, where $\mathcal{Z} \coloneqq \{0, 1, 2, \ldots\} \in \mathbb{N}_0$ is now infinite. Therefore, the reward function must change, since there will be no more job drops.

The majority of the formulation remains the same as given in Section 4.3.2. The exact discretization from Section 4.4 also stays the same, except that we now deal with infinite dimensions. This model can also be easily extended to the heterogeneous server model, as done in Section 4.6.1. Most importantly, the objective is changed, since there are no packet drops anymore.

For the objective, we shall instead penalize based on average waiting time of arriving jobs,

$$J(\hat{\pi}) = \mathbb{E}\left[-\sum_{t=0}^{\infty} \gamma^t W(t) \right], \tag{4.6.37}$$

where the average waiting time of all arriving jobs per queue shall be denoted by W_t. The average waiting time of a job arriving in a queue with $z \in \mathcal{Z}$ current jobs is simply given by $(z+1)/\mu$ for the servicing rate μ. As a result, analogously to $D(t)$, the expected total waiting times of all arriving jobs in a queue with $z \in \mathcal{Z}$ fillings during decision epoch t can be computed through

$$\dot{W}^z(t) = \sum_{z' \in \mathcal{Z}} \frac{z'+1}{\mu} P^z_{z'}, \quad W^z(\tau) = 0 \tag{4.6.38}$$

with the average over all queues given by

$$W(t) = \sum_{z} \nu_z(t) \cdot W^z(\Delta t)\,. \tag{4.6.39}$$

For the infinite capacity queue case, the linear matrix differential equation from Section 4.4 now becomes

$$\begin{bmatrix} \dot{\mathbf{P}}^z \\ \dot{W}^z(t) \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{Q} & 0 \\ \lambda_z(t) \cdot \mathbf{B} & 0 \end{bmatrix}}_{\mathbf{Q}'} \cdot \begin{bmatrix} \mathbf{P}^z \\ W^z(t) \end{bmatrix} \tag{4.6.40}$$

TABLE 4.1: System parameters used in the experiments.

Symbol	Name	Value
Δt	Time step size	$1 - 10$
μ	Service rate	1
(λ_h, λ_l)	Arrival rates	$(0.9, 0.6)$
N	Number of agents	$1000 - 1000000$
M	Number of queues	$100 - 1000$
d	Number of accessible queues	2
n	Monte Carlo simulations	100
B	Queue buffer size	5
$\nu(0)$	Queue starting state distribution	$[1, 0, 0, \ldots]$
D	Drop penalty per job	1
T	Training episode length	500
T_{e}	Evaluation episode length	$50 - 500$

TABLE 4.2: Hyperparameter configuration for PPO.

Symbol	Name	Value
γ	Discount factor	0.99
λ_{RL}	GAE lambda	1
κ	KL coefficient	0.2
ϵ	Clip parameter	0.3
l_r	Learning rate	0.00005
B_b	Training batch size	4000
B_m	SGD Mini batch size	128
T_b	Number of epochs	30

with $\mathbf{B} = [\frac{1}{\mu}, \ldots, \frac{B+1}{\mu}, \ldots]$ and thus we obtain exact discretization by

$$\begin{bmatrix} \mathbf{P}^z(\tau + \Delta t) \\ W^z(\tau + \Delta t) \end{bmatrix} = \mathrm{Exp}\ (\mathbf{Q}' \Delta t) \cdot \begin{bmatrix} \mathbf{e}_z \\ 0 \end{bmatrix} . \tag{4.6.41}$$

To obtain numerical tractability, since we cannot represent infinite-dimensional vectors numerically, we propose to cut off both simulation and parametrization of $\mathcal{H}$ at some high $B_{\max} \in \mathbb{N}$, i.e. any queue with fillings going beyond $B_{\max}$ are treated as having $B_{\max}$ fillings. This approach remains well-founded as long as the serving rates μ of the servers are greater than the maximum arrival rate [136], and that can be achieved by assigning all packets to a subset of queues with z fillings, since in that case the likelihood of achieving queue fillings larger than $B_{\max}$ tends to zero as $B_{\max} \to \infty$. Concretely, this holds true as long as

$$\mu > d \cdot \lambda_z(t) \quad \forall z \in \mathcal{Z}, \tag{4.6.42}$$

since the arrival rate $\lambda_z(t)$ is trivially bounded by $d \cdot \lambda(t)$.

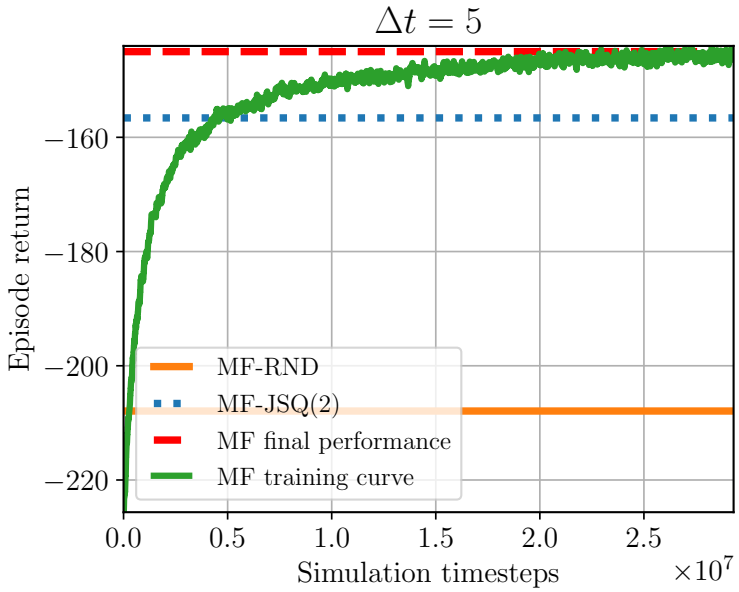

FIGURE 4.6: Training curve for the MF policy for $\Delta t = 5$ and $T_e = 500$ timesteps – i.e. the expected negative number of packet drops per episode during training – together with a comparison to the MF-JSQ(2) and MF-RND policies. The horizontal lines indicate the estimated expected returns for each policy. The red dotted line indicates the final achieved return of the learned MF policy in the mean-field MDP.

4.7 EVALUATION

In this section, we will begin by giving details on the experimental setup. Afterward, we will demonstrate numerical results of applying reinforcement learning to the MFC MDP problem.

We have M homogeneous queues with exponential service rate μ and N load-balancers (agents) with Markov modulated arrival rate λ. Beginning with $\lambda_0 \sim \text{Unif}(\{\lambda_h, \lambda_l\})$, at each decision epoch the arrival rate switches between high, λ_h, and low, λ_l, levels, using the transition law

$$\mathbb{P}(\lambda(t+1) = \lambda_l \mid \lambda(t) = \lambda_h) = 0.2, \tag{4.7.43}$$
$$\mathbb{P}(\lambda(t+1) = \lambda_h \mid \lambda(t) = \lambda_l) = 0.5. \tag{4.7.44}$$

In general, the experiments could be conducted with more levels of arrival rates and with different modulation rates estimated from a real system, though in this chapter we will use two arbitrarily chosen values to show the theoretical applicability of our methodology. The values for the system parameters in all of our experiments are given in Table 4.1.

In order to assess the performance of our MF policy, we compare it to the following policies:

- Join-the-shortest-out-of-d-queues (JSQ(d)): The agents, at every decision epoch, select d out of M queues and jobs are allocated to the shortest one.

- Random (RND): The agents, at every decision epoch, select d queues randomly out of M and allocate the jobs to a random queue out of these d queues, which will be equivalent to a completely random selection out of M queues for sufficiently large $N \gg M$.

In this chapter, we have used $d = 2$, since in [113] it has been shown that while moving from $d = 1$ to $d = 2$ shows an exponential increase in performance of JSQ(d), an additional increase to $d = 3$ does not add much in terms of achieved performance.

In order to obtain our MF policy by solving the optimal control problem, we apply proximal policy optimization PPO, see Section C for details on it, using the RLlib implementation [137], a well-known and robust policy gradient reinforcement learning algorithm. The learning algorithm hyperparameters used in our experiments can be found in Table 4.2. A detailed description of what these parameters do is given in Appendix C.

input : System parameters from Table 4.1, Markovian upper-level policy $\tilde{\pi} = (\tilde{\pi}(t))_{t\geq 0}$
output : Number of dropped packets, D

Initialize $\lambda(0) \sim \mathrm{Unif}(\{\lambda_h, \lambda_l\})$.

for $j = 1, \ldots, M$ **do**
  Initialize queue states $z_j(0) \sim \nu(0)$.
end
for $t = 0, 1, \ldots, T_\mathrm{e}$ **do**
  Compute empirical distribution $\mathbb{H}^M(t) = \frac{1}{M}\sum_{j=1}^{M} \delta_{z_j(t)}$
  Sample decision rule $h(t) \sim \tilde{\pi}(\mathbb{H}^M(t), \lambda(t))$
  for $i = 1, \ldots, N$ **do**
    Sample agent state $x_i(t) \sim \otimes_{k=1}^{d} \mathrm{Unif}(\{1, \ldots, M\})$
    Compute anonymous state $\bar{z}_i(t) = (z_{x_{i,1}(t)}(t), \ldots, z_{x_{i,d}(t)}(t))$
    Sample agent action $u_i(t) \sim h(t)(\bar{z}_i(t))$
  end
  for $j = 1, \ldots, M$ **do**
    Simulate the CTMC y_j with jump rates $\lambda_j(t), \alpha$ and $y_j(0) = z_j(t)$ for Δt time units
    Count number of dropped packets, D
    Set queue state $z_j(t+1) = y_j(\Delta t)$
  end
  Sample $\lambda(t+1) \sim \mathbb{P}_\lambda(\lambda(t+1) \mid \lambda(t))$
end
return D

FIGURE 4.7: Application of MFC policy in finite system.

For the policy architecture, we use a standard feedforward neural network with two hidden layers of 256 nodes, outputting the parameters of a $|\mathcal{Z}^d||\mathcal{U}|$-dimensional diagonal Gaussian distribution. The input of the policy network consists of the fractions of queues in each state, $\nu(t)$ as well as the current arrival rate $\lambda(t)$. To allow the $|\mathcal{Z}^d||\mathcal{U}|$ real-valued network

output values $a_{i,j}$ to parametrize elements in $\mathcal{H}$, we squash the values to $[0, 1]$ using the $\tanh$ function, i.e.

$$\tilde{a}_{i,j} = \frac{\tanh a_{i,j}}{2} + \frac{1}{2} \tag{4.7.45}$$

and then normalize each row to obtain

$$h(u_j \mid \bar{z}_i) \equiv \frac{\tilde{a}_{i,j}}{\sum_{j=1}^{|\mathcal{U}|} \tilde{a}_{i,j}} \tag{4.7.46}$$

for some arbitrary ordering of states and actions $\{\bar{z}_1, \dots, \bar{z}_{|\mathcal{Z}^d|}\} \equiv \mathcal{Z}^d$, $\{u_1, \dots, u_{|\mathcal{U}|}\} \equiv \mathcal{U}$.

In Fig. 4.6, we observe the learning curve of the applied reinforcement learning algorithm for $\Delta t = 5$ and find that the simple parameterization of the lower-level policies is indeed successful and leads to stable learning. For the demonstrated experiment, we trained in parallel (offline) on 20 cores of a commodity server CPU for approximately 35 hours, after which the optimal policy can be applied in practice, to finite systems. Here, MF-JSQ(2) and MF-RND refer to the corresponding JSQ and RND policies in the mean-field model, i.e. each applies a fixed $h(t)$ regardless of the current queue state distribution $\nu(t)$. In the case of MF-JSQ given by

$$h(u \mid \bar{z}) = \begin{cases} 0 & \text{if } u \notin \arg\min_{u'} \bar{z}_{u'} \\ \frac{1}{N_{\min}} & \text{else} \end{cases} \tag{4.7.47}$$

where $N_{\min}$ is the number of actions u that minimize the chosen queue's state $\bar{z}_u$. In the case of MF-RND, we similarly choose

$$h(u \mid \bar{z}) = \frac{1}{|\mathcal{U}|}, \quad \forall (\bar{z}, u) \in \mathcal{Z}^d \times \mathcal{U} \tag{4.7.48}$$

As expected, indicated by the horizontal lines, the JSQ(2) and random (RND) assignment policies in the mean-field case are both suboptimal for the chosen delay time of $\Delta t = 5$, and our reinforcement learning approach is capable of finding better load balancing policies after approximately 5 million simulated decision epochs. Though we have tried Dirichlet-parameterized upper-level policies to directly output simplex-valued actions in order to eliminate the need for manual normalization, we found that performance was significantly worse, hence motivating our approach.

PERFORMANCE COMPARISON ON FINITE SYSTEMS: We will now compare the performance of the evaluated load balancing algorithms on systems of finite size. For simulation of the finite-agent and finite-queue system, we simulate the continuous-time Markov processes exactly by sampling exponential waiting times for all events according to the Gillespie algorithm [138]. For an easy comparison between different Δt, we set the episode lengths T_{e} for evaluation to the integer nearest to $\frac{500}{\Delta t}$. Pseudo-code for simulating and applying our MF policy in the finite system is given in Algorithm 4.7 [1].

1 https://github.com/AnamTahir7/mfc_large_queueing_systems.git

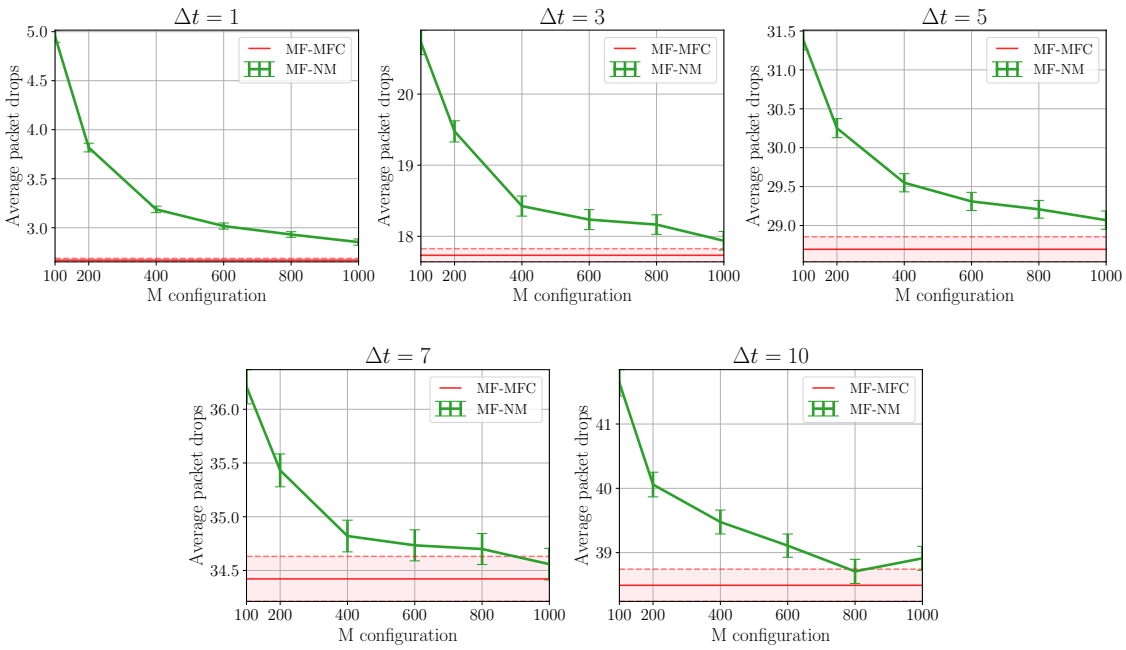

FIGURE 4.8: Comparison of the estimated expected packet drops (lower is better) of MF policies over the number of queues M in the finite system for different values of Δt, together with 95% confidence intervals depicted as shaded regions and error bars. Here, we use total running times of approximately 500 time units, and $N = M^2$ to fulfill $N \gg M$. The red dotted line indicates the equivalent achieved return of the learned MF policy in the mean-field control MDP, i.e. the limiting model as $N \gg M \to \infty$. It can be observed that as the system size $N = M^2$ increases, the performance under the MF policy (green) becomes increasingly close to the mean-field system performance (red), validating the accuracy of our mean-field formulation.

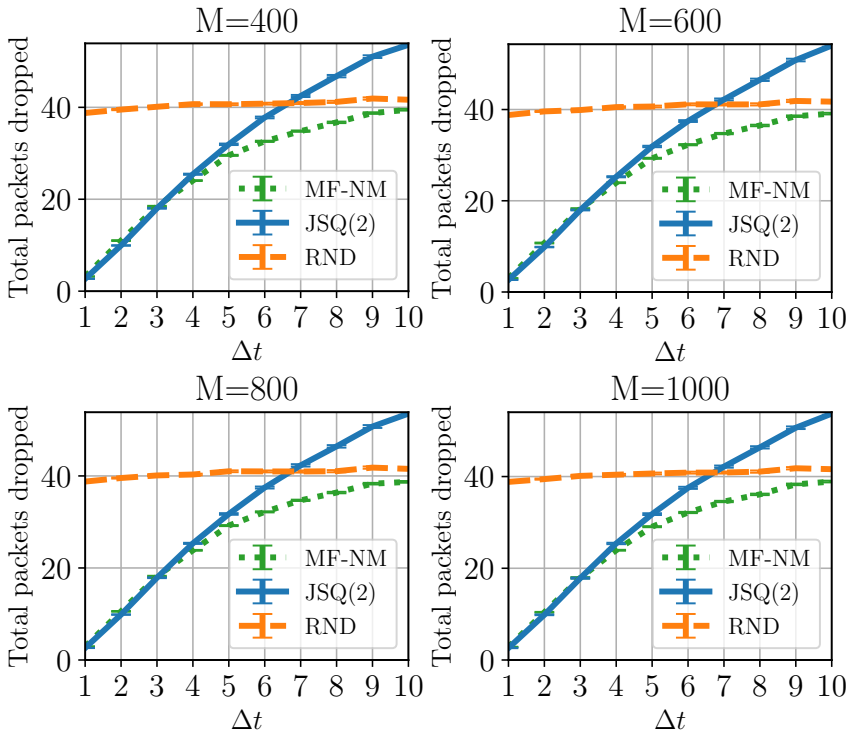

FIGURE 4.9: Comparison of the estimated expected packet drops of MF, JSQ(2), RND policies together with 95% confidence intervals for different configurations of M and $N = M^2$. We keep the total running time of each setting approximately equal to 500 time units to compare the effect of Δt. It can be observed that as Δt rises, the achievable performance by choosing emptier queues degrades.

In Fig. 4.8, we show that the performance of the final learned MF policies over a wide range of delays Δt and system sizes (N, M). It can be seen that the overall achievable performance of our MF policy increases up to the performance achieved in the MFC MDP (red dotted line) as the system size (N, M) becomes sufficiently large ($N \gg M \gg 1$). Hence, our findings empirically validate the fact that our mean-field approximations are indeed accurate for sufficiently large system sizes.

The returns for the policies at each Δt, for the case where all experiments are run for approximately equal overall time instead of an equal number of decision epochs, are given in Fig. 4.9. Here, we have trained a separate MF policy for each of the Δt and, compared to JSQ(2) and RND. It can be seen that – as expected due to fewer updates – the overall achievable performance in the system worsens as the synchronization delay Δt of the system increases. It can be seen that MF achieves better performance than JSQ(2) starting from $\Delta t > 2$, while it outperforms RND in all cases. This stems from the fact that reinforcement learning only finds approximately optimal solutions. Nonetheless, at an intermediate level of synchronization delay beginning with $\Delta t = 3$, our learning-based methodology appears to be able to find a better policy than the optimal policies for $\Delta t \to 0$ (JSQ(2)) and $\Delta t \to \infty$ (RND). Even for small $\Delta t = 1$, our MF policy has comparable performance to the optimal JSQ(2) policy, as long as N, M are sufficiently large. As Δt keeps increasing, MF and RND are therefore expected to perform equally good in sufficiently large systems as long as we indeed have $N \gg M$.

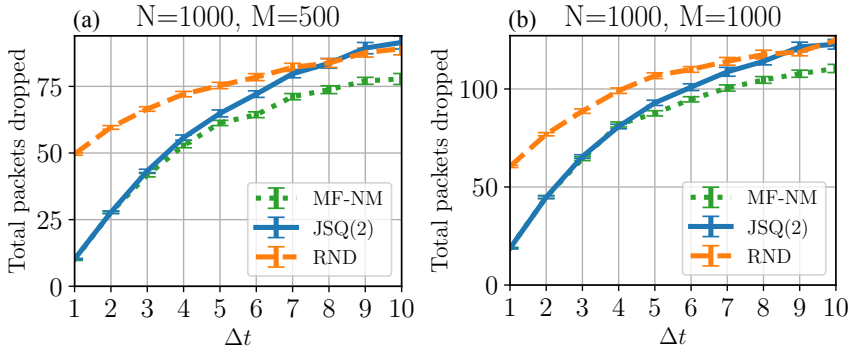

FIGURE 4.10: Comparison of the estimated expected packet drops of MF, JSQ(2), RND policies together with 95% confidence intervals for the same setting as in Fig. 4.9, equal total running time, for the case when $M = 1000$, $N = \frac{M}{2}$ and $N = M$. As Δt increases, the performance of our MF policy performs better than the other policies, even when $N \not\gg M$.

Finally, we perform experiments for $N \not\gg M$, i.e. we violate the formal approximation assumption used to obtain our mean-field system. Even though the assumptions made in our approximation are violated, our policy nonetheless obtains good comparative performance. As shown in Fig. 4.10, we find that the qualitative performance differences remain the same for around 1000 agents and queues. It can also be observed that the random policy no longer obtains approximately equal performance over Δt, which is caused by the fact that the queues are increasingly sampled unequally often by an agent, and resampling resolves the resulting increased focus on a subset of queues.

4.8 SUMMARY

In this chapter, we have proposed a mean-field-control-style formulation, with enlarged state-action space, for large-scale distributed queuing systems with synchronization delays. We have achieved this by formulating the finite-Nagent finite-Mqueue system and then considering $N \to \infty$, $M \to \infty$.

Firstly, we provide theoretical performance guarantees which show that the performance in the N, M system becomes arbitrarily close to the performance in the MFC system as long as N, M are large enough. Then, assuming a synchronous system with exact discretization of the underlying processes, we end up with an exactly discretized discrete-time Markov decision process on which we have applied reinforcement learning algorithm, PPO. As a result, we find that our learned solution can outperform the delay-free-optimal JSQ(d) policy as well as the infinite-delay-optimal random policy in the regime of intermediate delays Δt, even if $N \not\gg M$ as long as the system size N, M is sufficiently large.

5

LOCALIZED CONTROL FOR LARGE QUEUING SYSTEMS

This chapter extends the work of Chapter 4 to consider localized queuing systems. As mentioned in the previous chapter, scalable load balancing algorithms are of great interest in cloud networks and data centers, necessitating the use of tractable techniques to compute optimal load balancing policies for good performance. However, most existing scalable techniques, especially asymptotically scaling methods based on mean-field theory, have not been able to model large queuing networks with strong locality. Meanwhile, general multi-agent reinforcement learning techniques can be hard to scale and usually lack a theoretical foundation. In this chapter, we address this challenge by leveraging recent advances in sparse mean-field theory to learn a near-optimal load balancing policy in sparsely connected queuing networks in a tractable manner, which may be preferable to global approaches in terms of communication overhead. This work is based on the currently under peer review publication [3]. Importantly, we obtain a general load balancing framework for a large class of sparse bounded-degree topologies. By formulating a novel mean-field control problem in the context of graphs with bounded degree, we reduce the otherwise difficult multi-agent problem to a single-agent problem. Theoretically, the approach is justified by approximation guarantees. Empirically, the proposed methodology performs well on several realistic and scalable network topologies. Moreover, we compare it with a number of well-known load balancing heuristics and with existing scalable multi-agent reinforcement learning methods. Overall, we obtain a tractable approach for load balancing in highly localized networks.

Related work

The aforementioned decentralized queuing systems with underlying graph structure can be modeled as various variants of multi-agent (partially-observable) MDP [48, 96]. The goal is typically to learn an optimal load balancing policy, which has been done by using numerous available MARL approaches, mainly building upon MMDP [139–141] and decentralized partially observable MDPs (Dec-POMDP) [35, 36]. However, these methods are usually either not highly scalable or are difficult to analyze [118]. This has resulted in recent popularity of mean-field theory for modeling multi-agent systems, via a single representative agent interacting with the mean-field (distribution) of all agents [12, 142]. Within mean-field theory, one differentiates between mean-field games (MFG) for agents in a competitive setting [50] and mean-field control (MFC) for cooperation [59]. This chapter focuses on the latter for reducing the overall jobs dropped in the system.

Note that mean-field limits have also been used to study load balancing algorithms, such as JSQ and the power-of-d variants (where the load-balancer only obtains the state information of $d \ll M$ out of the total available queues) [113], in terms of sojourn time and average queue lengths [20, 115, 116]. The asymptotic analysis of load balancing policies for systems with different underlying random dense graphs has already been studied to show that as long as the degree $d(M)$ scales with the number of servers, the topology does not affect the performance of the [143], while here we consider the sparse case. Graphons are also used to describe the limit of dense graph sequences [144] and have been studied with respect to both MFG and MFC [145–147]. However, for sparse graphs, the limiting graphon is not meaningful, making it unsuitable for networks whose degree does not scale with system size, see also [144, 148, 149]. MFGs for relatively sparse networks have been studied using Lp-graphs [150], but they too cannot be extended to bounded degrees. We believe that bounded-degree graph modeling is necessary to truly represent and analyze large fixed-degree distributed systems to avoid the need for increasingly global interactions and thus difficulties in scaling.

In models where the graph degree is fixed and small, for different topologies such as the ring and torus, the power-of-two policy was studied using pair approximation to analyze its steady state and to show that choosing between the shorter of two local neighboring queues improves the performance drastically over purely random job allocations [151]. In contrast, in this chapter, we will also look at a similar model, where the degree d does not scale with the number of agents, and find that one can improve beyond existing power-of-d policies. In order to find scalable load balancing policies, we apply relatively new results from sparse mean-field approximations, see [142] and references therein. Recently, mean-field analysis of the load balancing policies was done using the coupling approach but mainly for spatial graphs and only for power-of-d algorithms, see [152]. They consider tasks and servers which may be non-exchangeable on random bipartite geometric graphs and random regular bipartite graphs. To the best of our knowledge, MFC on sparse (bounded degree) graphs has neither been analyzed theoretically in its general form, nor has it been applied to queuing systems.

Additionally, algorithms such as JSQ, SED and their power-of-d variant are based on the unrealistic assumption of instantaneous knowledge of the queue states, which is not true

in practice, especially in large systems. We remove this assumption by introducing a synchronization (communication) delay Δt into our system model. Though in non-local systems it has already been proven that for $\Delta t = 0$, JSQ is the optimal load balancing policy, it is also known that as the delay increases, JSQ fails due to the phenomenon of "herd behavior" [113], i.e. allocating all packets simultaneously to momentarily empty queues. It has also been shown that as $\Delta t \to \infty$, RND allocation is the optimal policy [113]. While for the in-between range of synchronization delays in non-sparse queuing systems, a scalable policy has been learned before using mean-field approximations [2], it only considered fully connected graphs with sampling d neighbors, and not the sometimes more realistic scenario of local, sparsely connected queuing networks. For other works in this direction, we refer to [117, 118, 143] and references therein.

5.1 QUEUING SYSTEM MODEL

We consider a system with a set of $\mathcal{N} = \{1, \dots, N\}$ load-balancers/agents and $\mathcal{M} = \{1, \dots, M\}$ servers, where each server has its own queue with finite buffer capacity, B, see also Fig. 5.1 for an overview. The queues work in a first-in-first-out (FIFO) manner, and servers take the jobs one at a time from their queue, processing them at an exponential rate μ. Once a job has been processed, it leaves the system forever. Somewhat to the successful power-of-d policies [113], we assume that each load-balancer accesses only a limited number (e.g. d out of the M) of available queues and can only allocate its arriving jobs to these d accessible queues, with $d \ll M$ and fixed. We assume $M = N$ and associate each server (queue) with one agent, though it is possible to extend the model to varying or even random numbers of queues per agent.

Jobs arrive to the system according to a Markov modulated Poisson process, with total rate $\lambda(t)M$, and then are divided uniformly amongst all agents, which is also equivalent to independent Poisson processes at each agent given the shared arrival rate $\lambda(t)$ by Poisson thinning [133]. The agent takes the allocation action based on a learned or predetermined, memory-less policy ζ, which considers the current queue state information of its d accessible queues. This information is periodically sent by servers to neighboring agents, such that agents only obtain information on d queues, reducing the amount of messages to be sent. If the job allocation is done to an already full queue, it is dropped and results in a penalty. Similarly, jobs depart from a queue at a rate μ. The goal of the agent is to minimize the overall number of jobs dropped.

5.1.1 *Locality and Scalability*

Note that in contrast to many other analyzed settings, in this chapter the d out of M available queues are not sampled randomly for each package, but instead fixed for each agent according to some predefined topology (see also Section 5.5.1). In other words, we assume a strong concept of locality where agents have access only to a very limited subset of queues, implying also a strong sense of partial observability in the system. The value for d therefore depends on the type of graph topology being considered. Note

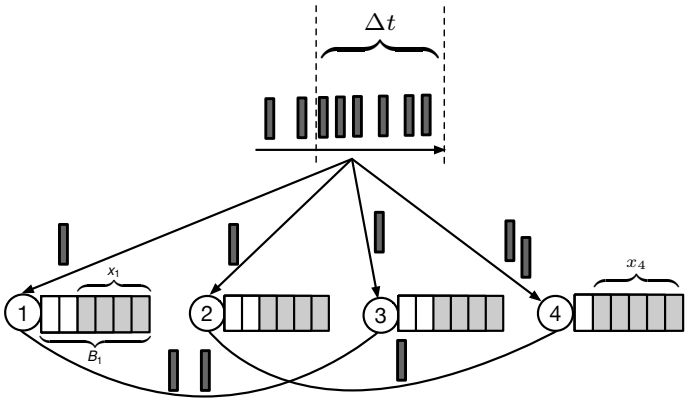

FIGURE 5.1: System model for $N = M = 4$. Jobs arriving in a certain synchronization time Δt are allocated to all agents using Poisson thinning. Each agent can allocate its incoming jobs to its own queue or the other queue it has access to, indicated by the connecting edges between agents $(1, 3)$ and agents $(2, 4)$. These connections will be defined by the chosen graph topology, see Section 5.5.1.

that an agent always has access to its own queue, as we associate each of the M queues with an agent. Accordingly, our queuing model contains an associated static underlying undirected graph $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ is the set of edges between vertices (agents) $\mathcal{N} \coloneqq \{1, \dots, N\}$ based on the d queue accessible by the agent. An agent i will have an edge $(i, j) \in \mathcal{E}$ to another agent j whenever they have access to the queues associated to that agent, and vice versa. Therefore, an agent can have a maximum of d edges (neighbors) to other agents. We will denote the set of neighbors of agent i as $N_i \coloneqq \{j \in \mathcal{N} \mid (i, j) \in \mathcal{E}\}$.

The motivation to use a graph structure with bounded degree arises from the fact that the corresponding model allows us to find more tractable local solutions for large systems. We need large systems that are regular in some sense, otherwise the system is too heterogeneous to find a tractable and sufficiently regular solution. Therefore, we look at systems where the regularity can be expressed graphically. The regularity condition already includes basic regular graphs such as grids and torus, but also allows many other random graphs such as the configuration model, see also Section 5.5.1. Here, we then apply RL and MFC to find otherwise hard-to-compute near-optimal queuing strategies in this highly local setting. The simplicity of the queuing strategy – instantaneously assigning a packet to one of the neighboring queues based on periodic and local information – not only allows for fast execution and high scalability, since information does not need to be exchanged for each incoming packet, but also allows for easy addition of more nodes to scale the system to arbitrary sizes.

In the following, we will obtain tractable solutions by first formulating the finite queuing system, then formulating its' limiting mean-field equivalent as the system grows, and lastly applying (partially-observed) RL and MFC, see Fig. 5.2 for a generic example.

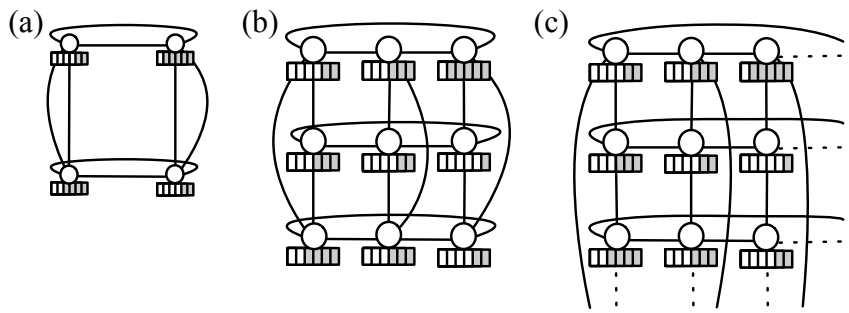

FIGURE 5.2: An example queuing system on a generic sparse, regular graph topology (2D torus). The circular nodes are the servers, and each server has its own finite buffer queue. From left to right, the size of the underlying graph is increasing in a regular manner.

5.1.2 *Finite Queuing System*

To begin, consider the following system. Each agent i is associated with a local state, $x_i \in \mathcal{X} \coloneqq \{0, 1, \ldots, B\}$ and local action $u_i \in \mathcal{U}_i$. The state x_i will be the current queue filling of the d queues from $\mathcal{M}$ accessible to the agent i, and the set of actions $\mathcal{U}_i$ will be the set of these d accessible queues. Hence, we have a finite set of action and state space. The global state of the system is given by $\mathbf{x} = (x_1, \ldots, x_N) \in \mathcal{X}^N$. Similarly, the global action is defined as $\mathbf{u} = (u_1, \ldots, u_N) \in \mathcal{U}^N \coloneqq \mathcal{U}_1 \times \cdots \times \mathcal{U}_N$.

5.1.2.1 *Synchronous system*

We want the agents to work in a synchronized manner, e.g. to model communication delays, and also for the servers to send their queue state information to the respective agents once every fixed time interval. To achieve this, we model our system at discrete decision epochs $\{0, \Delta t, 2\Delta t, 3\Delta t, \ldots\}$, where $\Delta t > 0$ is the synchronization delay, the time passing between each decision epoch. The interval Δt may be understood as a type of synchronization or update delay, assuming that it takes Δt amount of time to obtain updated local information from the servers and update the queuing strategy (e.g. routing table). Note that we may easily adjust Δt to approximate continuous time. Using this delay, we can later model our system as a MFC-based RL problem and learn the optimal policy using state-of-the-art RL-based algorithms [26].

5.1.2.2 *Localized policies*

For the moment, let each agent be associated with a localized policy $\zeta_i^{\theta_i}$ parameterized by θ_i which defines a distribution of the local action u_i conditioned on the states of its neighborhood $\mathbf{x}_{N_i}(t)$. The size of the neighborhood depends on d and the type of graph used. Given the global state $\mathbf{x}(t)$, each agent takes an action $u_i(t)$ which is independently drawn from $\zeta_i^{\theta_i}(\cdot \mid x_{N_i}(t))$. In other words, agents do not coordinate with each other, and decide independently where to send arriving packets. The parameters $\theta = (\theta_1, \ldots, \theta_N)$

parameterize the tuple of localized policies $\zeta_i^{\theta_i}$, with the joint policy as its product $\zeta^\theta(\mathbf{u} \mid \mathbf{x}) = \prod_{i=1}^N \zeta_i^{\theta_i}(u_i \mid x_{N_i})$, as agents act independently.

Note: We do not include the actions of other agents, because all agents may take an action at the same time, so we will not have this information and each agent will act independently.

5.1.2.3 *Symmetrization*

For simplicity of learning, we further assume that any agent may choose to either send to their own queue, or offload uniformly randomly to any of their neighbors, i.e. we consider the actions $\mathcal{U}_i = \mathcal{U} = \{0, 1\}$ for all agents i, of either sending to its own queue (0), or randomly sending to a neighbor (1), see Fig. 5.3 for an example visualization. This assumption symmetrizes the model and allows tractable, regular solutions with theoretical guarantees.

Indeed, to some extent such symmetrization is necessary to obtain a mean-field limit, as otherwise behavior depends on the ordering of neighbors. Consider a simple one-dimensional line graph where nodes are connected in a straight path. If we use a model that is not symmetric, e.g. if all agents send all their packets to the first of their two neighbors under some ordering of their neighbors, then we obtain different behavior depending on this ordering, i.e. which neighbor is considered the first and which is the second. Cutting the graph into sets of three, we could define the center node of each set to be the first neighbor for the other nodes, leading to a packet arrival rate of $2\lambda(t)$ at the center node. On the other hand, if we define first neighbors such that there are no overlapping first neighbors between any agents, then any node will have a packet arrival rate of at most $\lambda(t)$. This shows that a certain symmetrization is natural to obtain a scaling limit.

At most, we could consider a solution that anonymizes but still differentiates between neighbors whenever one is fuller than the other, which may be important especially if e.g. queue serving rates are heterogeneous, and can be analogously handled theoretically. However, in our experiments we found that such an assumption complicates training of a RL policy due to the significant addition of action space complexity. Therefore, trading off between policy expressiveness and RL training stability, we use the aforementioned symmetrization.

5.1.2.4 *Dynamical system model*

The state of the agent is the state of its associated queue, and is affected by the actions of itself and neighboring agents. Therefore, at every epoch, $t \in \mathbb{N}$, given the current global state $\mathbf{x}(t)$ and action $\mathbf{u}(t)$, the next local state $x_i(t+1)$ of the agent i can be calculated independently only using the neighbors' states $\mathbf{x}_{N_i}(t) \equiv (x_j(t))_{j \in N_i}$ and actions $\mathbf{u}_{N_i}(t) \equiv (u_j(t))_{j \in N_i}$, i.e.

$$\mathbb{P}(\mathbf{x}(t+1) \mid \mathbf{x}(t), \mathbf{u}(t)) = \prod_{i=1}^N \mathbb{P}(x_i(t+1) \mid x_{N_i}(t), u_{N_i}(t)) \tag{5.1.1}$$

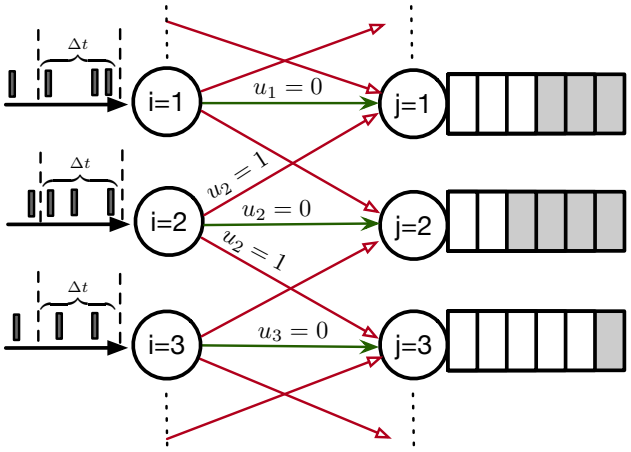

FIGURE 5.3: Visualization of how agents implement their policy. For instance, agent $i = 2$ has neighbors $j \in \{1, 3\}$. If its action is $u_2 = 0$, it will allocate all its arriving jobs to its own queue $j = 2$ (green arrow). In contrast, if $u_2 = 1$ then, the arriving jobs are allocated randomly to one of its neighbor j (red arrows).

where each $\mathbb{P}(x_i(t+1) \mid x_{N_i}(t), u_{N_i}(t))$ can be computed by the Kolmogorov equation for continuous-time Markov chains, given that the rate of an arrival at a queue is given by the sum of arrival rates assigned by the agents, $\lambda_i(t) = \lambda(t)(1 - u_i + \sum_{j \in N_i} \frac{u_j}{|N_j|})$, and the package departure rate is fixed to the serving rate $\mu > 0$.

Each agent is associated with a local stage reward $r_i(x_i, u_i)$ and according global stage reward $r(\mathbf{x}, \mathbf{u}) = \frac{1}{N} \sum_{i=1}^{N} r_i(x_i, u_i)$. The reward is given in terms of a penalty for job drops due to each action u_i. The objective is then to find the localized policy tuple θ such that the global reward is maximized, starting from initial state distribution $x_i(0) \sim \alpha(0)$.

5.2 SPARSE GRAPH MEAN-FIELD QUEUING SYSTEM

In this section, we will consider the limits of large graphs and the behavior in such systems, by using mean-field theory from [142]. In order to obtain a limiting mean-field system, we assume a shared policy $\zeta_i = \zeta$ for all agents i. This assumption is natural, as it often gives state-of-the-art performance in various RL benchmarks [153–155] and also allows immediate application to arbitrary sizes of the finite system, as all schedulers can use the same policy. Furthermore, it will allow scaling up the system at any time by simply adding more schedulers and queues, without retraining a policy.

In contrast to typical mean-field games [50] and mean-field control [59], we cannot reduce the entire system exactly to a single representative agent and its probability law. This is because in a sparse setting, the neighborhood state distribution of any agent remains an empirical distribution of finitely many neighbors and hence cannot follow a law of large numbers into a deterministic mean-field distribution. Therefore, the neighborhood and its state distribution remain stochastic, and it is necessary to model the probability law of entire neighborhoods. The modeling of such graphical neighborhoods is formalized in the following.

5.2.1 *Topological structure*

To make sense of the limiting topology of our system formally, we introduce technical details, letting finite systems be given by sequences of (potentially random) finite graphs and initial states $(G_n, \mathbf{x}_n)$ converging in probability in the local weak sense to some limiting random graph $(G, \mathbf{x})$. Here, we assume that graphs are of bounded degree, i.e. there exists a finite degree d such that all nodes have at most d neighbors. In other words, we define a sequence of systems of increasing size according to a certain topology, which formalizes the scalable architectural choice of a network structure, such as a ring topology or torus.

To define what one understands as convergence in the local weak sense, define first the space of marked rooted graphs $\mathcal{G}_*$, the elements of which essentially constitute a snapshot of the entire system at any point in time. Such a marked rooted graph consists of a tuple $(G, \varnothing, \mathbf{x}) \in \mathcal{G}_*$, where $G = (\mathcal{N}, \mathcal{E})$ is a graph, $\varnothing \in \mathcal{N}$ is a particular node of G (the so-called root node), and $\mathbf{x}_n \in \mathcal{X}^{\mathcal{N}}$ defines states ("marks") for each node in G, i.e. the current queue filling of queues associated to any agent (node). Denote by $B_k(G, \varnothing, \mathbf{z})$ the marked rooted subgraph of vertices in the k-hop neighborhood of the root node $\varnothing$. The space $\mathcal{G}_*$ is metrized such that sequences $(G_n, \varnothing_n, \mathbf{x}_n) \to (G, \varnothing, \mathbf{x}) \in \mathcal{G}_*$ whenever for any $k \in \mathbb{N}$, there exists n' such that for all $n > n'$ there exists a mark-preserving isomorphism $\phi\colon B_k(G_n, \varnothing_n, \mathbf{x}_n) \to B_k(G, \varnothing, \mathbf{x})$, i.e. with $x_{ni} = x_{\phi(i)}$ for all nodes $i \in G_n$ (local convergence).

We will abbreviate elements $(G_n, \varnothing_n, \mathbf{x}_n), (G, \varnothing, \mathbf{x}) \in \mathcal{G}_*$ as $(G_n, \mathbf{x}_n), (G, \mathbf{x})$, and their node sets as G_n, G whenever it is clear from the context. Then, finally, convergence in the local weak sense is formally defined by $\lim_{n\to\infty} \frac{1}{|G_n|} \sum_{i \in G_n} f(C_i(G_n)) = \mathbb{E}[f(G)]$ in probability for every continuous and bounded $f\colon \mathcal{G}_* \to \mathbb{R}$, where $C_i(G_n)$ denotes the connected component of i in G_n. In other words, wherever we randomly look in the graph, there will be little difference between the distribution of the random local system state (including its topology), and of the limiting G. This holds true e.g. initially if we initialize all queues as empty or i.i.d., and consider various somewhat regular topologies, see Section 5.5.1. More details also in [142].

5.2.2 *Localized system model*

As discussed in the prequel, consider sequences of possibly random rooted marked graphs (i.e. finite graphs and initial states) $(G_n, \mathbf{x}_n) \to (G, \mathbf{x})$ with $|G_n|$ agents $i = \{1, \dots, |G_n|\}$, converging in the local weak sense to the potentially infinite-sized system $(G, \mathbf{x})$. For a moment, consider a decentralized, stochastic control policy $\zeta\colon \mathcal{Z} \times \mathcal{T} \to [0, 1]$ such that an agent $i \in G$ chooses to offload to a random neighbor (action 1) with probability $\zeta(t, x_i(t))$, depending on its own queue state $x_i(t)$ only. We can then consider the probability law $\mathcal{L}(G, \mathbf{x})$ of the limiting system $(G, \mathbf{x})$ as the state of a single-agent MDP, similar to the MFC MDP formalism in standard MFC [71]. At least formally, we do so by identifying the choice of $\zeta(t)$ at any time $t \in \mathbb{N}$ as an action, and letting the state of the MFC MDP be given by the law of the time-variant system $(G(t), \mathbf{x}(t))$ resulting from the application of $\zeta(t)$ when starting at some initial law.

Deferring for a moment (until Section 5.4) (i) the question of how to simulate, represent or compute the probability law of a possibly infinite-sized rooted marked graph, and (ii) the detailed partial information structure (i.e. observation inputs) of the following policy, we consider a hierarchical upper-level MFC policy π that, for any current MFC MDP state, assigns to all agents at once such a policy $\zeta(t) = \pi(\mathcal{L}(G(t), \mathbf{x}(t)))$ at time t. To reiterate, the decentralized control policy $\zeta(t)$ at any time t now becomes the action of the MFC MDP, where the dynamics are formally given by the usually infinite-dimensional states $\mathcal{L}(G(t), \mathbf{x}(t))$, i.e. the probabilities of the limiting rooted marked graph $(G(t), \mathbf{x}(t))$ being in a particular state after applying a sequence of policies $(\zeta(0), \ldots, \zeta(t-1))$. The cost function $J^{G,\mathbf{x}}(\pi)$ for any such upper-level policy π is then given by the number of expected packet drops per agent $D(t)$ in the limiting system,

$$J^{G,\mathbf{x}}(\pi) = -\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t D(t)\right].$$

Using analogous definitions for the finite system based on the topologies and initial states $(G_n, \mathbf{z}_n)$, we can apply the upper-level MFC policy π to each agent and use the resulting number of average packet drops $D^N(t)$ at time t. Using our newly introduced graphical formulation, we thus write the cost function $J(\theta)$ in the finite system $(G_n, \mathbf{z}_n)$ as

$$J^{G_n,\mathbf{z}_n}(\theta) = -\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t D^N(t)\right].$$

5.3 THEORETICAL GUARANTEES

One can now show that the performance of the finite system is approximately guaranteed by the performance in the limiting mean-field system. Informally, this means that for any two policies, if the performance of one policy is better in the mean-field system, it will also be better in large finite systems.

Theorem 2. *Consider a sequence of finite graphs and initial states $(G_n, \mathbf{x}_n)$ converging in probability in the local weak sense to some limiting $(G, \mathbf{x})$. For any policy π, as $n \to \infty$, we have convergence of the expected packet drop objective*

$$J^{G_n,\mathbf{x}_n}(\pi) \to J^{G,\mathbf{x}}(\pi).$$

Proof of Theorem 2. We apply the framework of [142], which does not include actions or rewards, by including actions and rewards via separate time steps. Specifically, each time step is split in three, and we define the agent state space $\mathcal{X} := \mathcal{Z} \cup (\mathcal{Z} \times \mathcal{U}) \cup (\mathcal{Z} \times \mathcal{U} \times [0, D_{\max}])$ for each of the agents, indicating at any third decision epoch $t = 0, 3, 6, \ldots$ the state of its queue. After each such epoch, the agent states will contain the state of its queue together with its choice of action at times $t = 1, 4, \ldots$, and lastly also the number of packet drops at times $t = 2, 5, \ldots$. Here, $D_{\max}$ is the maximum expected number of packet drops and is given by the $(d+1)$ times the maximum per-scheduler arrival rate $\lambda_{\max} = \lambda_h$.

We formally rewrite the system defined earlier, using the symbol $\mathbf{x}$ for the state of the rewritten system instead of $\mathbf{z}$. As a result, each agent i is endowed with a random local $\mathcal{X}$-valued state variable $X_i^{G,x}(t)$ at each time t. For the dynamics, we let $S^{\sqcup}(\mathcal{X})$ denote the space of unordered terminating sequences (up to some maximum degree) with the discrete topology as in [142], and define in the following a system dynamics function $F\colon \mathcal{X} \times S^{\sqcup}(\mathcal{X}) \times \Xi \to \mathcal{X}$ returning a new state for any current local state $X_i^{G,x}(t)$, any current $S^{\sqcup}(\mathcal{X})$-valued neighborhood queue fillings $X_{N_i}^{G,x}(t)$ and i.i.d. sampled Ξ-valued noise $\xi_i(t+1)$. More precisely, we use

$$
\begin{aligned}
X_i^{G,x}(t+1) &\coloneqq F[X_i^{G,x}(t), X_{N_i}^{G,x}(t), \xi_i(t+1)] \\
&= \begin{cases}
(X_i^{G,x}(t), \xi_{i, X_i^{G,x}(t)}(t+1)), & \text{if} \quad X_i^{G,x}(t) \in \mathcal{Z}, \\
(X_{i,1}^{G,x}(t), X_{i,2}^{G,x}(t), \xi_{i, X_{i,1}^{G,x}(t), X_{i,2}^{G,x}(t), X_{N_i}^{G,x}(t)}(t+1)) & \text{if} \quad X_i^{G,x}(t) \in \mathcal{Z} \times \mathcal{U}, \\
\xi_{i, X_i^{G,x}(t), X_{N_i}^{G,x}(t)}(t+1), & \text{if} \quad X_i^{G,x}(t) \in \mathcal{Z} \times \mathcal{U} \times [0, D_{\max}],
\end{cases}
\end{aligned}
$$

where we define Ξ and the random noise variable ξ as a contingency over all transitions, through a finite tuple of random variables ξ_i consisting of components for (i) randomly sampling the next action $(\xi_{i,z})_{z \in \mathcal{Z}}$ according to π, (ii) computing the expected lost packets $(\xi_{i,x_1,x_2,\beta})_{(x_1,x_2) \in \mathcal{Z} \times \mathcal{U}, \beta \in S^{\sqcup}(\mathcal{X})}$, and (iii) sampling the next queue state $(\xi_{i,x,\beta})_{x \in \mathcal{Z}, \beta \in S^{\sqcup}(\mathcal{X})}$.

For the first step, fixing any π, we let

$$\xi_{i,z} \sim \operatorname{Bern}(\pi_{\lfloor t/3 \rfloor}(z)),$$

for all $z \in \mathcal{Z}$, to sample action from π. Deferring for a moment the second step, for the new state in the third step, we use the Kolmogorov forward equation for the queue state during the epoch for any initial (x, β):

$$P(x, \beta) = \operatorname{Exp} \mathbf{Q}(x, \beta) \cdot \mathbf{e}_{x_1} \in \Delta^{\mathcal{Z}}$$

with unit vector $\mathbf{e}_{x_1}$, which results from Poisson thinning, as the equivalent arrival rate at a queue of an agent currently in state x with neighborhood β will be given by $\lambda\left(1 - x_2 + \frac{\beta(\mathbf{1}_{\mathcal{Z} \times \{1\} \times S^{\sqcup}(\mathcal{X})})}{d}\right)$. Thus, we have the transition rate matrix $\mathbf{Q}(x, \beta)$ with $\mathbf{Q}_{i,i-1} = \lambda\left(1 - x_2 + \frac{\beta(\mathbf{1}_{\mathcal{Z} \times \{1\} \times S^{\sqcup}(\mathcal{X})})}{d}\right)$, $\mathbf{Q}_{i-1,i} = \mu$ for $i = 1, \ldots, B$, and where $\mathbf{Q}_{i,i} = -\sum_j \mathbf{Q}(\lambda, z)_{j,i}$ for $i = 0, \ldots, B$, and zero otherwise. Therefore, the next queue filling is sampled as

$$\xi_{i,x,\beta} \sim \operatorname{Cat}(P(x, \beta)).$$

Lastly, for the second step we consider the non-random conditional expectation of packet losses during any epoch:

$$\xi_{i,x_1,x_2,\beta} = [\operatorname{Exp} \bar{\mathbf{Q}}(x, \beta) \cdot \mathbf{e}_{x_1}]_{B+1}$$

under the executed actions (allocations), analogously to the new state, by adding another row to the transition matrix $\mathbf{Q}(x, \beta)$, giving $\bar{\mathbf{Q}}(x, \beta)$ where $\bar{\mathbf{Q}}_{B+1,B} = \lambda\left(1 - x_2 + \frac{\beta(\mathbf{1}_{\mathcal{Z} \times \{1\}})}{d}\right)$, i.e. counting all the expected packet arrivals whenever the queue is already full (B).

We use the same definitions for finite systems $(G_n, \mathbf{z}_n)$. We can verify [142, Assumption A], since F is continuous e.g. by discreteness of the relevant spaces, and all ξ_i are sampled independently and identically over agents and times. By [142, Theorem 3.6], the empirical distribution $\beta^{G_n,\mathbf{z}_n} \coloneqq \frac{1}{|G_n|}\sum_{i\in G_n}\delta_{X_i^{G_n,\mathbf{z}_n}(t)}$ converges in probability in $\mathcal{P}(\mathcal{X}^\infty)$ to its mean-field limit $\mathcal{L}(X_\varnothing^{G,x})$, and in particular

$$\beta^{G_n,\mathbf{z}_n}(t) \to \mathcal{L}(X_\varnothing^{G,x}(t)) \quad \text{in distribution in } \mathcal{P}(\mathcal{X}) \tag{5.3.2}$$

at any time t, where $\mathcal{P}(\cdot)$ denotes the space of probability measures equipped with the topology of weak convergence. Hence, the above describes the original objective by

$$\begin{aligned} J^{G_n,\mathbf{z}_n}(\pi) &= \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t D_N(t)\right] \\ &= \sum_{t=0}^{\infty}\sqrt[3]{\gamma}^{(t-2)}\mathbb{E}\left[r^\infty(\beta^{G_n,\mathbf{z}_n}(t))\right] \\ &\to \sum_{t=0}^{\infty}\sqrt[3]{\gamma}^{(t-2)}\mathbb{E}\left[r^\infty(\mathcal{L}(X_\varnothing^{G,x}(t)))\right] = J^{G,\mathbf{z}}(\pi) \end{aligned}$$

since we split any time step into three by the prequel, the continuous mapping theorem, and dominated convergence, where we use the continuous function $r^\infty(\beta) = \int \mathbf{x}_3 1_{x\in\mathcal{Z}\times\mathcal{U}\times[0,D_{\max}]}\beta(\mathrm{d}\mathbf{x})$, where $\beta = \mathcal{P}(\mathcal{X})$, to sum up the expected packet drops in the system, since the integrand is continuous and bounded (under the sum topology for the union in $\mathcal{X}$, and the product topology for products in $\mathcal{X}$). □

As a result, we have obtained a limiting mean-field system for large systems, which may be more tractable for finding improved load balancing schemes. In particular, if we have a number of policies between which to choose, then the policy performing best in the MFC system will also perform best in sufficiently large systems. Here, the set of MFC policies can include well-known algorithms such as JSQ, which is known to be optimal in many special cases.

Corollary 1. *Consider a finite set $\{\pi_1,\dots,\pi_K\}$ of MFC policies with MFC objective values $J^{G,\mathbf{x}}(\pi_1),\dots,J^{G,\mathbf{x}}(\pi_K)$. Let π_i be the policy with maximal MFC objective value $J^{G,\mathbf{x}}(\pi_i) \geq \max_j J^{G,\mathbf{x}}(\pi_j)$. Then, there exists n' such that for all $n > n'$, we also have optimality in the finite system*

$$J^{G_n,\mathbf{x}_n}(\pi_i) \geq \max_j J^{G_n,\mathbf{x}_n}(\pi_j).$$

Proof of Corollary 1. Define the optimality gap:

$$\Delta J \coloneqq J^{G,\mathbf{z}}(\pi_i) - \max_{j\neq i} J^{G,\mathbf{z}}(\pi_j).$$

Then, by Theorem 2, there exists n such that:

$$\max_i \left|J^{G_n,\mathbf{z}_n}(\pi_i) - J^{G,\mathbf{z}}(\pi_i)\right| \leq \frac{\Delta J}{2}.$$

Therefore,

$$
\begin{aligned}
&J^{G_n,\mathbf{z}_n}(\pi_i) - \max_j J^{G_n,\mathbf{z}_n}(\pi_j) \\
&= \min(J^{G_n,\mathbf{z}_n}(\pi_i) - J^{G_n,\mathbf{z}_n}(\pi_j)) \\
&= \min(J^{G_n,\mathbf{z}_n}(\pi_i) - J^{G,\mathbf{z}}(\pi_i) + J^{G,\mathbf{z}}(\pi_i) - J^{G,\mathbf{z}}(\pi_j) + J^{G,\mathbf{z}}(\pi_j) - J^{G_n,\mathbf{z}_n}(\pi_j)) \\
&\geq \min_{j\neq i}(J^{G_n,\mathbf{z}_n}(\pi_i) - J^{G,\mathbf{z}}(\pi_i)) + \min_{j\neq i}(J^{G,\mathbf{z}}(\pi_i) - J^{G,\mathbf{z}}(\pi_j)) + \min_{j\neq i}(J^{G,\mathbf{z}}(\pi_j) - J^{G_n,\mathbf{z}_n}(\pi_j)) \\
&\geq -\frac{\Delta J}{2} + \Delta J - \frac{\Delta J}{2} \\
&= 0
\end{aligned}
$$

which is the desired conclusion. □

These proofs use the theoretical framework of [142] for general dynamical systems.

In our experiments, we will also allow for randomized assignments per packet to further improve performance empirically, such that all packets arriving at a particular scheduler during the entirety of any epoch are independently randomly either allocated to the local queue or a random neighbor, i.e. formally we replace offloading choices $\mathcal{U} = \{0, 1\}$ by probabilities for offloading each packet independently, $\mathcal{U} = [0, 1]$.

The MFC MDP formulation and theoretical guarantees give us the opportunity to use MFC together with single-agent control, such as RL in order to find good scalable solutions while circumventing hard exact analysis and improving over powerful techniques such as JSQ in large queuing systems. All that remains is to solve the limiting MDP.

5.4 LEARNING OPTIMAL LOAD BALANCING POLICY FOR THE LIMITING SYSTEM

Building upon the preceding MFC formulation, all that remains to find optimal load balancing in large sparse graphs, is to solve the MFC problem. Due to its complexity, the limiting MFC problem will be solved by considering it as a variant of an MDP, i.e. a standard formulation for single-agent centralized RL, which will also allow a model-free design methodology. More precisely, we will consider partially-observed MDP (POMDP) variant of the problem, since at any time t, we cannot evaluate the potentially infinite system $\mathcal{L}(G(t), \mathbf{x}(t))$ exactly to obtain action $\zeta(t) = \pi(\mathcal{L}(G(t), \mathbf{x}(t)))$. Instead, we will use the empirical distribution $\beta^{G_n,\mathbf{x}_n}(t)$ as an observation that is only correlated with the state of the entire system, but of significantly lower dimensionality ($|\mathcal{X}|$-dimensional vector, instead of $\mathcal{X}^{|G|}$ plus additional topological information). This also means that we need not consider the limiting system of potentially infinite size, or include the information of root nodes when considering network topologies in the following, which is intuitive as there is no notion of global root in local queuing systems.

Here, the centralized RL controller could have estimated or exact global information on the statistics of the queue states of all nodes, or alternatively we can understand the approach as an optimal open-loop solution for any given known starting state, since the limiting MFC

dynamics on $\beta^{G_n,\mathbf{x}_n}(t)$ are deterministic, and therefore the centralized RL controller can be used to compute an optimal deterministic sequence of control, which can then be applied locally.

input : Hyperparameters and system parameters from Table 5.1 and Table 5.2.
output : Policy π^θ

Initialize finite system, i.e. rates $\lambda(0) \sim \mathrm{Unif}(\{\lambda_h, \lambda_l\})$ and queue states $\mathbf{x}_n = \mathbf{0} \in \mathcal{Z}^{G_n}$ on topology G_n.

Initialize PPO policy π^θ, critic V^ψ.

for *PPO iteration* $n = 0, 1, \ldots$ **do**
  Initialize batch buffer $B = \emptyset$
  while $|B| < B_b$ **do**
    for $t = 0, 1, \ldots, T_\mathrm{e}$ **do**
      Observe empirical state distribution $\beta^{G_n,\mathbf{x}_n}(t)$.
      Sample decision rule $\zeta(t) \sim \pi^\theta(\beta^{G_n,\mathbf{x}_n}(t))$.
      for $i = 1, \ldots, N$ **do**
        Observe agent state $x_i(t)$.
        Sample action $u_i(t) \sim \zeta(t, x_i(t))$ (*allocation rule*).
      end
      Execute allocation rules $(u_i(t))_i$ for Δt time units.
      Resample arrival rates $\lambda(t+1) \sim \mathbb{P}(\lambda(t+1) \mid \lambda(t))$.
      Count number of dropped packets per agent $D(t)$.
      Observe new distribution $\beta^{G_n,\mathbf{x}_n}(t)$.
      Save (state-action-reward-state) transition
        $B = B \cup \{(\beta^{G_n,\mathbf{x}_n}(t), \zeta(t), -D(t), \beta^{G_n,\mathbf{x}_n}(t+1))\}$
    end
  end
  Compute GAE advantages $\hat{A}$ on B.
  for *epoch* $i = 1, \ldots, T_b$ **do**
    Sample mini-batch b with $|b| = B_m$ from B.
    Update policy parameter θ using PPO loss gradient $\nabla_\theta L_\theta$ on b.
    Update critic parameter ψ using L_2 loss gradient $\nabla_\psi L_\psi$ on b.
  end
end
return π^θ

FIGURE 5.4: Learning MFC policies in finite systems.

In our experiments we also allow to simply insert the empirical distribution of the locally observed neighbor queue states at each node (a simple estimate of the true empirical distribution), i.e. by changing lines 8-13 in Algorithm 5.4 to instead sample decision rules for each agent according to the local empirical state distribution, which is verified to be successful. Thus, our approach leans into the centralized training decentralized execution (CTDE) scheme [48] and learns a centralized policy, which can then be executed in a

decentralized manner among all schedulers. As desired, our approach is applicable to localized queuing systems.

5.4.1 *Training on a finite queuing system*

Note that the considered observation $\beta^{G_n,\mathbf{x}_n}(t)$ and any other variables such as the number of dropped packets at the root node can indeed be computed without evaluating an infinite system until any finite time t, since at any time t, at most any node less than t steps away from the root node may have had an effect on the root node state. Therefore, the computation of root node marginals can be performed exactly until any finite time t, even if the limiting system consists of an infinitely large graph G. However, the cost of such an approach would still be exponential in the number of time steps, as a k-hop neighborhood would typically include exponentially many nodes, except for very simple graphs with degree 2.

We therefore consider alternatives: For one, we could apply a sequential Monte Carlo approach to the problem by simulating M instances of a system from times 0 to some terminal time T that consists of all nodes less than T away from the root node. However, this means that we would have to simulate many finite systems in parallel. Instead, using the fact that the empirical distribution of agent states $\beta^{G_n,\mathbf{x}_n}(t)$ in the finite system converges to $\mathcal{L}(X_{\varnothing}^{G,x}(t))$ as seen in Theorem 2, it should be sufficient to evaluate $\mathcal{L}(X_{\varnothing}^{G,x}(t))$ via the empirical distribution of a sufficiently large system.

Thus, we simulate only a single instance of a large system with many nodes by using it for the limiting MFC, which is equivalent to learning directly on a large finite system. In other words, our approach learns load balancing strategies on a finite system by using the MFC formalism for tractability of state and action representations, with theoretical guarantees.

5.4.2 *Implementation*

In order to solve the POMDP, we apply the established proximal policy optimization (PPO) RL method [153, 156] with and without recurrent policies, as commonly and successfully used in POMDP problems [157]. PPO is a policy gradient method with a clipping term in the loss function, such that the policy does not take gradient steps that are too large while learning [153, 156]. For our experiments, we have worked with the stable and easy-to-use RLlib 1.10 implementation [137] of PPO. The overall training code is given in Algorithm 5.4, which also shows how to analogously apply a trained MFC policy to a finite system. We use diagonal Gaussian neural network policies with $\tanh$-activations and two 256-node hidden layers, parameterizing MFC MDP actions $\zeta(t)$ by values in $[0, 1]$ for each entry $\zeta(t, u \mid x)$, normalized by dividing by the sums $\sum_u \zeta(t, u \mid x)$.

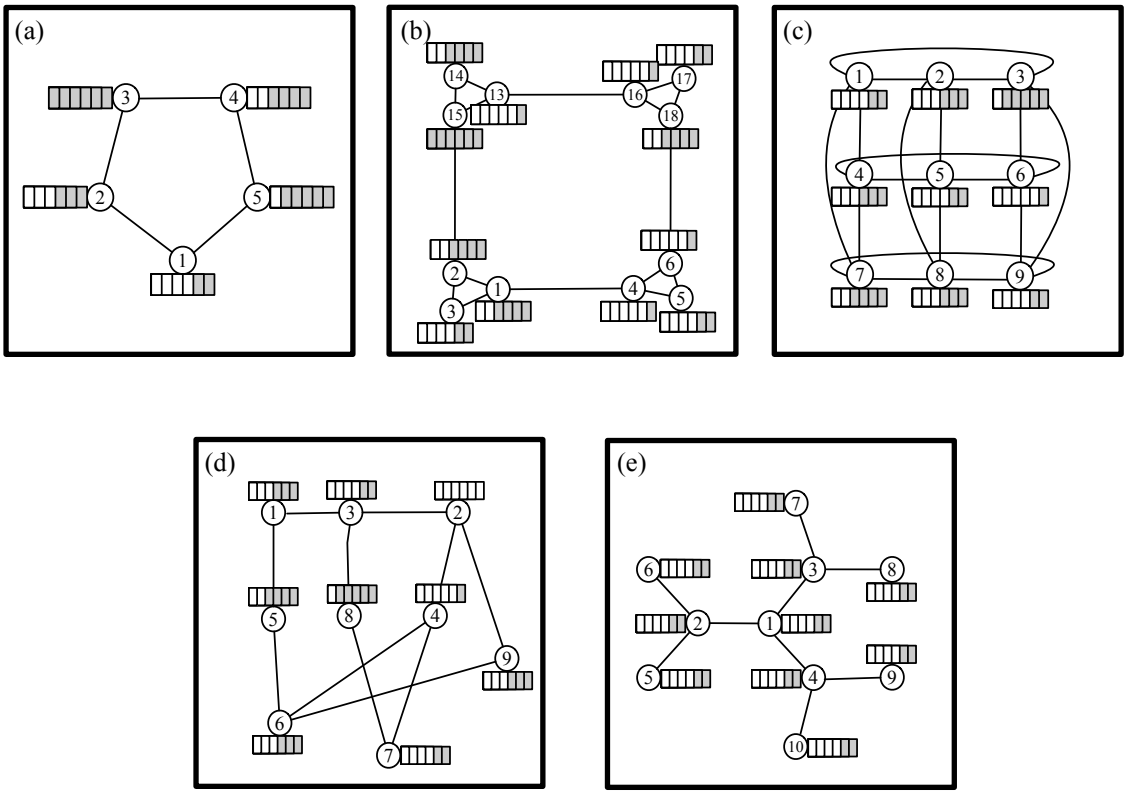

FIGURE 5.5: Small scale illustration of all the topologies used in this chapter. (a) shows a 1-D cyclic graph for $N = M = 5$. (b) shows one surface of the cube-connected cycle graph of order $o_c = 3$ and $N = M = 24$, adapted from Fig. 2 of [158]. (c) is the 2-D torus grid graph of $o_t = 3$ rows and columns, having $N = M = 9$. (d) shows a configuration model of sequence $\mathcal{D} = \{2, 3\}$ for $N = M = 9$. And (e) is the Bethe lattice with depth $o_b = 2$, degree $d = 3$ and $N = M = 10$.

5.5 EVALUATION

In this section, we first give an explanation of the different types of graph topologies we have used to verify our aforementioned theoretical analysis. We also give a description of the different load balancing algorithms we have chosen to compare against, as well as discuss hyperparameters used for training and evaluation.

5.5.1 *Topologies*

A brief description of the practical topologies of interest, most of which fulfill the convergence in the local weak sense defined earlier, is given here. For each of the following mentioned topologies, the agents are numbered from 1 to N.

First, we consider the simple 1-D **cyclic** (CYC-1D) graph, which has extensively been used in the study of queuing networks and is highly local [159, 160]. Each agent i has access to $d = 2$ other queues/servers, $\{i - 1, i + 1\}$, while the edge nodes, 1 and N, form a connection, as highlighted in Fig. 5.5.

Next, we define the **cube-connected cycle** (CCC) graph. This undirected cubic graph has been considered as a network topology for parallel computing architectures [158, 161]. It is characterized by the cycle order, o_c which is the degree d of each node and defines the total number of nodes $N = o_c 2^{o_c}$ in the graph, as shown in Fig. 5.5.

We also apply the **torus** (TORUS) grid graph that has been repeatedly used to represent distributed systems for parallel computing [162, 163], as a higher-dimensional extension of the CYC-1D graph. We here consider a 2-D torus, which is a rectangular lattice having o_t rows and columns. Every node connects to its $d = 4$ nearest neighbors, and the corresponding edge nodes are also connected. The total nodes in a 2-D torus are $N = o_t^2$, see Fig. 5.5.

Another highly general topology is the **configuration model** (CM). This sophisticated generalized random graph is one of the most important theoretical models in the study of large networks [164, 165]. In contrast to the prior highly local topologies, the CM can capture realistic degree distributions under little clustering. Here every agent is assigned a certain degree, making the graph heterogeneous as compared to every agent having the same degree as in previously mentioned graphs. The degree sequence we have used is in the set $\mathcal{D} = \{2, 3\}$ with equal likelihood, as shown in Fig. 5.5.

And lastly, for an ablation study on graphical convergence assumptions, we use the **Bethe** (BETHE) lattice. This cycle-free regular tree graph is used for analysis of many statistical physics, mathematics related models and potential games [166, 167]. It is characterized by a pre-defined lattice depth o_b, with all nodes in the lattice having the same fixed number of neighbors, d. The number of nodes at a depth o away from the root node are given by: $d(d-1)^{(o_b-1)}$ and the total nodes in the Bethe lattice are calculated as: $N = 1 + \sum_{o=1}^{o_b} d(d-1)^{(o-1)}$. We use $d = 3$ for our experiments, as in Fig. 5.5. Note that it has already been formally shown in [142, Sections 3.6 and 7.3] that for a sequence of increasing regular trees, the empirical measure does not converge to the same limit law as the root particle, even in the weak sense. This is because even as $N \to \infty$, a large proportion of the nodes are leaves, which greatly influences the behavior of the empirical measure, as particles at different heights behave differently, and the root particle is the only particle at height zero. We have also verified this mismatch using our experiments in Section 5.5.4, though we nevertheless obtain improved performance in certain regimes.

5.5.2 *Load Balancing Algorithms*

We now explain first the mean-field solver that we have designed, based on our mathematical modeling from Section 5.1, namely the MF-Random solver. Then we mention all the existing state-of-the-art finite-agent solvers which we have used for performance comparison. The different kinds of load balancing policies that we have verified on the above-mentioned topologies are given in the following:

- MF-Random (MF-R): The agent is only aware of the state of its own queue. The upper-level learned policy is a vector that gives the probability of sending jobs to their own queue or to a random other accessible queue.

TABLE 5.1: System parameters used in the experiments.

Symbol	Name	Value
Δt	Synchronization delay [ms]	$1-10$
μ	Service rate [1/ms]	1
(λ_h, λ_l)	Arrival rates [1/ms]	$(0.9, 0.6)$
N	Number of agents	$4-6000$
M	Number of queues	$4-6000$
d	Number of accessible queues	$3-6$
I	Monte Carlo simulations	100
B	Queue buffer size	5
$\mathbf{z}_n$	Queue (agent) starting state	$\mathbf{0} \in \mathcal{Z}^{G_n}$
$\nu(0)$	Queue starting state distribution	$[1, 0, 0, \ldots]$
D	Drop penalty per job	1
T	Training episode length	50
G_n, G	Graph topologies	Section 5.5.1

- MARL-PS (NA-PS-RND): The policy is trained, using PPO with parameter sharing [154], as it often gives state-of-the-art performance in various RL benchmarks [153, 155], on a smaller number of agents. The learned policy can be used for any arbitrary number of agents. It generates a continuous policy which gives the probability of either sending to your queue or randomly to one of the neighbors' queues while only observing the state of your own queue, similar to MF-R.
- Join-the-shortest-queue (JSQ): A discrete policy that sends jobs to the shortest out of all accessible queues.
- Random (RND): A policy where the probability of sending the job to any accessible queues is equally likely.
- Send-to-own-queue (OWN): A discrete policy where the agent only sends jobs to its own queue.
- scalable-actor-critic (SAC) [140]: This method learns a discrete policy of sending to any one of the d accessible queues. Each agent learns an individual policy while using as an observation the agent's own queue state and also the queue states of all its neighbors. However, a trained policy cannot be used for an arbitrary number of agents since the policy of an agent is influenced by its neighbors states as well, which is not assumed to be the same for every agent.

Note that the action from the above-mentioned load balancing policies is obtained at the beginning of each Δt and then used for that entire Δt timestep. Also note that for all the algorithms, the value for the number of accessible queues d is dependent on the type of graph topology being used, and we refer to Section 5.5.1 for more details on this. In the next section, we discuss all the required parameters and chosen values for our experiments.

TABLE 5.2: Hyperparameter configuration for PPO.

Symbol	Name	Value
γ	Discount factor	0.99
λ_{RL}	GAE lambda	1
κ_c	KL coefficient	0.2
κ_t	KL target	0.01
ϵ	Clip parameter	0.3
l_r	Learning rate	0.00005
B_b	Training batch size	$4000 - 24000$
B_m	SGD Mini batch size	$128 - 4000$
I_m	Number of SGD iterations	$5 - 8$
T_b	Number of epochs	50

5.5.3 *Training*

For all our experiments, we consider that each agent is associated with only one queue, so $N = M$. The servers work at an exponential rate of μ. At every decision epoch we simulate a Markov modulated arrival rate $\lambda(t)$, with $\lambda(0) \sim \text{Unif}(\{\lambda_h, \lambda_l\})$ with transition law for switching between rates,

$$\begin{aligned} &\mathbb{P}(\lambda(t+1) = \lambda_l \mid \lambda(t) = \lambda_h) = 0.2, \\ &\mathbb{P}(\lambda(t+1) = \lambda_h \mid \lambda(t) = \lambda_l) = 0.5. \end{aligned} \tag{5.5.3}$$

Note that these values were chosen to depict the switching of the system between high and low traffic regimes. In principle, any reasonable values can be considered. The rest of the system parameters are given in Table 5.1. We use the well-established policy gradient based RL algorithm, proximal policy optimization (PPO) as discussed in Section 5.4 with hyperparameters in Table 5.2. A detailed description of what these parameters do is given in Appendix C. We have used a localized reward function which penalizes the drops in each queue.

Lastly, the number of agents N and degrees d in different graph topologies are fixed during training of the MF-R policies as:

- 1-D cyclic (CYC-1D): $N = M = 101$, $d = 2$,
- Cube-connected cycle (CCC): $o_c = 5$, $N = 125$, $d = 3$,
- 2-D Torus grid (TORUS): $o_t = 11$, $N = 111$, $d = 4$,
- Configuration model (CM): $N = 101$, $d \in \{2, 3\}$,
- Bethe lattice (BETHE): $o_b = 5$, $N = 94$, $d = 3$.

Once trained on these parameters, the learned policy can be evaluated on varying graph sizes without the need of retraining, as done in our experiments.

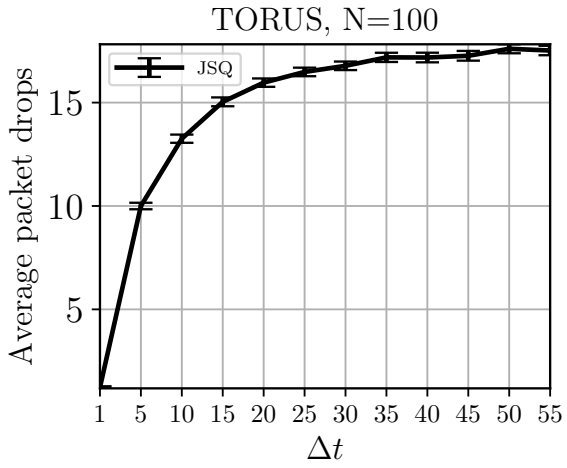

FIGURE 5.6: JSQ converges as Δt increases, validating our simulator.

5.5.4 *Experiment Results*

We now present the performance comparison of the load balancing policies of Section 5.5.2 on graph topologies from Section 5.5.1. For the exact simulation of associated continuous-time Markov chains y, we sample exponential waiting times of all events using the Gillespie algorithm [138]. For training, we use the same simulation horizon for each episode consisting of $T = 50$ discrete decision epochs, and for comparability the performance is evaluated in terms of the average number of packets dropped per 50 time units and queue. However, note that simulating the same time span with different Δt does provide slightly different results, due to the switching between high and low traffic regimes after each Δt epoch. Each evaluation was repeated for 100 simulated episodes, and error bars in all figures depict the 95% confidence interval.

Simulator analysis

Firstly, we performed a small experiment in order to ensure that our simulator works as expected. Performance of the JSQ algorithm was evaluated for the TORUS graph with $o_t = 100$, $N = 100$ and $d = 4$ for the JSQ algorithm. It can be seen in Fig. 5.6 that on increasing Δt there comes a point when the performance of JSQ starts to converge, which is the expected behavior for finite systems.

We also tried different sizes of the neural network while keeping all other parameters and environment. The training was performed for the CCC graph with $o_c = 5$ and $\Delta t = 5$. The tested network sizes are shown in the legend of Fig. 5.7, where the first value is the size of the layer $(128, 256)$ and the second value is the number of layers $(2, 3)$ used. The performance was quite similar for all of them, and we used the default network size $[256 - 2]$ for all our experiments.

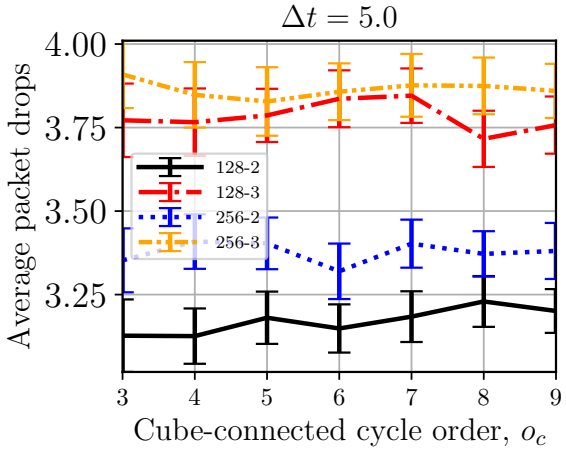

FIGURE 5.7: Performance evaluation on a fixed environment while changing only the neural network parameters.

Comparison to SAC

To begin, while training for SAC, we observed that on increasing the number of agents, the convergence to a locally optimal policy takes longer (using one core of Intel Xeon Gold 6130), making it not too feasible for the larger setups we consider in this chapter. See Fig. 5.8(a) for time taken to learn a SAC policy on a 1-D cyclic topology with $\Delta t = 3$, $N = M$, and same computational resources for all training setups of N. Our implementation of SAC was adapted from [168]. Furthermore, Fig. 5.8(b) shows the performance of the learned SAC policy as compared to other algorithms. Although performance did improve as the number of agents rises, indicating the scalability of SAC to many agents, the performance of SAC remained suboptimal. Due to these limitations, we did not investigate the SAC algorithm further in our experiments.

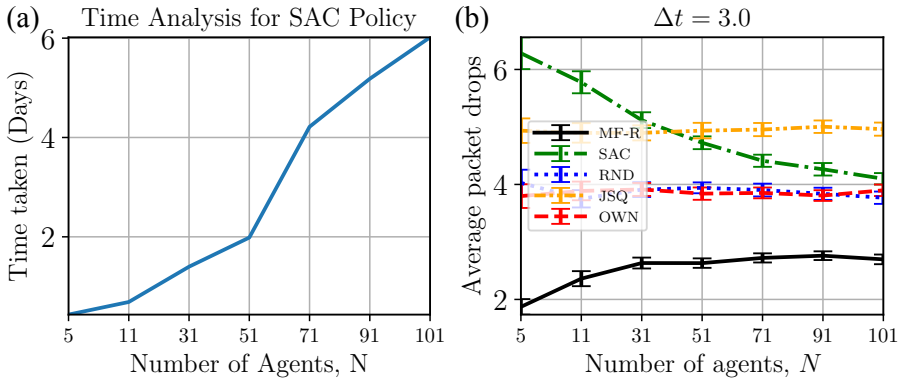

FIGURE 5.8: (a): SAC training time to convergence increases with the number of agents, making it difficult to train and use for larger setups. (b): SAC performs worse than our proposed MF-R policy for a number of agents between 5 and 101.

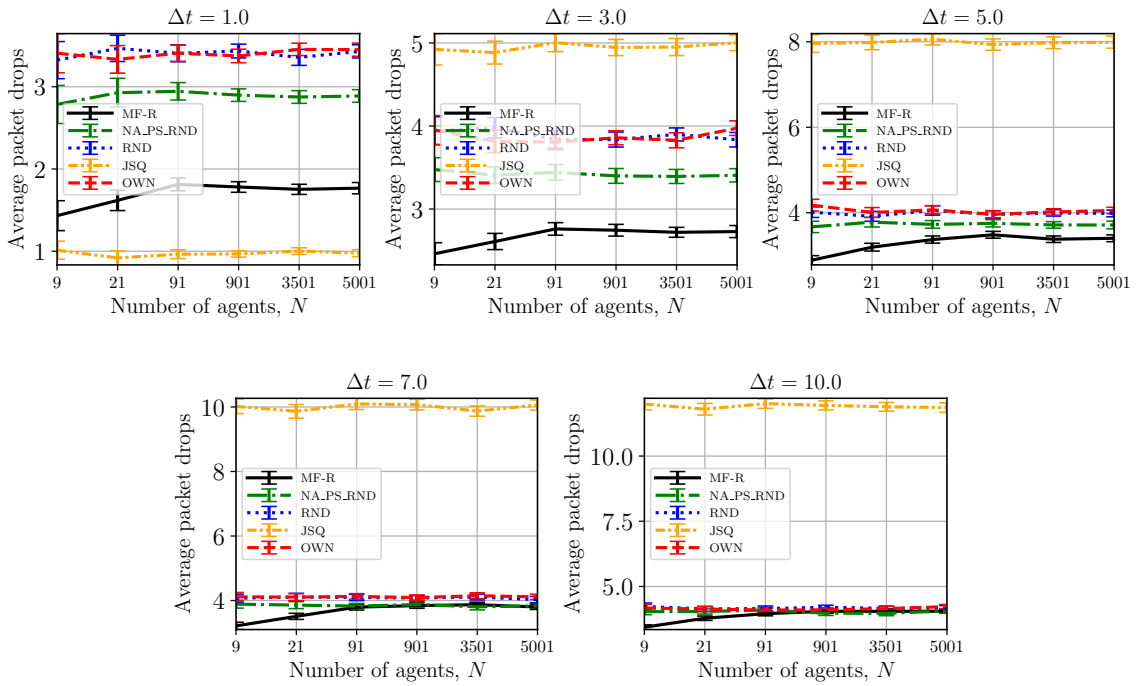

FIGURE 5.9: Performance comparison of the learned MF-R policy to NA-PS-RND, JSQ, RND and OWN algorithms, on a 1-D cyclic graph, over a range of synchronization delays is shown, with 95% confidence intervals depicted by error bars. The degree of each agent is $d = 2$ and the number of agents (queues) used to make the graph are $N \in \{9, 21, 91, 901, 3501, 5001\}$. It can be seen that for the in-between range of Δt our MF-R policy is the optimal one.

Finite system performance

The second result of interest is the performance analysis on the 1-D cyclic graph over a range of Δt for the learned MF-R policy as compared to the NA-PS-RND, JSQ, RND and OWN algorithms, see Fig. 5.9. The number of agents (queues) used to generate the graph are $N \in \{9, 21, 91, 901, 3501, 5001\}$, with every agent always having the fixed degree $d = 2$. For this graph, we also trained an NA-PS-RND policy on a sufficiently small size with $N = M = 5$ to obtain convergence even with batch and minibatch sizes of 50000 and 8000. We used PPO with parameter sharing so that the learned policy could be evaluated on any graph size.

It can be seen that JSQ is the best strategy at $\Delta t = 1$, while the performance of MF-R is very close to JSQ. And as expected, the performance of JSQ deteriorates with increasing synchronization delay Δt. For an intermediate range of $\Delta t \in \{3, 5, 7\}$, neither JSQ nor RND are the optimal load balancing policies. Here, our learned MF-R policy performs best and approaches the optimal RND performance as Δt increases to 10. As discussed earlier, it has already been proven in certain scenarios that as $\Delta t \to \infty$, RND is the optimal load balancing policy [113]. The learned NA-PS-RND policy has a similar trend to our MF-R policy, but is unable to outperform it even for small systems. For the CYC-1D, CCC, and TORUS graphs with homogeneous degrees, the performance of OWN in the limit – sending only to its own queue – is equivalent to RND, since for homogeneous degrees, sending to your own queue results in the same packet arrival rate at each queue as randomly sending

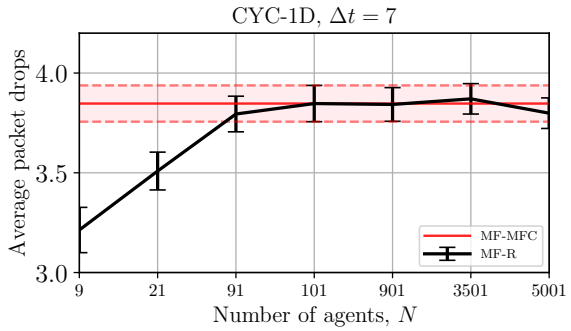

FIGURE 5.10: Performance of the MF-R policy for increasingly large CYC-1D graphs. The red horizontal line indicates the evaluated episode return of the learned MF-R policy during training on $N = 101$, (MF-MFC). Shaded regions depict the 95% confidence intervals. As the system size increases, the performance of the learned policy (black) increasingly converges to the mean-field system performance (red), validating the accuracy of our mean-field formulation and choice of N for training.

to any queue in your neighborhood. In Fig. 5.10, it can also be seen that as the size of the graph increases, the performance of the policy converges and approaches the learned MF-R policy, confirming our theoretical mean-field analysis. The convergence also confirms that the N chosen for training was large enough to represent the mean-field system.

Performance in large systems

In Fig. 5.11 each subfigure shows results for a different topology. We show the performance over a range of $\Delta t \in \{1, 2, \ldots, 10\}$ for large sparse networks. Fig. 5.11(a) is a compact representation of Fig. 5.9 over all Δt and $N = 5001$. Fig. 5.11(b) is for the CCC graph with $d = 3$ and $N = 4608$. Fig. 5.11(c) shows the performance for the TORUS graph with $d = 4$, $o_t = 70$ and $N = 4900$, while Fig. 5.11(d) is for the CM graph for $N = 5001$ with uniformly distributed degrees in the set $d \in \{2, 3\}$. Our learned MF-R policy is compared to JSQ, RND and OWN algorithms. The following observations can be made for all the subfigures: (i) For small $\Delta t = 1$ JSQ performs better than all the algorithms which is theoretically guaranteed, however performance of MF-R is very close to it; (ii) For in between range of $\Delta t = \{2, 3, \ldots, 7\}$ MF-R has the best performance in terms of packets dropped; (iii) Except in the CM graph (where the varying degrees lead to differing packet arrival rates between OWN and RND), we find that OWN and RND again have equivalent performance by regularity of the graph, as discussed in previously; (iv) For higher $\Delta t \in \{8, 9, 10\}$ our MF-R policy performance coincides with the RND policy, where the RND policy has been theoretically proven to be optimal as the synchronization delay $\Delta t \to \infty$ [113]. Hence, we find that our MF-R policy performs consistently well in various topologies.

Ablation on topological assumptions

As mentioned in Section 5.5.1, the theoretical analysis and work in [142] does not apply to the BETHE topology, and the effect is further supported by our experimental results.

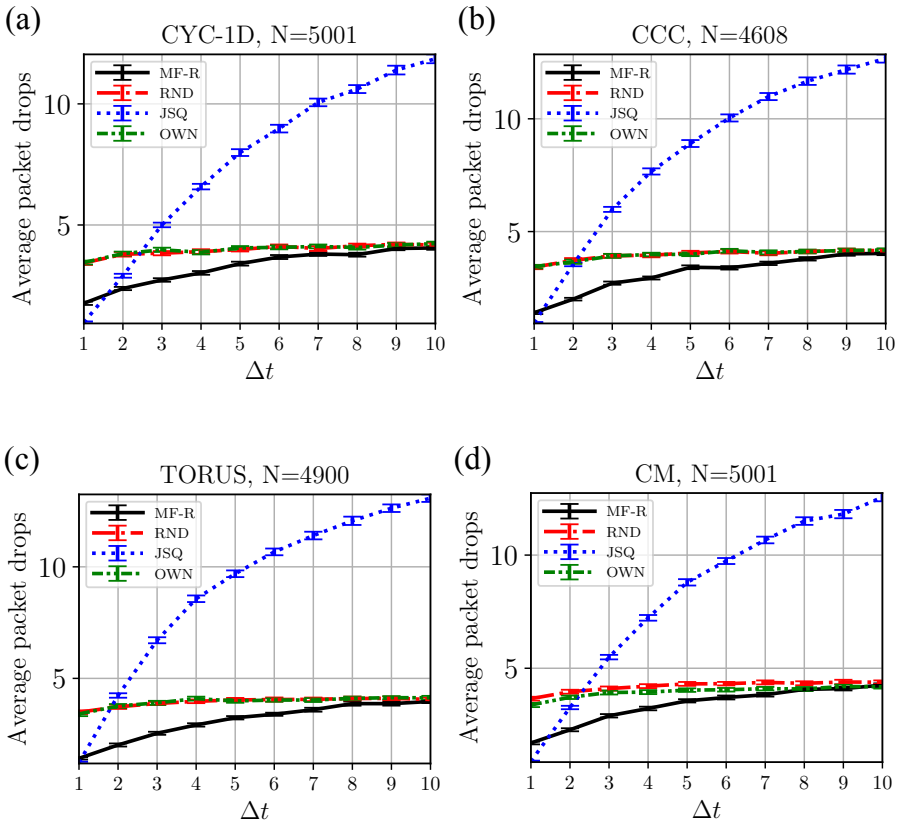

FIGURE 5.11: Performance over $\Delta t \in \{1, 2, \ldots, 10\}$ for large sparse graphs with underlying topologies of CYC-1D in (a), CCC in (b), TORUS in (c) and CM in (d). The number of nodes used to generate the graphs is given at the top of each subfigure. It can be seen that for smaller synchronization delays, JSQ is the optimal policy, while for higher delays RND is the optimal one as expected from prior literature [113]. For the in-between range of delays, our MF-R policy performs the best and approaches the optimal performance of RND as the Δt increases. OWN and RND have equivalent performance in CYC, CCC and TORUS, since fixed degrees imply the same packet arrival rate when always sending to your own queue, or always sending randomly to any neighbor.

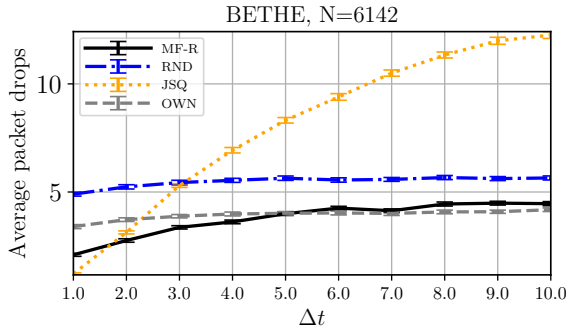

FIGURE 5.12: Performance comparison on large-sized Bethe lattice graph. The MF-R can be worse than RND due to violation of modeling assumptions.

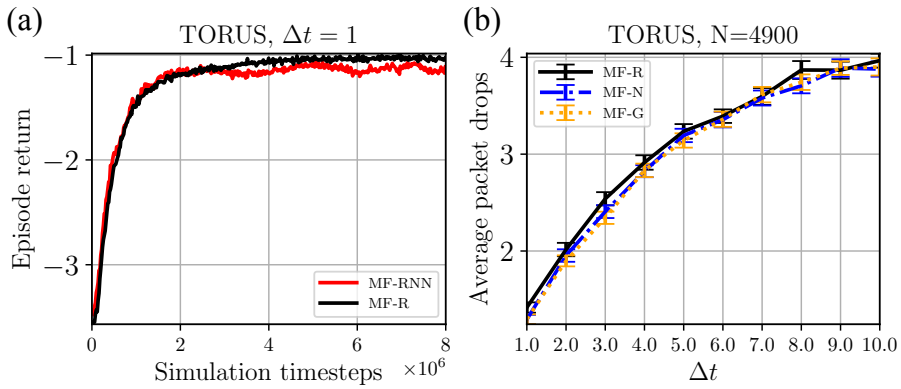

FIGURE 5.13: (a): The training for TORUS with and without RNN policies converges almost to the same return. (b): Evaluation of the learned policy using different observations. MF-R uses only own queue state, MF-N additionally uses neighbor queue states, and MF-G uses state of all queues in the system.

Fig. 5.12 shows the performance comparison for Bethe lattice graphs of the learned MF-R policy with JSQ, RND, and OWN algorithms over different Δt. We show the result only for a large BETHE network, with $o_b = 11$, $d = 3$ and $N = 6142$. OWN outperforms RND more significantly than in CM, since disproportionately many packets arrive at nodes connected to leaf nodes. It can be seen that the system behaves as expected when Δt is small, with JSQ quickly outperformed by MF-R, while MF-R eventually performs worse than OWN as Δt increases. This behavior is due to the fact that in structures such as regular trees, the central node as the root node is not sufficient to represent the performance of all nodes, especially the many leaf nodes that behave differently even in the limit. Because of this loss of regularity, theoretical guarantees fail when scaling large Bethe or similar topologies, as explained in Section 5.5.1. Nevertheless, we see that the learned policy can outperform existing solutions in certain regimes, e.g., at $\Delta t = 3$.

RNNs and decentralized execution

We additionally trained our MF-R policy with recurrent neural networks in PPO, as is typical for partially-observed problems [157], MF-RNN. In Fig. 5.13(a), we see that their effect

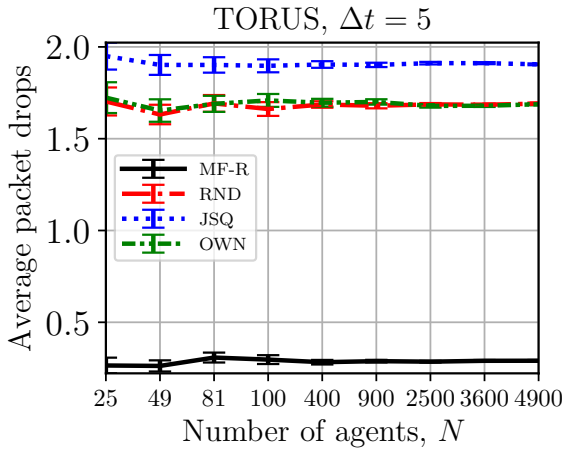

FIGURE 5.14: Performance comparison for increased buffer capacity of each queue to 20. The system utilization was increased to ensure occurrence of packet drops in a limited amount of time.

is negligible and can even be negative. Finally, we used partially observed decentralized alternatives to the empirical distribution as input to the learned MF-R policy in the evaluation, see Fig. 5.13(b). In MF-N we use the distribution of neighbors' queue states, in MF-G we use the empirical distribution of all queues in the system, and in MF-R only the agent's own queue state information is used as a one-hot vector. Similar performances indicate that the learned policy can be executed locally without using global information.

Increased buffer size

For completeness and to illustrate generality, we also performed an experiment in which the buffer size for each queue was increased from 5 to 20. To achieve faster convergence to a learned policy, we increased the arrival rate to the service rate of 1. We also increased the time steps from 50 to 200 so that the queues can be sufficiently filled, and the policy can be learned over the increased state space. Fig. 5.14 shows that our learned policy outperforms the other algorithms at $\Delta t = 5$ for TORUS graph. However, this increased the training time.

Heterogeneous servers

We also conducted experiments where we considered the servers to be heterogeneous, with randomly assigned speed of fast (rate 2) or slow (rate 1). The workload was the same; modulating between $[0.9, 0.6]$. Fig. 5.15 shows that our algorithm has comparable performance to RND and OWN while outperforming JSQ. We also compared with SED, the state-of-the-art load balancing algorithm for heterogeneous servers [22], which performs better since it makes decisions based on both the neighbors' queue state information and the server speeds. We believe our model is not an ideal fit for this setting and further work needs to be done in this direction.

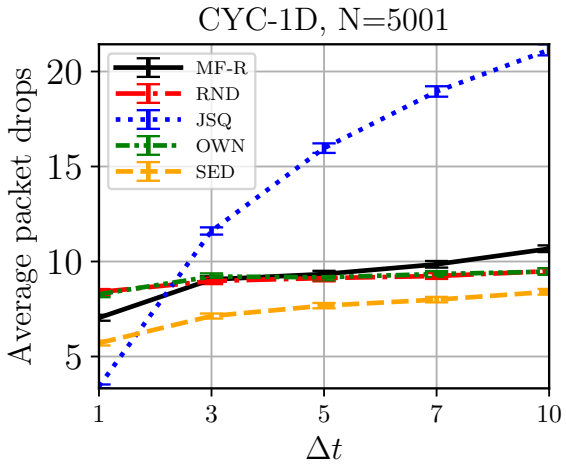

FIGURE 5.15: Performance comparison of MF-R when the servers are considered to be heterogeneous. Additional comparison was done with the Shortest-expected-delay algorithm [22], which is the state-of-the-art when the servers are heterogeneous.

5.6 SUMMARY

In this chapter, we considered a large and highly local queuing system and learned an efficient load balancing algorithm for it. In order to learn a scalable as well as localized load balancing policy, we have used the sparse mean-field approximations, while also providing theoretical guarantees. After modeling our multi-agent queuing system as a sparse MFC MDP, we have used the state-of-the-art RL algorithm PPO to learn the policy. For the multi-agent system to be synchronous, we have also considered the synchronization delay, as in the previous chapter. We have used different kinds of regular graph topologies for evaluation of our proposed solution and then compared to well-known baselines such as JSQ, RND, etc. Our approach outperforms the well-known baselines and can scale to queuing systems of arbitrary size with arbitrary synchronization delays.

6

CONCLUSIONS AND OUTLOOK

In this thesis, we have looked into different into two main challenges of queuing networks, namely network delays and scalable load balancing policies. The main components of our queuing system are first-in-first-out (FIFO) queues, servers which process the jobs waiting in queues and load-balancers. The task of the load-balancer is to allocate the arriving jobs to available queue resources, given the queue information it has available at that time. While, the allocation rule is dependent on the load balancing policy the load-balancer uses, which translates the state of the queues into a decision of which queue to send the jobs to at this time.

We started by looking at a system with a large number of queues and a single load-balancer in Chapter 3. Due to the first challenge, the updated queue state information is received by the load-balancer with a delay, leading to partial observability. We address this by modelling the system ad a partially observable Markov decision process (POMDP) and then solving it using the state-of-the-art Monte Carlo tree search (MCTS) algorithm, under different underlying inter-arrival and service time distributions and real network data provided by Kaggle.

In Chapter 4 we extended this model to also have a large number of load-balancers (agents). In order to still have a synchronous system, we consider an arbitrary network delay such that all agents can receive the queue state information. Learning scalable policies in multi-agent systems is tricky because of challenges such as non-stationary, computational complexity, partial observability and credit assignment. To cater these, we have used the mean-field control approach to convert the multi-agent multi-queue system into a single-agent Markov decision process (MDP), which is then solved using the state-of-the-art proximal policy optimization (PPO) algorithm. We have also provided theoretical guarantees to show that the policy learned i n the limiting single-agent system performs well in the finite multi-agent multi-queue system as long as the system size is large.

A natural extension to the work of Chapter 4 was then to consider locality of interaction between load-balancers, which can be represented by an underlying sparse but regular graph structure. Here we have used the recently published sparse mean-field theory to convert the multi-agent multi-queue system having sparse neighbors and queue accesses into a sparse single-agent mean-field control MDP with theoretical performance guarantees. Different types of network topologies, including torus and cube connected cycles, were considered, and it was shown that the learned load balancing policy can be used on queuing systems of arbitrary size with arbitrary synchronization delays.

This work can have many interesting future directions. One would be to consider a system with heterogeneous jobs types, such that a job could take more than one slot in the queue or could have priority over other jobs. This would change the learned policy since every action will not have the same effect on the queue state, rather it would now also depend on the job

type. Another direction would be to not allocate every queue its own designated server, rather one server could have access to multiple queues. This combined with heterogeneous servers and/or non FIFO queues would make the system much more complex, especially in the limiting case. Another direction would be to consider non-exponential inter-arrival service times also in Chapter 4 and 5 and derive a mean-field model for them.

APPENDICES

APPENDIX A: PROBABILITY THEORY

Here we first give definitions related to probability theory and then give an explanation of the probability distributions used throughout this thesis. For more details see [15, 136, 169].

PROBABILITY SPACE is defined by the tuple $(\Omega, \mathcal{F}, \mathcal{P})$, where Ω is the sample space, $\mathcal{F}$ is the σ-algebra and $\mathcal{P}\colon \mathcal{F} \to [0, 1]$ is the probability measure.

RANDOM VARIABLE (RV) is a function which assigns values to each outcome of an experiment, $X\colon \Omega \to \mathcal{X}$ for some space $\mathcal{X}$. If these values are countable, the RV X is discrete, else it is a continuous RV.

A.1 PROBABILITY DISTRIBUTIONS

A.1.1 *Exponential Distribution*

An exponential distribution is defined by rate parameter $\lambda > 0$ and its PDF is given by

$$\mathrm{Exp}\,(x \mid \lambda) = \begin{cases} \lambda e^{\lambda x}, & x \geq 0, \\ 0, & x < 0, \end{cases} \tag{A.1.1}$$

with support $x \in [0, \infty)$. The expected value and variance of a continuous random variable $X \sim \mathrm{Exp}\,(\lambda)$ which is exponentially distributed are $E[X] = \frac{1}{\lambda}$ and $\mathrm{Var}[X] = \frac{1}{\lambda^2}$.

A.1.2 *Poisson Distribution*

A discrete RV has a Poisson distribution, $X \sim \mathrm{Pois}\,(\lambda)$, with rate parameter $\lambda > 0$ if its probability mass function (pmf) is given as :

$$\mathrm{Pois}\,(k \mid \lambda) = \mathbb{P}(X = k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{A.1.2}$$

where k is the number of occurrences. A Poisson Distribution tells the probability that k number of events occur in a fixed interval if the events occur with a constant rate and are independent of the time since last event. The mean and variance of X is given as $E[X] = \text{Var}[X] = \lambda$.

A.1.3 *Binomial Distribution*

The binomial distribution is defined by two parameters: (i) number of trials $n \in \mathbb{N}$ and (ii) the success probability of each trial $p \in [0, 1]$. It tells the probability of having k successes in a sequence of n independent trials, with success probability of p and the trials only have 2 possible outcomes. A discrete RV follows the binomial distribution, $X \sim \text{Bin}\,(n, p)$ if its pmf is given as:

$$\text{Bin}\,(k \mid n, p) = \begin{cases} \binom{n}{k} p^k (1-p)^{n-k}, & \text{for } k = 0, 1, 2, \ldots, n \\ 0, & \text{otherwise} \end{cases} \tag{A.1.3}$$

where k is the number of successes and $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ is the binomial coefficient. The expected value and variance of X are given as $E[X] = np$ and $\text{Var}[X] = np(1-p)$.

A.1.4 *Bernoulli Distribution*

It is a special case of the Binomial distribution, defined by probability parameter p, where the Bernoulli distributed RV X takes value 1 with probability p and value 0 with probability $1 - p$. The pmf of $X \sim \text{Bern}\,(p)$ over k possible outcomes is given as:

$$\text{Bern}\,(k \mid p) = \begin{cases} p, & \text{if } k = 1 \\ 1-p, & \text{if } k = 0. \end{cases} \tag{A.1.4}$$

The expected value and variance of X are given as $E[X] = p$ and $\text{Var}[X] = p(1-p)$.

A.1.5 *Categorical Distribution*

It is the generalization of the Bernoulli distribution when the categorical RV $X^{(k)}$ can have one of the k possible outcomes with each outcome or category having a separately specified probability, p_k. The categorical RV $X^(k)$ having set k as sample space has pmf defined as:

$$\text{Cat}\,(x = i \mid \mathbf{p}) = p_i \tag{A.1.5}$$

with support $x \in \{x^{(i)} \mid i = 1, \ldots, n\}$, probability vector $\mathbf{p} = [p_1, \ldots, p_n]$ and p_i the probability of seeing element i, $\sum_i p_i = 1$. The expected value and variance of $X^{(k)}$ are given as $E[X^{(k)}] = p_k$ and $\text{Var}[X^{(k)}] = p_k(1 - p_k)$.

A.1.6 *Geometric Distribution*

It gives the probability of the number of failures before a success occurs in a Bernoulli trial having success probability p. A discrete RV $X \sim \text{Geom}\,(p)$ for $k = 1, 2, 3, 4, \ldots,$ has a geometric distribution if its pmf is given as:

$$\text{Geom}\,(k \mid p) = (1-p)^{k-1}p. \tag{A.1.6}$$

The expected value and variance of X are given as $E[X] = \frac{1}{p}$ and $\text{Var}[X] = \frac{1-p}{p^2}$.

A.1.7 *Uniform Distribution*

If all possible outcomes for a RV X are equally likely then it is represented using a uniform distribution with parameters $a, b > 0$ with $b \geq a$ and support $[a, b]$. Depending on the values of X, uniform distribution can be both discrete and continuous. For continuous RV $X \sim \text{Unif}\,(a, b)$ the pdf is given as:

$$\text{Unif}\,(x \mid b, a) = \begin{cases} \frac{1}{b-1}, & \text{if } a \leq x \leq b \\ 0, & \text{otherwise.} \end{cases} \tag{A.1.7}$$

And the expected value and variance of X are given as $E[X] = \frac{b+a}{2}$ and $\text{Var}[X] = \frac{(b-1)^2}{12}$.

A.1.8 *Gamma Distribution*

A continuous RV X is Gamma distributed if its pdf is given as:

$$\text{Gam}\,(x \mid a, b) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}, \tag{A.1.8}$$

where $\alpha \in \mathbb{R}_{>0}$ and $\beta \in \mathbb{R}_{>0}$ are the shape and rate parameters, respectively, defining the gamma distribution with support $x \in \mathbb{R}_{>0}$. The expected value and variance of a gamma distributed RV X are given as $E[X] = \frac{\alpha}{\beta}$ and $\text{Var}[X] = \frac{\alpha}{\beta^2}$.

A.1.9 *Pareto Distribution*

It is a heavy-tailed distribution defined by parameters scale $x_m > 0$ and shape (tail parameter) $\alpha > 0$. A continuous random variable X following Pareto distribution has the following pdf:

$$\text{Par}\,(x \mid b, a) = \begin{cases} \frac{\alpha x_m^\alpha}{x^{\alpha+1}}, & x \geq x_m \\ 0, & x < x_m. \end{cases} \tag{A.1.9}$$

The expected value and variance of a Pareto distributed RV X are given as:

$$E[X] = \begin{cases} \infty, & \alpha \leq 1, \\ \frac{\alpha x_m}{\alpha+1}, & \alpha > 1, \end{cases} \tag{A.1.10}$$

$$\text{Var}[X] = \begin{cases} \infty, & \alpha \in (1, 2], \\ \left(\frac{x_m}{\alpha-1}\right)^2 \frac{\alpha}{\alpha-2}, & \alpha > 2. \end{cases} \tag{A.1.11}$$

B

APPENDIX B: TREE SEARCH ALGORITHMS

B.1 MONTE CARLO TREE SEARCH

MCTS is a heuristic search and planning framework for finding optimal decisions by sampling a given model [170]. The key idea is to do a tree search that keeps a balance between exploration and exploitation, which simply means that it exploits the best found actions up till now while also continuing to explore the action space for alternate better actions. It rests on two fundamental concepts, (i) the true value of an action may be approximated using random simulation, and (ii) that these values may be used efficiently to adjust the policy towards a best-first strategy. In an MCTS the nodes of the tree are the states, x, while the tree edges represent the possible actions, u, from that state. And each node contains:

- Total count for the state: $N(x)$
- Action value: $V(x, u)$
- A count for each action in that state: $N(x, u)$

An MCTS can be used to solve an MDP and its basic working consists of the following four steps:

- Selection: choosing the action to take at the current node. This is the most delicate part of the algorithm and has many sophisticated methods available. The one we have used in Chapter 3 is called as upper confidence on trees (UCT) algorithm and is explained in Section B.1.1.
- Expansion: Once a leaf node is reached and no more selection is possible, the tree is expanded by one node either in breadth or height.
- Simulation: from the selected child node, simulate the tree until a pre-decided $depth$ to evaluate its value.
- Backpropagation: update the entire tree till the root by backpropagating. $N(x)$ and $N(x, u)$ are incremented by one for that state.

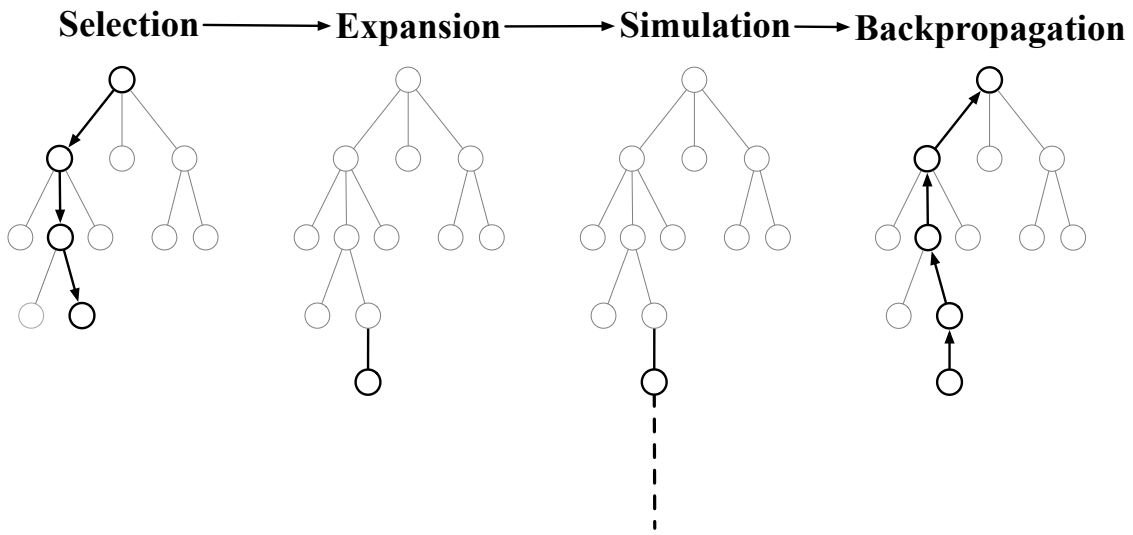

FIGURE B.1: Visualization of one iteration of the MCTS algorithm.

For a visualization of the algorithm, see Fig. B.1. MCTS is an attractive framework because it is a-heuristic (does not need domain-specific knowledge), asymmetric (tree growth is quickly skewed towards more promising actions) and an anytime (back-propagation ensures all values are always up-to-date, so learning can be stopped to use the current best estimate) algorithm.

However, it also has some drawbacks. Firstly, it is memory intensive but this can be resolved by pruning the tree after some timesteps. Secondly, depending on the complexity of the problem it may need a large number of iterations to find a good solution, however different kinds of parallelizations can be used to speed this up [111, 171].

B.1.1 *Upper confidence bound on trees*

The UCT algorithm [108] consider each node (state) of the tree as a multi-armed bandit problem [172] and then chooses the next action based on the following:

$$\tilde{V}(x,u) = V(x,u) + c\sqrt{\frac{\log N(x)}{N(x,u)}}$$

where c is the exploration constant. The actions is then selected as $u = \arg\max_u \tilde{V}(x,u)$. Note that if $c = 0$ then UCT becomes a purely greedy algorithm always choosing the best action. This enables a balance between the number of explored actions and the best action at the moment.

B.2 POMCP

The POMCP framework is an extension of the state-of-the-art MCTS algorithm to learn a scalable, optimal policy for a POMDP by constructing online a search tree of histories, h.

And each node, T, now keeps an estimates of a history instead of state. The state is then sampled from the belief state, $B(h)$, which is represented using a set of particles. They show that as long as the belief state is close to the actual state of the environment, POMCP will be able to learn an optimal policy for the POMDP. Monte Carlo simulations are used for the tree search and belief state updates.

The POMCP algorithm used in Chapter 3 is given here as Algorithm B.2. For further details see [98].

Function SEARCH (h)**:**
  repeat
    if $h = empty$ **then**
      $x \sim \mathcal{I}$
    else
      $x \sim B(h)$
    end
    Simulate($x, h, 0$)
  until *Timeout()*
return $\arg\max_b V(hb)$

Function ROLLOUT ($x, h, depth$)**:**
  if $\gamma^{depth} < \epsilon$ **then**
    return 0
  end
  $u \sim \pi_{rollout}(h, \cdot)$
  $(x', o, r) \sim \mathcal{G}(x, u)$
  return $r + \gamma \cdot ROLLOUT(x', huo, depth + 1)$

Function SimulateTree ($x, h, depth$)**:**
  if $\gamma^{depth} < \epsilon$ **then**
    return 0
  end
  if $h \notin T$ **then**
    for $u \in U$ **do**
      $T(hu) \leftarrow (N_{init}(hu), V_{init}(hu), \varnothing)$
    end
    return $ROLLOUT(x, h, depth)$
  end
  $u \leftarrow \arg\max_b V(hb) + c\sqrt{\frac{\log N(h)}{N(hb)}}$
  $(x', o, r) \sim \mathcal{G}(x, u)$
  $R \leftarrow r + \gamma \cdot SimulateTree(x', huo, depth + 1)$
  $B(h) \leftarrow B(h) \cup \{s\}$
  $N(h) \leftarrow N(h) + 1$
  $N(hu) \leftarrow N(hu) + 1$
  $V(hu) \leftarrow V(hu) + \frac{R - V(ha)}{N(ha)}$
return R

FIGURE B.2: Pseudo-code for working of POMCP.

C

APPENDIX C:PROXIMAL POLICY OPTIMIZATION

c.1 PPO EXPLAINED

PPO is a state-of-the-art RL algorithm that is easy to implement, stable, easy to tune and sample efficient. It has had huge success in tasks such as robotic arm control, Atari games and Dota 2 [153, 156]. It is a policy gradient (PG) method which learns to use online data (on-the-go) which the agent generates by interacting with the environment.

PG methods directly optimize the parameter θ which parameterizes the policy, $\pi_\theta(u \mid x)$, where $u \in \mathcal{U}$ is the action and $x \in \mathcal{X}$ is the state, such that expected total reward is maximized. An estimator of the policy gradient is computed and used in the stochastic gradient algorithm, which can be of the form: $\hat{g} = \hat{\mathbb{E}}[\nabla\theta \log \pi_\theta(u(t) \mid x(t))\hat{A}(t)]$, where $\pi_\theta(u(t) \mid x(t))$ is the stochastic policy represented by a neural network that takes as input the environment's observed states, $x(t)$, and outputs actions to take, $u(t)$. and $\hat{A}(t)$ is an estimate of the relative value of taking this action $u(t)$ in state $x(t)$, so $\hat{A}(t) = (\sum_{t=0}^{\infty} \gamma(t)r(t) + t) + V(x(t))$, where the first term is the discounted sum of rewards the agent received during each timestep t in the current episode and the second term is the value function which gives an estimate of the discounted sum of rewards from this state onwards, using neural networks. The neural network of the value function is updated using the experience agent collects from the environment. So advantage estimate, $\hat{A}(t)$, tells you how much better/worse was the action selected then the expected reward based on past experiences. Then, the loss function of the PG method is: $\mathrm{L}^{\mathrm{PG}}(\theta) = \hat{\mathbb{E}}[\log \pi_\theta(u(t) \mid x(t))\hat{A}(t)]$. Note that $\hat{A}(t)$ can be both positive (negative) when the chosen action yields return better (worse) than the expected return. If $\hat{A}(t) > 0$ then the gradient is positive and the probability of choosing action $u(t)$ in state $x(t)$ increases in future and vice versa. However, in order to have a stable training process, the change in policy or parameter updates should not have too much deviation after one step.

In order to avoid too high or low jumps in the policy update, PPO uses a clipped surrogate objective. Consider the ratio between old and new policies, $r(t, \theta) = \frac{\pi_\theta(u(t)|x(t))}{\pi_{\theta_{old}}(u(t)|x(t))}$, meaning

$r(t, \theta_{old}) = 1$ and $r(t, \theta) > 1$ if the probability of $u(t)$ in $x(t)$ is higher in policy π_θ than it was in policy $\pi_{\theta_{old}}$, otherwise it is in $[0, 1]$. The proposed surrogate loss function to maximize in PPO is given as:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}[\min(r(t, \theta)\hat{A}(t), \text{clip}(r(t, \theta), 1 - \epsilon, 1 + \epsilon)\hat{A}(t))] \tag{C.1.1}$$

where ϵ is a tunable hyperparameter that says how far the new policy can deviate from the old policy. The expectation is computed over batches of trajectories and is taken over the minimum of two terms. The first term inside the minimum, $r(t, \theta)\hat{A}(t)$, is the default policy gradient objective, which pushes the policy towards actions which yield higher positive advantage over the baseline [173]. While the second term is the clipped version of the first one where clipping ensures that the ratio $r(\theta)$ ranges between $1 \pm \epsilon$, where usually $\epsilon = 0.2$. The loss $L^{\text{CLIP}}(\theta)$ is then the minimum of the original and the clipped values. This clipping restricts the policy updates, making PPO stable.

With neural networks it is efficient to share parameters between policy and value neural networks, hence the loss function needs to keep a balance between the two. The final objective function of PPO is then given as:

$$L(\theta) = \hat{\mathbb{E}}[L^{\text{CLIP}}(\theta) + \kappa_c L^{\text{VF}}(\theta) + c_2 S[\pi_\theta](x(t))] \tag{C.1.2}$$

where c_2,κ_c are tunable coefficients, S is the entropy bonus or regularizer that ensures sufficient exploration during training and $L^{\text{VF}}(\theta) = (\text{V}_\theta(x(t)) - \text{V}^{\text{target}})^2$ is the squared-error loss which is the expected amount of rewards from this state onwards. For details, see [156].

C.2 HYPERPARAMETERS OF PPO

One of the main advantages of PPO is that its hyperparameters are easy to tune and understand. Here we give a list of these hyperparameters, their description and common range of values.

- Number of epochs, T_b: steps up to which the trajectories are carried out before performing optimization, like stochastic gradient descent (SGD) or Adam, on the collected experience. Typical range $32 - 5000$.
- SGD Mini batch size, B_m: sample size to take from the collected experience to perform optimization on. Typical range $4 - 4096$.
- Number of SGD iterations, I_m: total optimization iterations to perform on the sampled minibatch. Typical range: $3 - 30$.
- Clip parameter, ϵ: range up to which the policy can deviate while still improving the objective function. Typical range $0.1 - 0.3$.
- KL target, κ_t: how much KL divergence is acceptable between new and old policy after an update. Typical range: $0.01 - 0.05$.
- KL coefficient, κ_c: initial coefficient for KL divergence.

- Discount factor, γ: determines how much influence should future rewards have.
- GAE Parameter, λ_{RL}: to adjust the bias-variance trade-off in combination with γ in the generalized advantage estimator. Typical range: $0 - 1$.
- Learning rate, l_r: learning rate for policy and value function optimizers. Can be varying over time or fixed throughout.

C.3 PARAMETER SHARING

For homogeneous agents, it is more efficient if their policies are trained using parameter sharing (PS) [174, 175]. In PS, the agents share the parameters of a single policy while using the data collected from the experience of all agents simultaneously. However, each agent can still act differently based on their own current state, but they sample this action from the single policy being trained. Once the centralized policy is learned, it can then be used by each agent in a decentralized manner, falling under the centralized learning and decentralized execution regime [48].

SYMBOL	DESCRIPTION
$\mathbb{N}$	The set of natural numbers.
$\mathbb{N}_{>i}$, $\mathbb{N}_i$	The set of natural numbers with elements greater/ greater or equal to i.
$\mathbb{R}$	The set of real numbers.
$\mathbb{R}_{>t}$, $\mathbb{R}_{\geq t}$	The set of real numbers with elements greater/ greater or equal to t.
Δ^n	The n-dimensional probability simplex; i.e., $\Delta^n = \{\mathbf{x} \in \mathbb{R}^n : \sum_{i=1}^n x_i = 1 \wedge x_j \geq 0, \forall j \in \{1, \ldots, n\}\}$.
$\mathbb{1}_{(\cdot)}$	The indicator function.
$\mathbf{e}_x$	A Unit vector with all zeros except at the x-th position.
δ_x	Dirac measure defined for a given $x \in \mathcal{X}$.
$\mid \mathcal{X} \mid$	Cardinality of set $\mathcal{X}$.
$f(x)$	A function f of a variable x.
$J[f]$	A functional J of a function f.
$\nabla_x(\cdot)$ or $\frac{\partial}{\partial x}(\cdot)$	The gradient w.r.t. to x.
$\frac{\delta}{\delta f}(\cdot)$	The functional derivative w.r.t. to the function f.
$\mathcal{P}(\cdot)$	A probability measure.
$\mathbb{P}(\cdot)$	A probability density function or probability mass function
$\mathbb{E}[\cdot]$	The expectation operator.
$\mathrm{Gam}\,(\cdot \mid \alpha, \beta)$	Probability density function of the gamma distribution with shape parameter α and rate parameter β.
$\mathrm{Cat}\,(\cdot \mid \mathbf{p})$	Probability mass function of the categorical distribution with probability vector $\mathbf{p}$
$\mathrm{Exp}\,(\cdot \mid \lambda)$	Probability density function of the exponential distribution with rate parameter λ
$\mathrm{Unif}\,(\cdot \mid a, b)$	Probability density function of the uniform distribution with lower bound a and upper bound b.
$\mathrm{Bern}\,(\cdot \mid p)$	Probability mass function of the Bernoulli distribution with parameter p.
$\mathcal{N}\,(\cdot \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Probability density function of the (multivariate) normal or Gaussian distribution with mean parameter $\boldsymbol{\mu}$ and variance/covariance matrix $\boldsymbol{\Sigma}$.
$\mathrm{Bin}\,(\cdot \mid N, p)$	Probability mass function of the binomial distribution with number of trials N and success probability p
$\mathrm{Mult}\,(\cdot \mid N, \mathbf{p})$	Probability mass function of the multinomial distribution with number of trials N and probability vector $\mathbf{p}$
$\mathrm{Dir}\,(\cdot \mid \boldsymbol{\alpha})$	Probability density function of the Dirichlet distribution with concentration parameter vector $\boldsymbol{\alpha}$

SYMBOL	DESCRIPTION
$\mathrm{Pois}\,(\cdot \mid \lambda)$	Probability mass function of the Poisson distribution with rate parameter λ
$\mathrm{Beta}\,(\cdot \mid \alpha, \beta)$	Probability density function of the beta distribution with shape parameters α, β.
$\mathrm{Par}\,(\cdot \mid \alpha, \beta)$	Probability density function of the Pareto distribution with scale β and shape α.

ACRONYMS

POL partial observability load-balancer

POSMDP partially observable semi Markov decision process

SMDP semi Markov decision process

POMDP partially observable Markov decision process

MDP Markov decision process

MMDP multi-agent Markov decision process

MDPS Markov decision processes

MCTS Monte Carlo tree search

SIR sequential importance resampling

UCT upper confidence bounds for trees

SIS sequential importance resampling

RL reinforcement learning

MARL multi-agent reinforcement learning

MFC MDP mean-field control Markov decision process

MFC mean-field control

MFG mean-field games

MF mean-field

JSQ join-the-shortest-queue

JIQ join-the-idle-queue

SED shortest-expected-delay

PPO proximal policy optimization

RND random

OWN Send-to-own-queue

MF-R MF-Random

NA-PS-RND MARL-PS

SAC scalable-actor-critic

DEC-POMDP decentralized partially observable Markov decision process

MCMC Markov Chain Monte Carlo

DPP dynamic programming principle

BIBLIOGRAPHY

[1] A. Tahir, B. Alt, A. Rizk, and H. Koeppl, "Load balancing in compute clusters with delayed feedback", *IEEE Transactions on Computers*, 2022.

[2] A. Tahir, K. Cui, and H. Koeppl, "Learning mean-field control for delayed information load balancing in large queuing systems", in *Proceedings of the 51st International Conference on Parallel Processing*, 2022, pp. 1–11.

[3] A. Tahir, K. Cui, and H. Koeppl, "Sparse mean field load balancing in large localized queueing systems", *arXiv preprint arXiv:2312.12973*, pp. 1–22, 2023.

[4] A. Tahir, K. Cui, B. Alt, A. Rizk, and H. Koeppl, "Collaborative optimization of the age of information under partial observability", *arXiv preprint arXiv:2312.12977*, pp. 1–10, 2023.

[5] A. Tahir, H. Al-Shatri, K. Kiekenap, and A. Klein, "Ultra-reliable low latency communication for consensus control in multi-agent systems", in *IEEE Wireless Communications and Networking Conference (WCNC)*, IEEE, 2019, pp. 1–7.

[6] S. Azem, A. Tahir, and H. Koeppl, "Dynamic time slot allocation algorithm for quadcopter swarms", in *IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, IEEE, 2022, pp. 1–6.

[7] J. Jia, A. Tahir, and H. Koeppl, "Decentralized coordination in partially observable queueing networks", in *IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2022, pp. 1491–1496.

[8] K. Cui, A. Tahir, M. Sinzger, and H. Koeppl, "Discrete-time mean field control with environment states", in *60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 5239–5246.

[9] Y. Alkhalili, J. Weil, A. Tahir, T. Meuser, B. Koldehofe, A. Mauthe, H. Koeppl, R. Steinmetz, *et al.*, "Towards QoE-driven optimization of multi-dimensional content streaming", *Electronic Communications of the EASST*, vol. 80, 2021.

[10] J. Weil, Y. Alkhalili, A. Tahir, T. Gruczyk, T. Meuser, M. Mu, H. Koeppl, and A. Mauthe, "Modeling quality of experience for compressed point cloud sequences based on a subjective study", in *15th International Conference on Quality of Multimedia Experience (QoMEX)*, IEEE, 2023, pp. 135–140.

[11] K. Cui, M. B. Yilmaz, A. Tahir, A. Klein, and H. Koeppl, "Optimal offloading strategies for edge-computing via mean-field games and control", in *IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2022, pp. 976–981.

[12] K. Cui, A. Tahir, G. Ekinci, A. Elshamanhory, Y. Eich, M. Li, and H. Koeppl, "A survey on large-population systems and scalable multi-agent reinforcement learning", *arXiv preprint arXiv:2209.03859*, pp. 1–21, 2022.

[13] K. Cui, C. Fabian, and H. Koeppl, "Multi-agent reinforcement learning via mean field control: Common noise, major agents and approximation properties", *arXiv preprint arXiv:2303.10665*, 2023.

[14] H. Chen, D. D. Yao, *et al.*, *Fundamentals of queueing networks: Performance, asymptotics, and optimization*. Springer, 2001, vol. 4.

[15] G. Bolch, S. Greiner, H. De Meer, and K. S. Trivedi, *Queueing networks and Markov chains: Modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[16] U. N. Bhat, *An introduction to queueing theory: Modeling and analysis in applications*. Springer, 2008, vol. 36.

[17] J. R. Norris, *Markov chains*. Cambridge university press, 1998.

[18] W. Fischer and K. Meier-Hellstern, "The Markov-modulated Poisson process (MMPP) cookbook", *Performance evaluation*, vol. 18, no. 2, pp. 149–171, 1993.

[19] A. Hordijk and G. Koole, "On the optimality of the generalized shortest queue policy", *Probability in the Engineering and Informational Sciences*, vol. 4, no. 4, pp. 477–487, 1990.

[20] W. Winston, "Optimality of the shortest line discipline", *Journal of Applied Probability*, vol. 14, no. 1, pp. 181–189, 1977.

[21] S. A. Banawan and J. Zahorjan, "Load sharing in heterogeneous queueing systems", in *Proceedings 8th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 1989, pp. 731–732.

[22] J. Selen, I. Adan, S. Kapodistria, and J. van Leeuwaarden, "Steady-state analysis of shortest expected delay routing", *Queueing Systems*, vol. 84, no. 3-4, pp. 309–354, 2016.

[23] M. Mitzenmacher, "The power of two choices in randomized load balancing", *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.

[24] S. T. Maguluri, R. Srikant, and L. Ying, "Heavy traffic optimal resource allocation algorithms for cloud computing clusters", *Performance Evaluation*, vol. 81, pp. 20–39, 2014.

[25] M. Mitzenmacher, "Load balancing and density dependent jump Markov processes", in *Proceedings of 37th Conference on Foundations of Computer Science*, IEEE, 1996, pp. 213–222.

[26] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

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

[28] R. Bellman, "Dynamic programming", *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[29] R. Bellman, "A Markovian decision process", *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[30] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics", *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[31] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation", *Advances in neural information processing systems*, vol. 12, 1999.

[32] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients", *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[33] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey", *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[34] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains", *Artificial Intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[35] S. Guicheng and W. Yang, "Review on Dec-POMDP model for MARL algorithms", in *Smart Communications, Intelligent Algorithms and Interactive Methods*, Springer, 2022, pp. 29–35.

[36] C. Amato and F. Oliehoek, "Scalable planning and learning for multiagent POMDPs", in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, PKP Publishing Services Network, 2015, pp. 1–8.

[37] F. A. Oliehoek and C. Amato, *A concise introduction to decentralized POMDPs*. Springer, 2016.

[38] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network", *Physica D: Nonlinear Phenomena*, vol. 404, p. 132 306, 2020.

[39] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability", in *International Conference on Machine Learning*, PMLR, 2017, pp. 2681–2690.

[40] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs", in *2015 AAAI fall symposium series*, 2015.

[41] D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber, "Solving deep memory POMDPs with recurrent policy gradients", in *17th International Conference on Artificial Neural Networks (ICANN)*, Springer, 2007, pp. 697–706.

[42] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson, "Deep variational reinforcement learning for POMDPs", in *International Conference on Machine Learning*, PMLR, 2018, pp. 2117–2126.

[43] C. C. White, "Procedures for the solution of a finite-horizon, partially observed, semi-Markov optimization problem", *Operations Research*, vol. 24, no. 2, pp. 348–358, 1976.

[44] N. A. Vien, H. Ngo, S. Lee, and T. Chung, "Approximate planning for Bayesian hierarchical reinforcement learning", *Applied Intelligence*, vol. 41, pp. 808–819, 2014.

[45] S. Omidshafiei, A.-A. Agha-Mohammadi, C. Amato, and J. P. How, "Decentralized control of partially observable Markov decision processes using belief space macro-actions", in *IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2015, pp. 5962–5969.

[46] H. Yu *et al.*, "Approximate solution methods for partially observable Markov and semi-Markov decision processes", Ph.D. dissertation, Massachusetts Institute of Technology, 2006.

[47] H. C. Tijms, *A first course in stochastic models*. John Wiley and sons, 2003.

[48] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms", *Handbook of Reinforcement Learning and Control*, pp. 321–384, 2021.

[49] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning", *Autonomous Agents and Multi-Agent Systems*, vol. 33, no. 6, pp. 750–797, 2019.

[50] J.-M. Lasry and P.-L. Lions, "Mean field games", *Japanese Journal of Mathematics*, vol. 2, no. 1, pp. 229–260, 2007.

[51] M. Huang, R. P. Malhamé, P. E. Caines, *et al.*, "Large population stochastic dynamic games: Closed-loop McKean-Vlasov systems and the Nash certainty equivalence principle", *Communications in Information & Systems*, vol. 6, no. 3, pp. 221–252, 2006.

[52] M. Laurière, S. Perrin, M. Geist, and O. Pietquin, "Learning mean field games: A survey", *arXiv preprint arXiv:2205.12944*, 2022.

[53] W. U. Mondal, M. Agarwal, V. Aggarwal, and S. V. Ukkusuri, "On the approximation of cooperative heterogeneous multi-agent reinforcement learning (MARL) using mean field control (MFC)", *arXiv preprint arXiv:2109.04024*, 2021.

[54] A. Soret, *Mean Field Games with heterogeneous players: From portfolio optimization to network effects*. Columbia University, 2022.

[55] L.-H. Sun, "Mean field games with heterogeneous groups: Application to banking systems", *Journal of Optimization Theory and Applications*, vol. 192, no. 1, pp. 130–167, 2022.

[56] N. Saldi, T. Basar, and M. Raginsky, "Markov–Nash equilibria in mean-field games with discounted cost", *SIAM Journal on Control and Optimization*, vol. 56, no. 6, pp. 4256–4287, 2018.

[57] D. Andersson and B. Djehiche, "A maximum principle for SDEs of mean-field type", *Applied Mathematics & Optimization*, vol. 63, no. 3, pp. 341–356, 2011.

[58] J. Arabneydi and A. Mahajan, "Team optimal control of coupled subsystems with mean-field sharing", in *53rd IEEE Conference on Decision and Control*, IEEE, 2014, pp. 1669–1674.

[59] A. Bensoussan, J. Frehse, P. Yam, *et al.*, *Mean field games and mean field type control theory*. Springer, 2013, vol. 101.

[60] M. F. Djete, D. Possamaï, and X. Tan, "Mckean–Vlasov optimal control: The dynamic programming principle", *The Annals of Probability*, vol. 50, no. 2, pp. 791–833, 2022.

[61] M. Motte and H. Pham, "Mean-field Markov decision processes with common noise and open-loop controls", *The Annals of Applied Probability*, vol. 32, no. 2, pp. 1421–1458, 2022.

[62] N. Saldi, T. Başar, and M. Raginsky, "Approximate markov-nash equilibria for discrete-time risk-sensitive mean-field games", *Mathematics of Operations Research*, vol. 45, no. 4, pp. 1596–1620, 2020.

[63] M. Li, Z. Qin, Y. Jiao, Y. Yang, J. Wang, C. Wang, G. Wu, and J. Ye, "Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning", in *The world wide web conference*, 2019, pp. 983–994.

[64] K. Lin, R. Zhao, Z. Xu, and J. Zhou, "Efficient large-scale fleet management via multi-agent deep reinforcement learning", in *24th ACM SIGKDD International conference on knowledge discovery & data mining*, 2018, pp. 1774–1783.

[65] H. Wang, X. Wang, X. Hu, X. Zhang, and M. Gu, "A multi-agent reinforcement learning approach to dynamic service composition", *Information Sciences*, vol. 363, pp. 96–119, 2016.

[66] M. A. Wiering *et al.*, "Multi-agent reinforcement learning for traffic light control", in *Machine Learning: Proceedings of the Seventeenth International Conference (ICML)*, 2000, pp. 1151–1158.

[67] G. Nuño, "Optimal social policies in mean field games", *Applied Mathematics & Optimization*, vol. 76, pp. 29–57, 2017.

[68] J. Garnier, G. Papanicolaou, and T.-W. Yang, "Large deviations for a mean field model of systemic risk", *SIAM Journal on Financial Mathematics*, vol. 4, no. 1, pp. 151–184, 2013.

[69] H. Gu, X. Guo, X. Wei, and R. Xu, "Dynamic programming principles for learning mfcs", *arXiv preprint arXiv:1911.07314*, 2019.

[70] H. Gu, X. Guo, X. Wei, and R. Xu, "Mean-field controls with Q-learning for cooperative MARL: Convergence and complexity analysis", *SIAM Journal on Mathematics of Data Science*, vol. 3, no. 4, pp. 1168–1196, 2021.

[71] H. Pham and X. Wei, "Bellman equation and viscosity solutions for mean-field stochastic control problem", *ESAIM: Control, Optimisation and Calculus of Variations*, vol. 24, no. 1, pp. 437–461, 2018.

[72] C. Andrieu, N. De Freitas, A. Doucet, and M. I. Jordan, "An introduction to MCMC for machine learning", *Machine Learning*, vol. 50, pp. 5–43, 2003.

[73] D. Van Ravenzwaaij, P. Cassey, and S. D. Brown, "A simple introduction to Markov chain Monte–Carlo sampling", *Psychonomic Bulletin & Review*, vol. 25, no. 1, pp. 143–154, 2018.

[74] W. K. Hastings, *Monte Carlo sampling methods using Markov chains and their applications*. Oxford University Press, 1970.

[75] J. K. Kruschke, "Bayesian data analysis", *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 1, no. 5, pp. 658–676, 2010.

[76] M. D. Hoffman and A. Gelman, "The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo", *Journal Machine Learning Research*, vol. 15, no. 1, pp. 1593–1623, 2014.

[77] R. M. Neal *et al.*, "MCMC using Hamiltonian dynamics", *Handbook of Markov chain Monte Carlo*, vol. 2, no. 11, p. 2, 2011.

[78] A. Doucet, N. De Freitas, and N. Gordon, "An introduction to sequential Monte Carlo methods", *Sequential Monte Carlo Methods in Practice*, pp. 3–14, 2001.

[79] S. Särkkä and L. Svensson, *Bayesian filtering and smoothing*. Cambridge University Press, 2023, vol. 17.

[80] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters", *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[81] I. Polato, R. Ré, A. Goldman, and F. Kon, "A comprehensive view of Hadoop research—A systematic literature review", *Journal of Network and Computer Applications*, vol. 46, pp. 1–25, 2014.

[82] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "Above the clouds: A Berkeley view of cloud computing", *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, no. 13, 2009.

[83] S. A. Banawan and N. M. Zeidat, "A comparative study of load sharing in heterogeneous multicomputer systems", in *Proceedings 25th IEEE Annual Simulation Symposium*, 1992, pp. 22–31.

[84] M. van der Boor, S. C. Borst, J. S. van Leeuwaarden, and D. Mukherjee, "Scalable load balancing in networked systems: A survey of recent advances", *arXiv preprint arXiv:1806.05444*, 2018.

[85] W. Whitt, "Deciding which queue to join: Some counterexamples", *Operations Research*, vol. 34, no. 1, pp. 55–62, 1986.

[86] D. Arcelli, "Exploiting queuing networks to model and assess the performance of self-adaptive software systems: A survey", *Procedia Computer Science*, vol. 170, pp. 498–505, 2020.

[87] S. Shevtsov, M. Berekmeri, D. Weyns, and M. Maggio, "Control-theoretical software adaptation: A systematic literature review", *IEEE Transactions on Software Engineering*, vol. 44, no. 8, pp. 784–810, 2017.

[88] D. Shah and B. Prabhakar, "The use of memory in randomized load balancing", in *Proceedings IEEE International Symposium on Information Theory*, 2002, p. 125.

[89] K. Psounis and B. Prabhakar, "Efficient randomized web-cache replacement schemes using samples from past eviction times", *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 441–454, 2002.

[90] B. Liu, Q. Xie, and E. Modiano, "Reinforcement learning for optimal control of queueing systems", in *57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2019, pp. 663–670.

[91] U. Ayesta, "Reinforcement learning in queues", *Queueing Systems*, pp. 1–3, 2022.

[92] K. Krishnan, "Joining the right queue: A Markov decision-rule", in *26th IEEE Conference on Decision and Control*, IEEE, vol. 26, 1987, pp. 1863–1868.

[93] E. Altman and P. Nain, "Closed-loop control with delayed information", *ACM Sigmetrics Performance Evaluation Review*, vol. 20, no. 1, pp. 193–204, 1992.

[94] J. Kuri and A. Kumar, "Optimal control of arrivals to queues with delayed queue length information", *IEEE Transactions on Automatic Control*, vol. 40, no. 8, pp. 1444–1450, 1995.

[95] D. Artiges, "Optimal routing into two heterogeneous service stations with delayed information", in *Proceedings 32nd IEEE Conference on Decision and Control*, 1993, pp. 2737–2742.

[96] I. Chadès, L. V. Pascal, S. Nicol, C. S. Fletcher, and J. Ferrer-Mestres, "A primer on partially observable markov decision processes (POMDPs)", *Methods in Ecology and Evolution*, vol. 12, no. 11, pp. 2058–2072, 2021.

[97] J. Pineau, G. Gordon, and S. Thrun, "Anytime point-based approximations for large POMDPs", *Journal of Artificial Intelligence Research*, vol. 27, pp. 335–380, 2006.

[98] D. Silver and J. Veness, "Monte-Carlo planning in large POMDPs", in *Advances in Neural Information Processing Systems*, 2010, pp. 2164–2172.

[99] Z. Chen *et al.*, "Bayesian filtering: From Kalman filters to particle filters, and beyond", *Statistics*, vol. 182, no. 1, pp. 1–69, 2003.

[100] J. Elfring, E. Torta, and R. van de Molengraft, "Particle filters: A hands-on tutorial", *Sensors*, vol. 21, no. 2, p. 438, 2021.

[101] F. Daum and J. Huang, "Particle degeneracy: Root cause and solution", in *Signal Processing, Sensor Fusion, and Target Recognition XX*, SPIE, vol. 8050, 2011, pp. 367–377.

[102] J. S. Liu and R. Chen, "Blind deconvolution via sequential imputations", *Journal of the American Statistical Association*, vol. 90, no. 430, pp. 567–576, 1995.

[103] S. M. Ross, *Introduction to probability models*. Academic Press, 2014.

[104] J. Salvatier, T. V. Wiecki, and C. Fonnesbeck, "Probabilistic programming in Python using PyMC3", *PeerJ Computer Sci.*, vol. 2, e55, 2016.

[105] S. Ghosal and A. Van der Vaart, *Fundamentals of nonparametric Bayesian inference*. Cambridge University Press, 2017, vol. 44.

[106] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of Go with deep neural networks and tree search", *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[107] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play", *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[108] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning", in *European Conference on Machine Learning*, Springer, 2006, pp. 282–293.

[109] N. A. Vien and M. Toussaint, "Hierarchical Monte-Carlo planning", in *29th AAAI Conference on Artificial Intelligence*, 2015, pp. 1–7.

[110] P. L'Ecuyer, "Efficiency improvement and variance reduction", in *Proceedings of Winter Simulation Conference*, IEEE, 1994, pp. 122–132.

[111] G. M.-B. Chaslot, M. H. Winands, and H. Herik, "Parallel Monte-Carlo tree search", in *International Conference on Computers and Games*, Springer, 2008, pp. 60–71.

[112] J. S. Rojas, A. Pekar, Á. Rendón, and J. C. Corrales, "Smart user consumption profiling: Incremental learning-based OTT service degradation", *IEEE Access*, vol. 8, pp. 207 426–207 442, 2020.

[113] M. Mitzenmacher, "The power of two choices in randomized load balancing", *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.

[114] M. Mitzenmacher, "How useful is old information?", *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 1, pp. 6–20, 2000.

[115] D. Mukherjee, S. C. Borst, J. S. Van Leeuwaarden, and P. A. Whiting, "Universality of power-of-d load balancing in many-server systems", *Stochastic Systems*, vol. 8, no. 4, pp. 265–292, 2018.

[116] D. A. Dawson, J. Tang, and Y. Q. Zhao, "Balancing queues by mean field interaction", *Queueing Systems*, vol. 49, no. 3, pp. 335–361, 2005.

[117] D. Lipshutz, "Open problem—load balancing using delayed information", *Stochastic Systems*, vol. 9, no. 3, pp. 305–306, 2019.

[118] X. Zhou, N. Shroff, and A. Wierman, "Asymptotically optimal load balancing in large-scale heterogeneous systems with multiple dispatchers", *Performance Evaluation*, vol. 145, p. 102 146, 2021.

[119] M. van der Boor, S. Borst, and J. van Leeuwaarden, "Hyper-scalable JSQ with sparse feedback", *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 1, pp. 1–37, 2019.

[120] J. Anselmi and F. Dufour, "Power-of-d-choices with memory: Fluid limit and optimality", *Mathematics of Operations Research*, vol. 45, no. 3, pp. 862–888, 2020.

[121] X. Guo, A. Hu, R. Xu, and J. Zhang, "Learning mean-field games", in *Advances in Neural Information Processing Systems*, 2019, pp. 4966–4976.

[122] J. Subramanian and A. Mahajan, "Reinforcement learning in stationary mean-field games", in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2019, pp. 251–259.

[123] K. Cui and H. Koeppl, "Approximately solving mean field games via entropy-regularized deep reinforcement learning", in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2021, pp. 1909–1917.

[124] V. Aggarwal, "Machine learning for communications", *Entropy*, vol. 23, no. 7, 2021.

[125] R. Carmona, M. Lauriére, and Z. Tan, "Model-free mean-field reinforcement learning: Mean-field MDP and mean-field Q-learning", *arXiv preprint arXiv:1910.12802*, 2019.

[126] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning", *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[127] N. Brown and T. Sandholm, "Superhuman AI for multiplayer poker", *Science*, vol. 365, no. 6456, pp. 885–890, 2019.

[128] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey", *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[129] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey", *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[130] S. Stidham and R. Weber, "A survey of Markov decision models for control of networks of queues", *Queueing systems*, vol. 13, no. 1-3, pp. 291–314, 1993.

[131] Q.-L. Li, J.-Y. Ma, R.-N. Fan, and L. Xia, "An overview for Markov decision processes in queues and networks", in *International Conference of Celebrating Professor Jinhua Cao's 80th Birthday*, Springer, 2019, pp. 44–71.

[132] K. P. Murphy, *Machine learning: A probabilistic perspective*. MIT Press, 2012.

[133] P. Harremoës, O. Johnson, and I. Kontoyiannis, "Thinning and the law of small numbers", in *IEEE International Symposium on Information Theory*, IEEE, 2007, pp. 1491–1495.

[134] O. Hernández-Lerma and J. B. Lasserre, *Discrete-time Markov control processes: Basic optimality criteria*. Springer Science & Business Media, 2012, vol. 30.

[135] D. Majerek, W. Nowak, and W. Zieba, "Conditional strong law of large number", *Int. J. Pure Appl. Math*, vol. 20, no. 2, pp. 143–156, 2005.

[136] J. Medhi, *Stochastic models in queueing theory*. Elsevier, 2002.

[137] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning", in *International Conference on Machine Learning*, PMLR, 2018, pp. 3053–3062.

[138] D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions", *The Journal of Physical Chemistry*, vol. 81, no. 25, pp. 2340–2361, 1977.

[139] T. Chu, S. Chinchali, and S. Katti, "Multi-agent reinforcement learning for networked system control", in *8th International Conference on Learning Representations*, Addis Ababa, Ethiopia, April 26-30, 2020: OpenReview.net, 2020, pp. 1–17.

[140] Y. Lin, G. Qu, L. Huang, and A. Wierman, "Multi-agent reinforcement learning in stochastic networked systems", *Advances in Neural Information Processing Systems*, vol. 34, pp. 7825–7837, 2021.

[141] X. Liu, H. Wei, and L. Ying, "Scalable and sample efficient distributed policy gradient algorithms in multi-agent networked systems", *arXiv preprint arXiv:2212.06357*, pp. 1–30, 2022.

[142] D. Lacker, K. Ramanan, and R. Wu, "Local weak convergence for sparse networks of interacting processes", *The Annals of Applied Probability*, vol. 33, no. 2, pp. 843–888, 2023.

[143] D. Mukherjee, S. C. Borst, and J. S. Van Leeuwaarden, "Asymptotically optimal load balancing topologies", *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 1, pp. 1–29, 2018.

[144] L. Lovász, *Large networks and graph limits*. American Mathematical Society, 2012, vol. 60.

[145] Y. Hu, X. Wei, J. Yan, and H. Zhang, "Graphon mean-field control for cooperative multi-agent reinforcement learning", *arXiv preprint arXiv:2209.04808*, pp. 1–25, 2022.

[146] K. Cui and H. Koeppl, "Learning graphon mean field games and approximate Nash equilibria", in *The 10th International Conference on Learning Representations*, OpenReview.net, 2022, pp. 1–31.

[147] E. Bayraktar, S. Chakraborty, and R. Wu, "Graphon mean field systems", *The Annals of Applied Probability*, vol. 33, no. 5, pp. 3587–3619, 2023.

[148] P. E. Caines and M. Huang, "Graphon mean field games and the GMFG equations: ε-Nash equilibria", in *58th Conference on Decision and Control (CDC)*, IEEE, 2019, pp. 286–292.

[149] P. E. Caines and M. Huang, "Graphon mean field games and their equations", *SIAM Journal on Control and Optimization*, vol. 59, no. 6, pp. 4373–4399, 2021.

[150] C. Fabian, K. Cui, and H. Koeppl, "Learning sparse graphon mean field games", *arXiv preprint arXiv:2209.03880*, pp. 1–32, 2022.

[151] N. Gast, "The power of two choices on graphs: The pair-approximation is accurate?", *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 2, pp. 69–71, 2015.

[152] D. Rutten and D. Mukherjee, "Mean-field analysis for load balancing on spatial graphs", *arXiv preprint arXiv:2301.03493*, pp. 1–27, 2023.

[153] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of PPO in cooperative multi-agent games", *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 611–24 624, 2022.

[154] F. Christianos, L. Schäfer, and S. Albrecht, "Shared experience actor-critic for multi-agent reinforcement learning", *Advances in Neural Information Processing Systems*, vol. 33, pp. 10 707–10 717, 2020.

[155] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht, "Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks", in *Proceedings NeurIPS Datasets and Benchmarks*, MIT Press, 2021, pp. 15–19.

[156] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms", *arXiv preprint arXiv:1707.06347*, 2017.

[157] T. Ni, B. Eysenbach, and R. Salakhutdinov, "Recurrent model-free RL can be a strong baseline for many POMDPs", in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162, PMLR, 2022, pp. 16 691–16 723.

[158] H. Habibian and A. Patooghy, "Fault-tolerant routing methodology for hypercube and cube-connected cycles interconnection networks", *The Journal of Supercomputing*, vol. 73, no. 10, pp. 4560–4579, 2017.

[159] J. F. Shortle, J. M. Thompson, D. Gross, and C. M. Harris, *Fundamentals of queueing theory*. John Wiley & Sons, 2018, vol. 399.

[160] E. Koenigsberg, "Twenty five years of cyclic queues and closed queue networks: A review", *Journal of the Operational Research Society*, vol. 33, no. 7, pp. 605–619, 1982.

[161] F. P. Preparata and J. Vuillemin, "The cube-connected cycles: A versatile network for parallel computation", *Communications of the ACM*, vol. 24, no. 5, pp. 300–309, 1981.

[162] M. Deveci, K. D. Devine, K. Pedretti, M. A. Taylor, S. Rajamanickam, and Ü. V. Çatalyürek, "Geometric mapping of tasks to processors on parallel computers with mesh or torus networks", *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 9, pp. 2018–2032, 2019.

[163] R. Glantz, H. Meyerhenke, and A. Noe, "Algorithms for mapping parallel processes onto grid and torus architectures", in *23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, IEEE, 2015, pp. 236–243.

[164] M. Newman, *Networks*. Oxford University Press, 2018.

[165] B. K. Fosdick, D. B. Larremore, J. Nishimura, and J. Ugander, "Configuring random graph models with fixed degree sequences", *SIAM Review*, vol. 60, no. 2, pp. 315–355, 2018.

[166] M. Ostilli, "Cayley trees and bethe lattices: A concise analysis for mathematicians and physicists", *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 12, pp. 3417–3423, 2012.

[167] G. Szabó and I. Borsos, "Evolutionary potential games on lattices", *Physics Reports*, vol. 624, pp. 1–60, 2016.

[168] Y. Li, *Networked-MARL*, `https://github.com/yihenglin97/Networked-MARL`, 2021.

[169] E. Cinlar, *Introduction to stochastic processes*. Courier Corporation, 2013.

[170] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, "Monte Carlo tree search: A review of recent modifications and applications", *Artificial Intelligence Review*, vol. 56, no. 3, pp. 2497–2562, 2023.

[171] A. Bourki, G. Chaslot, M. Coulm, V. Danjean, H. Doghmen, J.-B. Hoock, T. Hérault, A. Rimmel, F. Teytaud, O. Teytaud, *et al.*, "Scalability and parallelization of Monte-Carlo tree search", in *Computers and Games: 7th International Conference, CG 2010, Kanazawa, Japan, September 24-26, 2010, Revised Selected Papers 7*, Springer, 2011, pp. 48–58.

[172] A. Mahajan and D. Teneketzis, "Multi-armed bandit problems", in *Foundations and applications of sensor management*, Springer, 2008, pp. 121–151.

[173] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning", in *Nineteenth International Conference on Machine Learning*, 2002, pp. 267–274.

[174] J. K. Terry, N. Grammel, A. Hari, L. Santos, and B. Black, "Revisiting parameter sharing in multi-agent deep reinforcement learning", *arXiv preprint arXiv:2005.13625*, pp. 1–18, 2020.

[175] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning", in *Autonomous Agents and Multiagent Systems (AAMAS)*, Springer, 2017, pp. 66–83.

CURRICULUM VITÆ

ANAM TAHIR

PERSONAL INFORMATION

HOMEPAGE https://github.com/AnamTahir7

EDUCATION

Master of Science (M.Sc.) 2014-2018
Information and Communication Engineering
Technische Universität Darmstadt, Darmstadt, Germany

Bachelor of Science (B.Sc.) 2008-2012
Information and Communication System Engineering
National University of Science and Technology, Pakistan

WORK EXPERIENCE

PostDoctorate 2024-current

Networks and Communication Systems Lab
University of Duisburg-Essen, Germany

Research Associate (PhD) 2018-2023
Self Organizing Systems Lab
Technische Universität Darmstadt,Darmstadt, Germany

Internship 04/2016 - 09/2016
AGT International Darmstadt, Germany

ERKLÄRUNG LAUT PROMOTIONSORDNUNG

§ 8 Abs. 1 lit. c PromO
Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§ 8 Abs. 1 lit. d PromO
Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§ 9 Abs. 1 PromO
Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§ 9 Abs. 2 PromO
Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 15. Januar 2024